

Introduction to Intel oneAPI for HPC and AI-DL

OneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

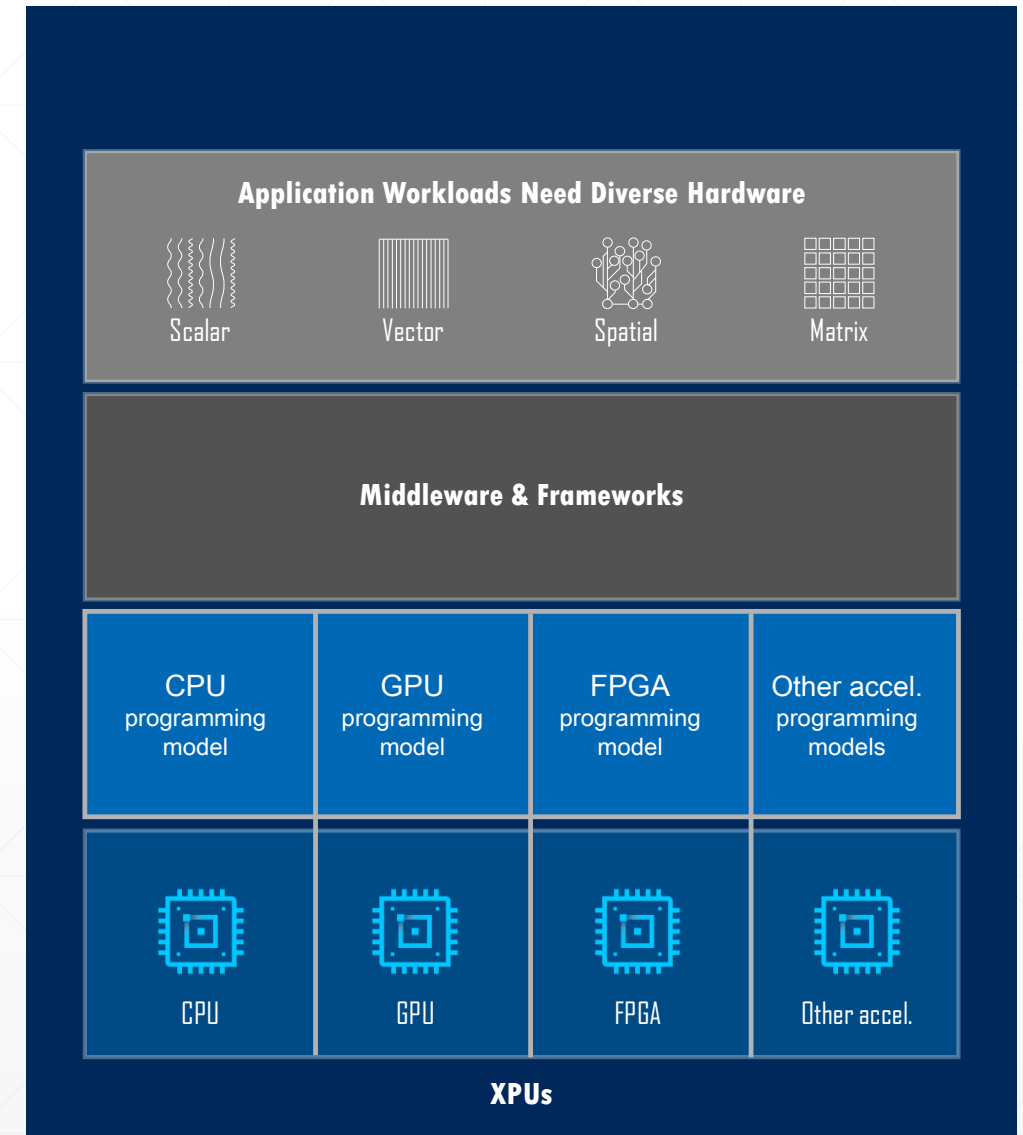
2020. 1. 21.
MOASYS

Outline

- What is oneAPI?
- Intel oneAPI DPC++ Programming
- High-performance computing (HPC)
- Machine learning, Deep learning, and analytics

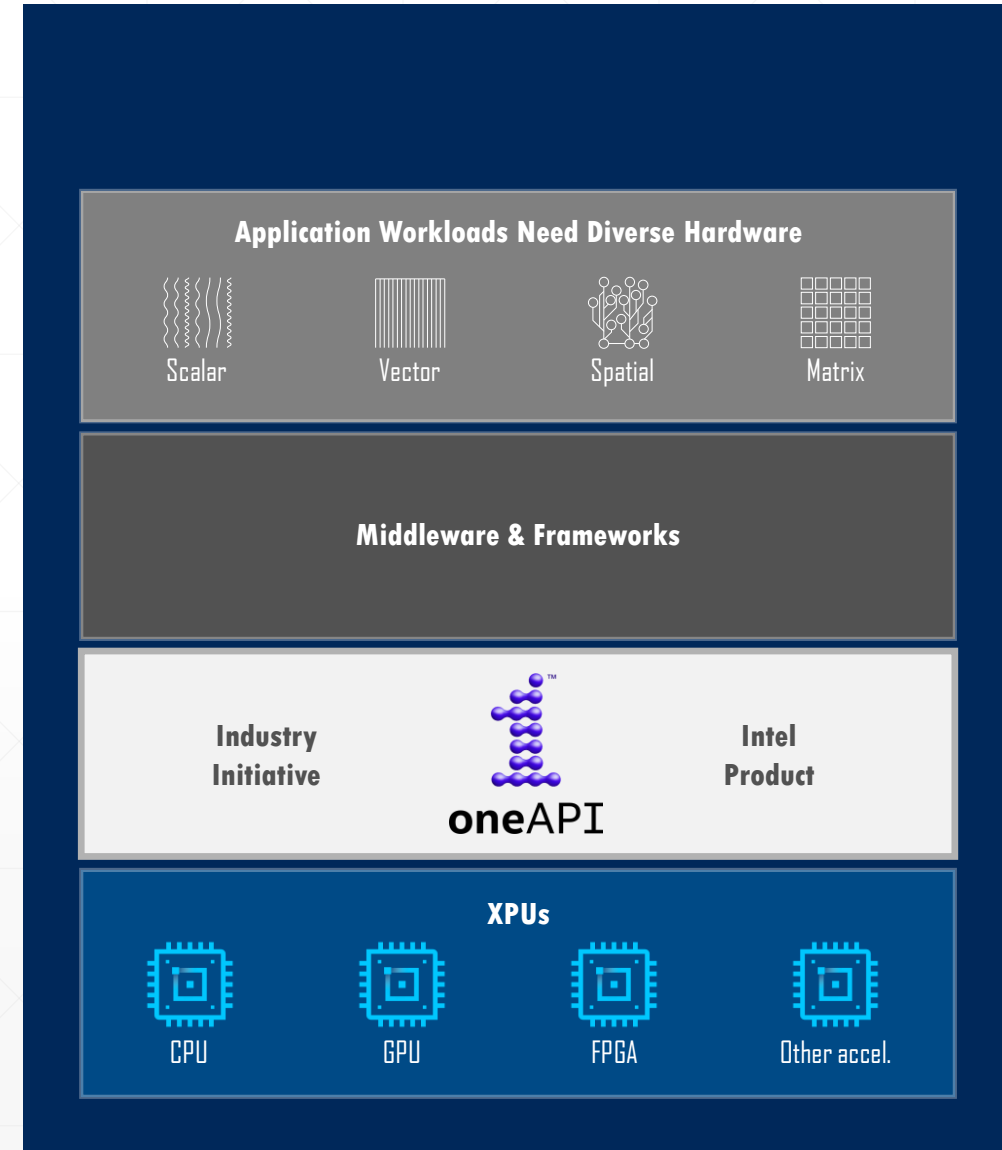
Programming Challenges for Multiple Architectures

- 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



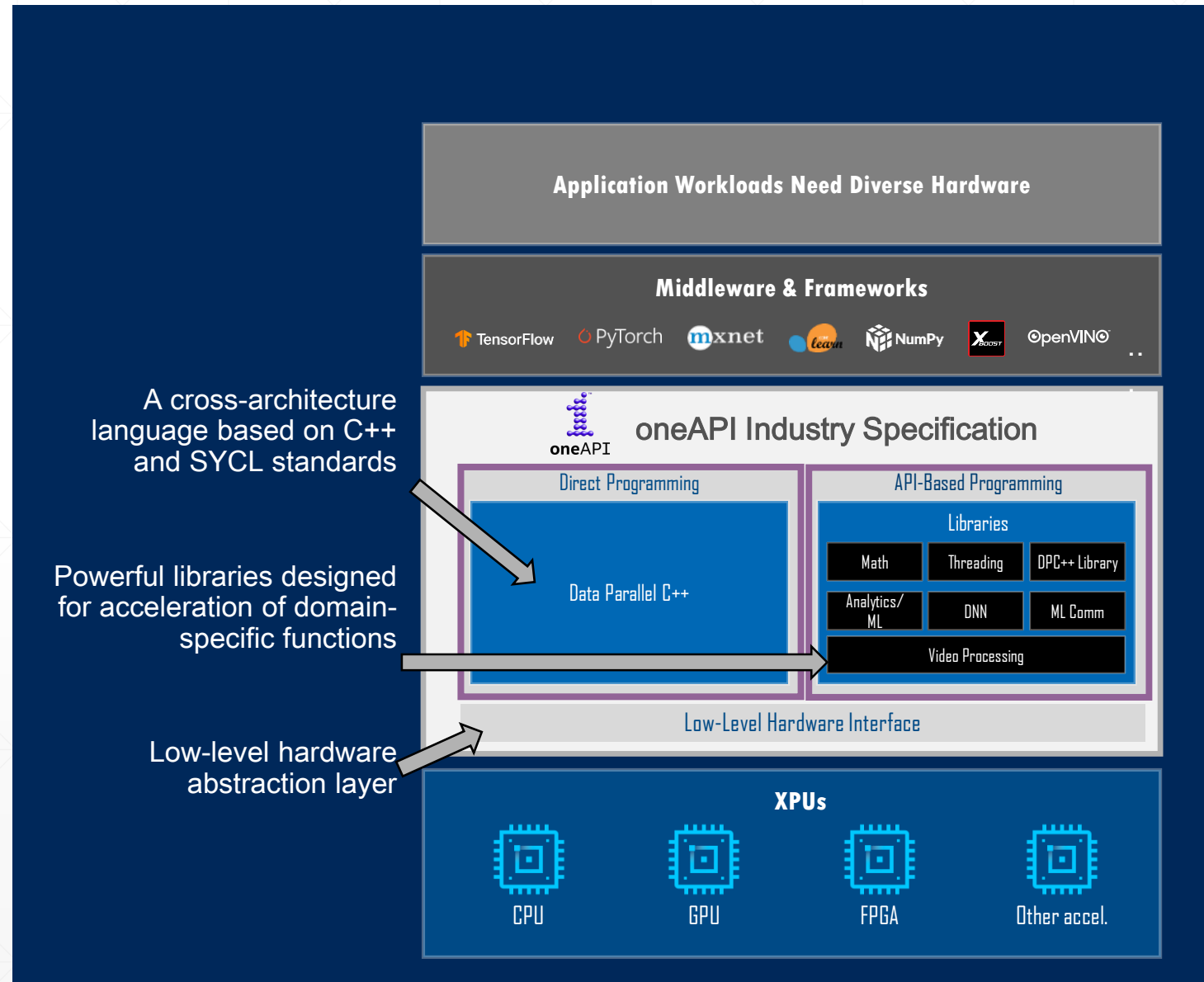
oneAPI: One Programming Model for Multiple Architectures and Vendors

- 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

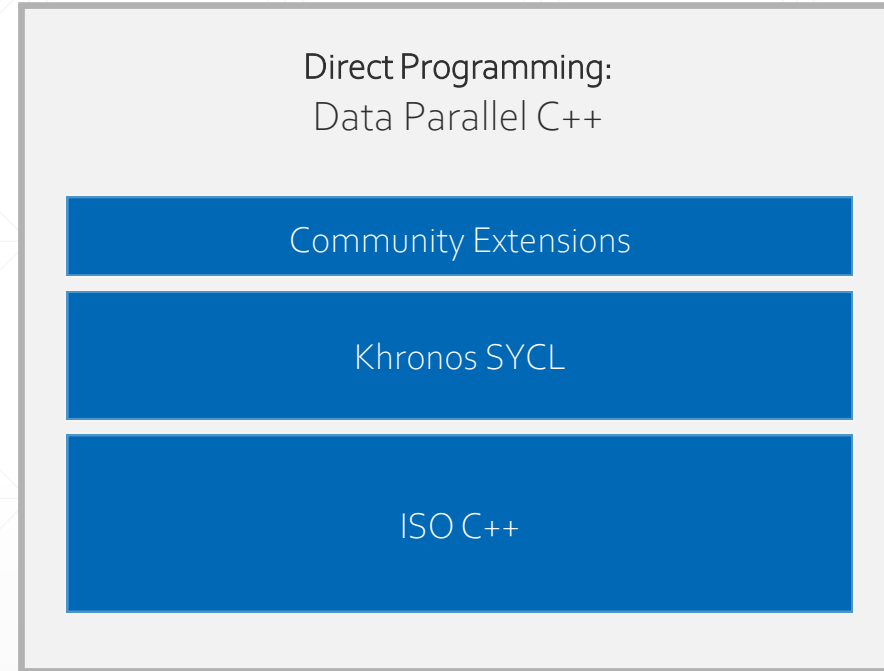
- Open to promote community and industry collaboration
- Enables code reuse across architectures and vendors



Data Parallel C++: Standards-based, Cross-architecture Language

- 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
 - Delivers C++ productivity benefits, using common, familiar C and C++ constructs
 - 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

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



Intel oneAPI DPC++ Programming

- DEVICE SELECTOR
 - The **device_selector** class enables the runtime selection of a particular device to execute kernels based upon user-provided heuristics.
 - The following code sample shows use of the standard device selectors(**default_selector**, **cpu_selector**, **gpu_selector**...) and a derived **device_selector**

```
default_selector selector;  
// host_selector selector;  
// cpu_selector selector;  
// gpu_selector selector;  
queue q(selector);  
std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;
```

Intel oneAPI DPC++ Programming

- QUEUE
 - A queue **submits command groups** to be executed by the SYCL runtime
 - Queue is a mechanism where work is submitted to a device.
 - A Queue map to one device and multiple queues can be mapped to the same device.

```
queue q;
```

```
q.submit([&](handler& h) {  
    // COMMAND GROUP CODE  
});
```


Intel oneAPI DPC++ Programming

- KERNEL

- The kernel class encapsulates methods and data for executing code on the device when a command group is instantiated
- Kernel object is not explicitly constructed by the user
- Kernel object is constructed when a kernel dispatch function, such as `parallel_for`, is called

```
q.submit([&](handler& h) {  
    h.parallel_for(range<1>(N), [=](id<1> i) {  
        A[i] = B[i] + C[i]);  
    });  
});
```

Intel oneAPI DPC++ Programming

- Parallel Kernels
 - Parallel Kernel allows multiple instances of an operation to execute in parallel.
 - Useful to offload parallel execution of a basic **for-loop** in which each iteration is completely independent and in any order.
 - Parallel kernels are expressed using the **parallel_for** function

for-loop in CPU application

```
for(int i=0; i < 1024; i++){  
    a[i] = b[i] + c[i];  
});
```



Offload to accelerator using **parallel_for**

```
h.parallel_for(range<1>(1024), [=](id<1> i){  
    A[i] = B[i] + C[i];  
});
```

Intel oneAPI DPC++ Programming

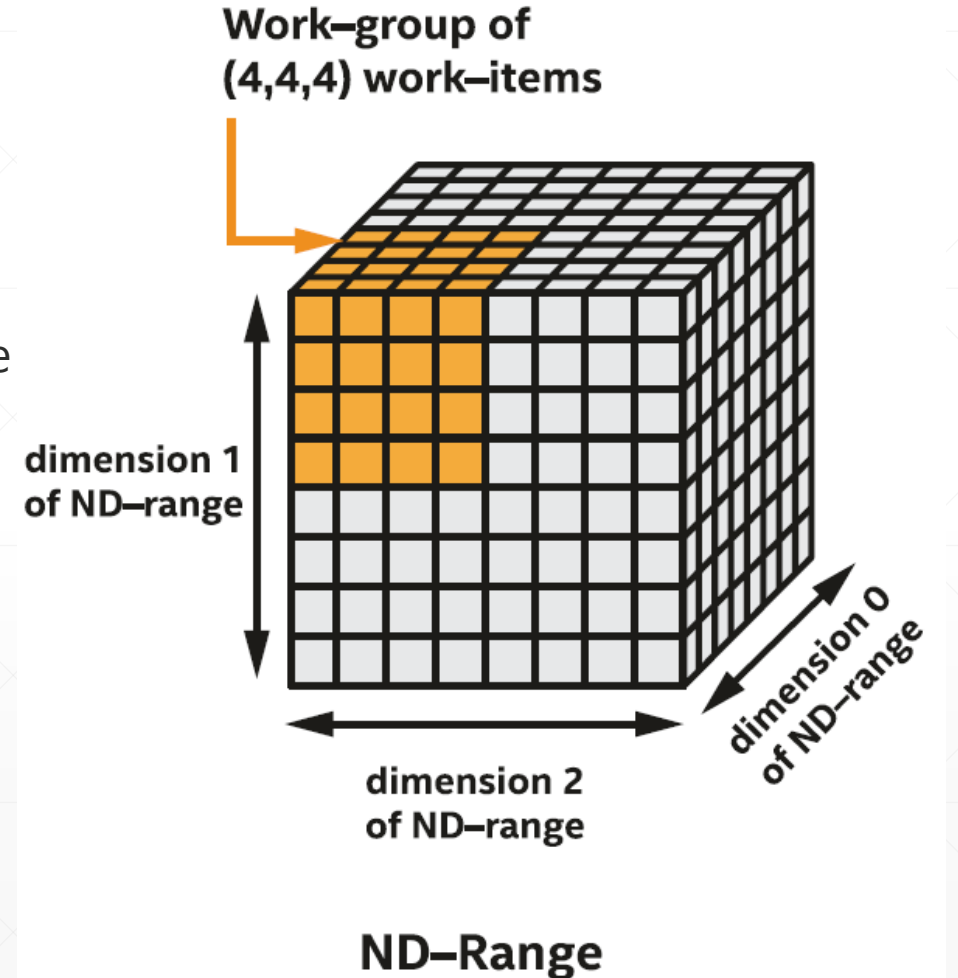
- Basic Parallel Kernels
 - The functionality of basic parallel kernels is exposed via range, id and item classes
 - **range** class used to describe the iteration space of parallel execution
 - **id** class is used to index an individual instance of a kernel in a parallel execution
 - **item** class represents an individual instance of a kernel function, exposes additional functions to query properties of the execution range

```
h.parallel_for(range<1>(1024), [=](id<1> idx){  
    // CODE THAT RUNS ON DEVICE  
});
```

```
h.parallel_for(range<1>(1024), [=](item<1> item){  
    auto idx = item.get_id();  
    auto R = item.get_range();  
    // CODE THAT RUNS ON DEVICE  
});
```

Intel oneAPI DPC++ Programming

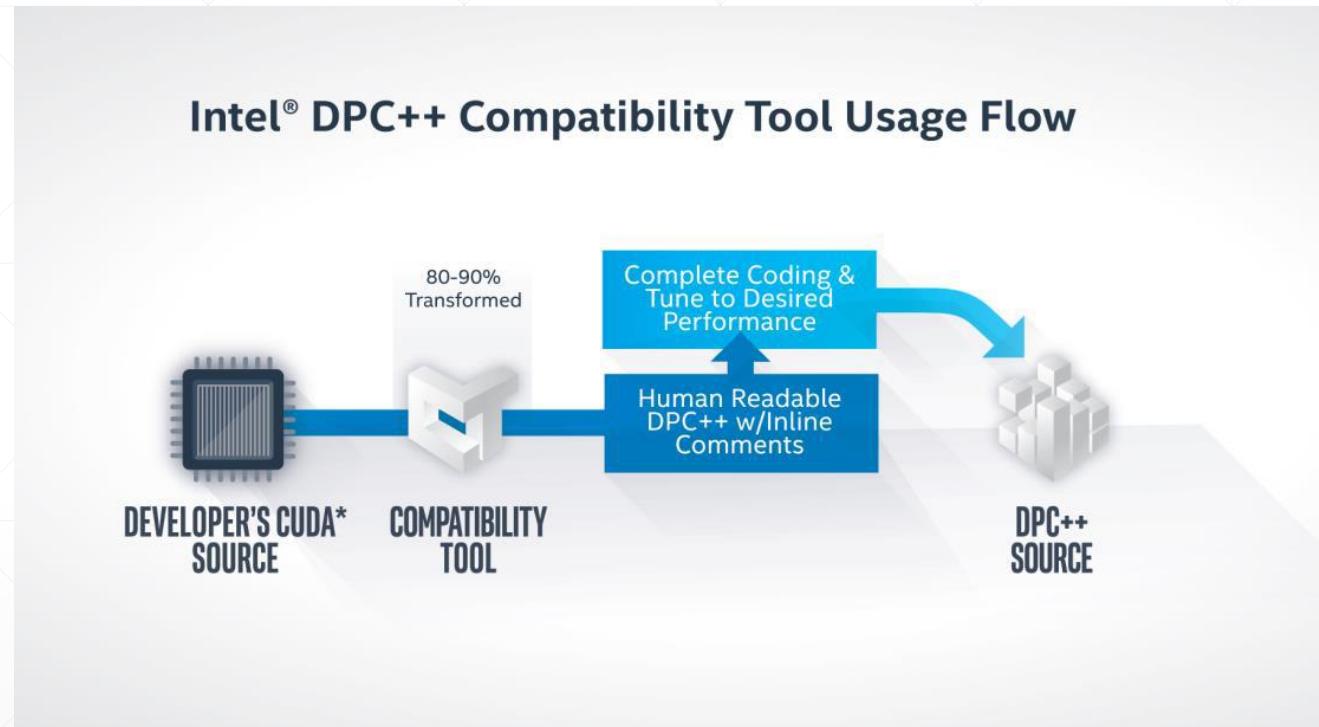
- ND-Range Kernels
 - Basic Parallel Kernels are easy way to parallelize a for-loop **but does not allow** performance optimization at hardware level.
 - **ND-Range kernel** is another way to expresses parallelism which enable **low level performance tuning** by providing access to local memory and mapping executions to compute units on hardware.



Intel oneAPI DPC++ Programming

- Intel DPC++ Compatibility Tool
 - Minimizes Code-Migration Time
 - Assists developers migrating code written in CUDA to DPC++ by generating DPC++ code wherever possible
 - Expect up to 80-90% of code to migrate automatically
 - Inline comments are provided to help developer complete code

```
$ dpct vector_add.cu  
$ dpcpp vector_add.dp.cpp
```



Intel oneAPI DPC++ Programming

```
#include <cuda.h>
#include <stdio.h>
#define VECTOR_SIZE 256
```

```
#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
#include <stdio.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];
}
```

Idx	0	1	2	...	255
A	1	2	3	...	256
B	1	2	3	...	256
C	2	4	6	...	512

```
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()
{
    float *d_A, *d_B, *d_C;
```

```
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;
```

```
cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));
cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));
cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));
```

```
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);
```

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

```
q_ct1.submit([&](sycl::handler &cgh) {
    cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, VECTOR_SIZE), // global
    sycl::range<3>(1, 1, VECTOR_SIZE)), // local
    [=](sycl::nd_item<3> item_ct1) {
        VectorAddKernel(d_A, d_B, d_C, item_ct1);
    });
});
```

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

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

```
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);
```

```
for (int i = 0; i < VECTOR_SIZE; i++) {
    if (i % 16 == 0) {
        printf("\n");
    }
    printf("%f ", Result[i]);
}

return 0;
}
```

```
for (int i = 0; i < VECTOR_SIZE; i++) {
    if (i % 16 == 0) {
        printf("\n");
    }
    printf("%f ", Result[i]);
}

return 0;
}
```

Header

Kernel definition
 CUDA: 1D
 SYCL: 3D (general)

Device selection
Queue initialization

USM allocation

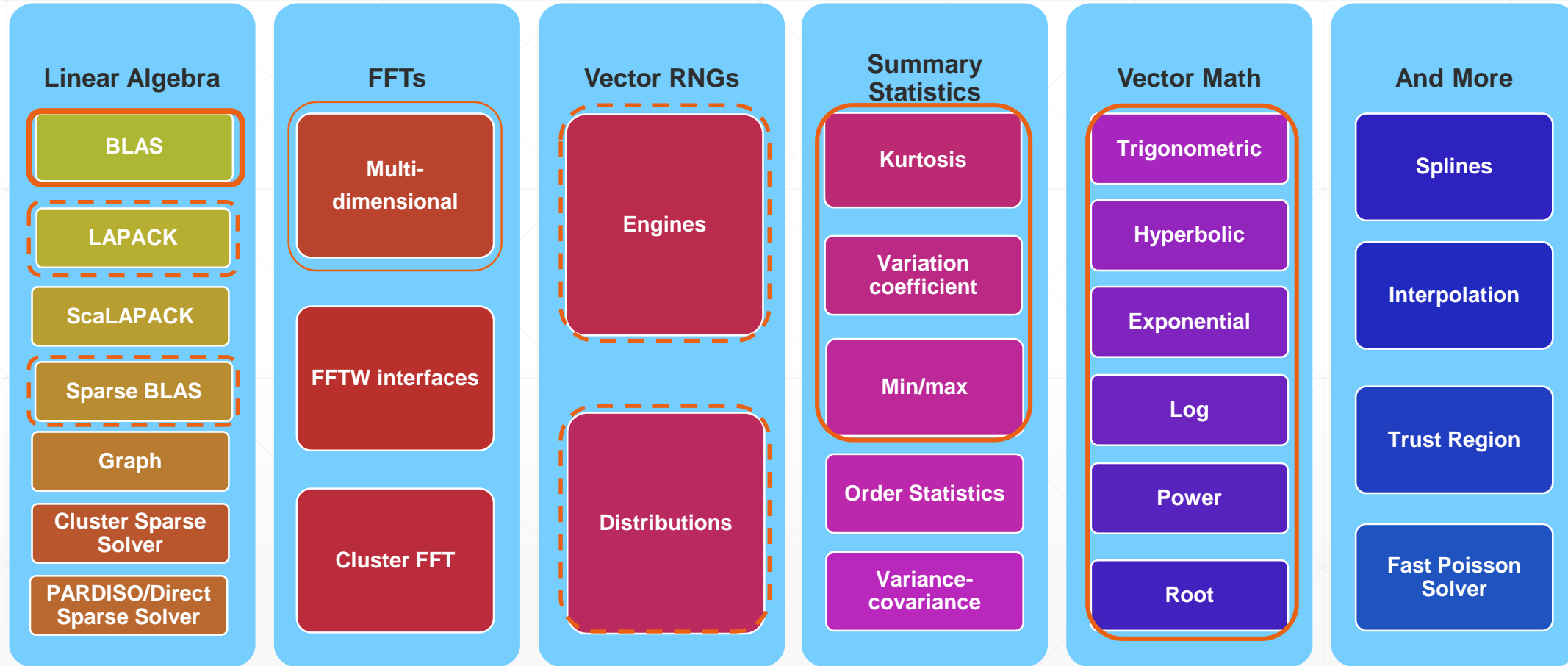
Kernel execution
 CUDA: 1 thread block
 SYCL: 1 work group

Data movement
Deallocation


Generic C code
 No migration

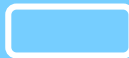
Intel oneAPI Libraries

- Intel oneMKL



 Beta Intel® Processor Graphics Gen9/Gen12 & Intel discrete GPU support

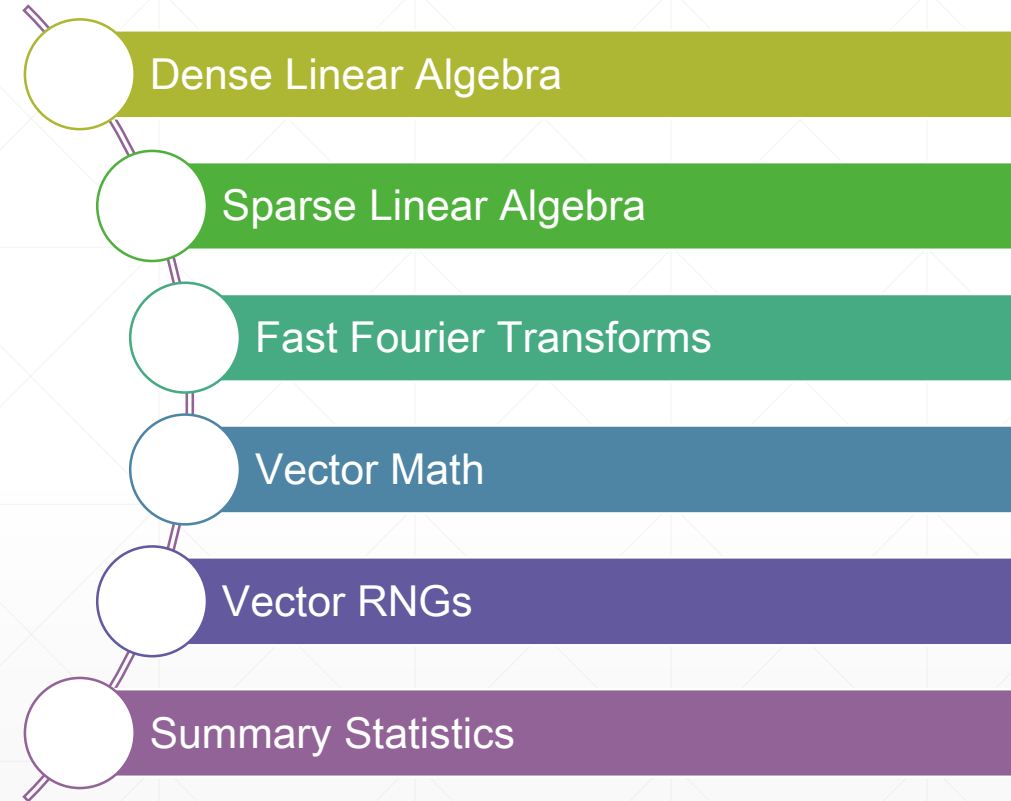
 Limited - Intel® Processor Graphics Gen9/Gen12 & Intel discrete GPU (see release notes)

 CPU C/Fortran support

Intel oneAPI Libraries

- Intel oneMKL
 - **Accelerate Math Processing, Increase Application Performance**
 - Language support for DPC++ and Intel® C & Fortran compilers
 - Available at no cost and royalty-free
 - Great performance with minimal effort
 - Full support for CPUs and select support for Intel® Processor Graphics Gen9, Gen12, and discrete Intel® GPUs
 - Speeds computations for scientific, engineering, and financial applications by providing highly optimized, threaded, and vectorized math functions
 - Provides key functionality for dense and sparse linear algebra (BLAS, LAPACK, PARDISO), FFTs, vector math, summary statistics, splines, and more
 - Dispatches optimized code for each processor automatically without the need to branch code
 - Optimized for single-core vectorization and cache utilization
 - Automatic parallelism for multi-core CPUs, GPUs, and scales from core to clusters

Intel® oneAPI Math Kernel Library offers



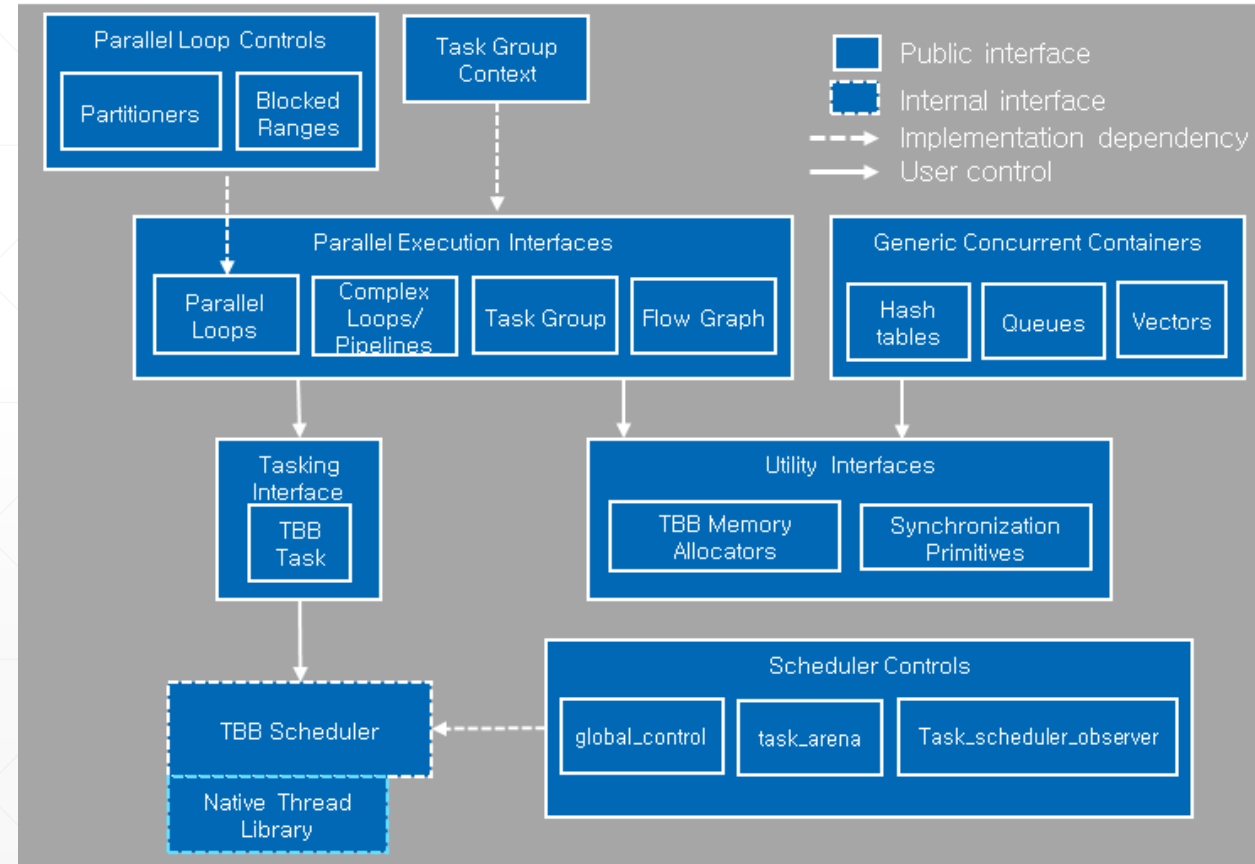
Intel oneAPI Libraries

- Intel oneTBB
 - **Advanced Scaling for Fast Applications**
 - Flexible C++ Library for Parallelism
 - An easy way for developers to express parallelism in applications without the need to have deep hardware knowledge
 - Future Proof & Scale Application Performance
 - Effectively parallelize and scale performance for computationally intensive workloads on current and future platforms
 - Compatible with Other Threading Packages
 - Keep legacy code as-is and use oneTBB for new implementations. Seamlessly coexist with other threading packages
 - Simplified and Enhanced Application Composability
 - Create composable, scalable parallelism on the CPU, and extendable with enhanced handling of accelerators

Intel oneAPI Libraries

oneTBB Architecture Overview

- A Collection of Building Blocks to Develop Highly Scalable Threaded Applications
- Includes high-level parallel execution interfaces
 - **Parallel Loops:** parallel_for, parallel_reduce, etc.
 - **Complex Algorithms:** pipelines, task groups
 - **Flow Graph:** Expressing data flow independent graphs
- All build on top of TBB tasks, these tasks execute on top of the TBB scheduler
- Scheduler controls & parallel loops controls to tightly control performance
- Concurrent Containers - Queues, Vectors, etc. are thread safe and thread friendly
- Scalable memory allocator, synchronization primitives



Intel® MPI Library: Flexible, Efficient and Scalable Cluster Messaging

Optimized MPI Application Performance

- Application-specific tuning
- Automatic tuning
- Support for latest Intel® Xeon® Scalable Processors

Lower Latency and Multi-vendor Interoperability

- Industry-leading latency
- Performance-optimized support for the fabric capabilities through OpenFabrics Interfaces (OFI)

Faster MPI Communication

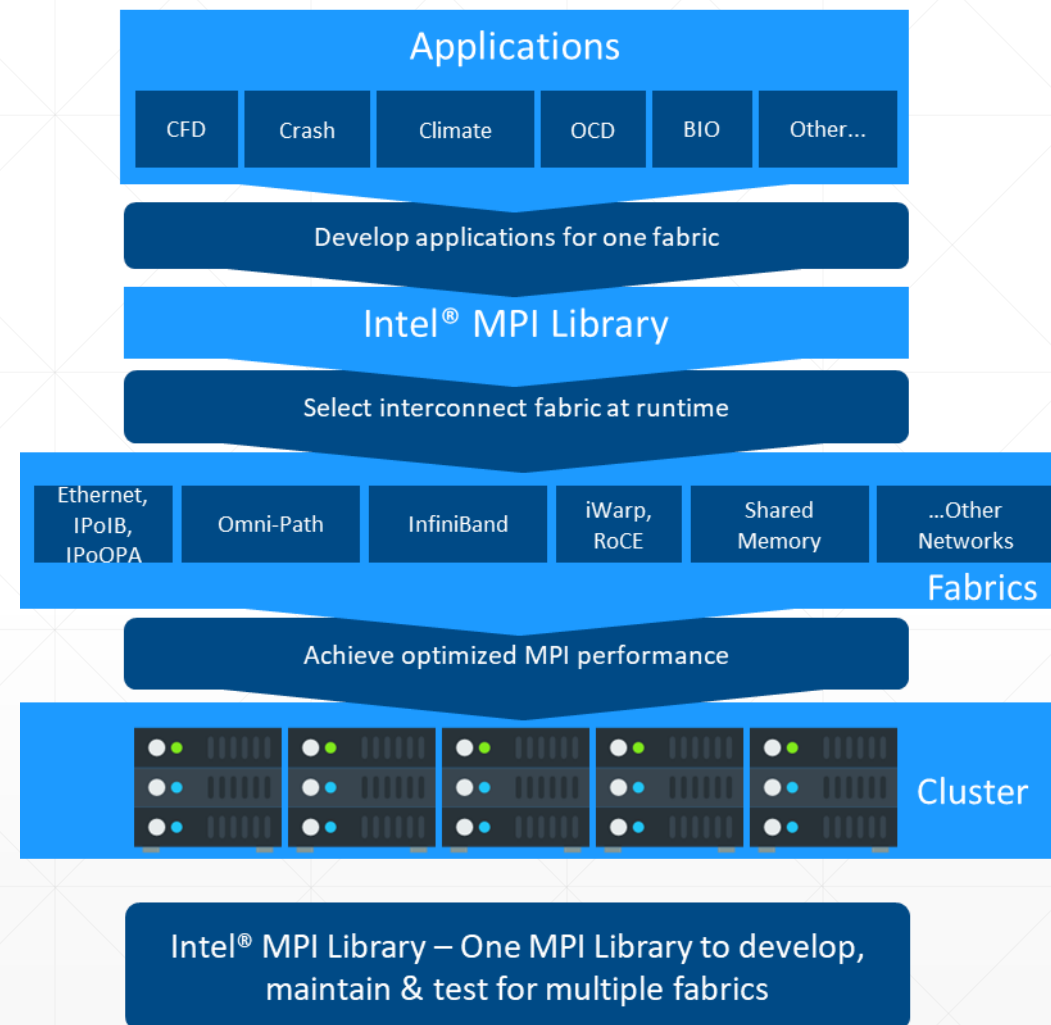
- Optimized collectives

Sustainable scalability

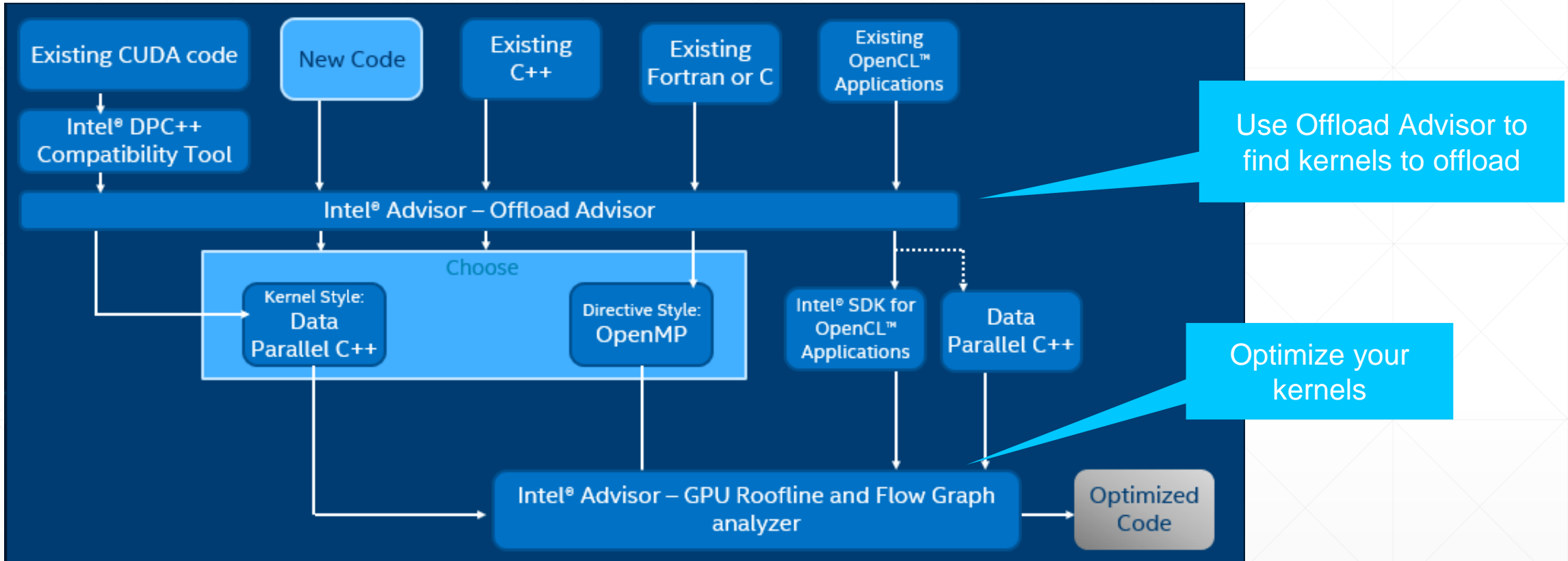
- Native InfiniBand interface support allows
- Lower latencies, higher bandwidth, reduced memory requirements

Key Updates

- Intel® GPU pinning support
- Distributed Asynchronous Object Storage (DAOS) support
- Intel® Xeon® Platinum processor 92XX optimizations
- Mellanox ConnectX: 3/4/5/6 (FDR/EDR/HDR) support enhancements



Using Intel Analysis Tools to Increase Performance



Intel® Advisor
Intel® VTune™ Profiler
Intel® Inspector
Intel® Trace Analyzer & Collector
Intel® Cluster Checker

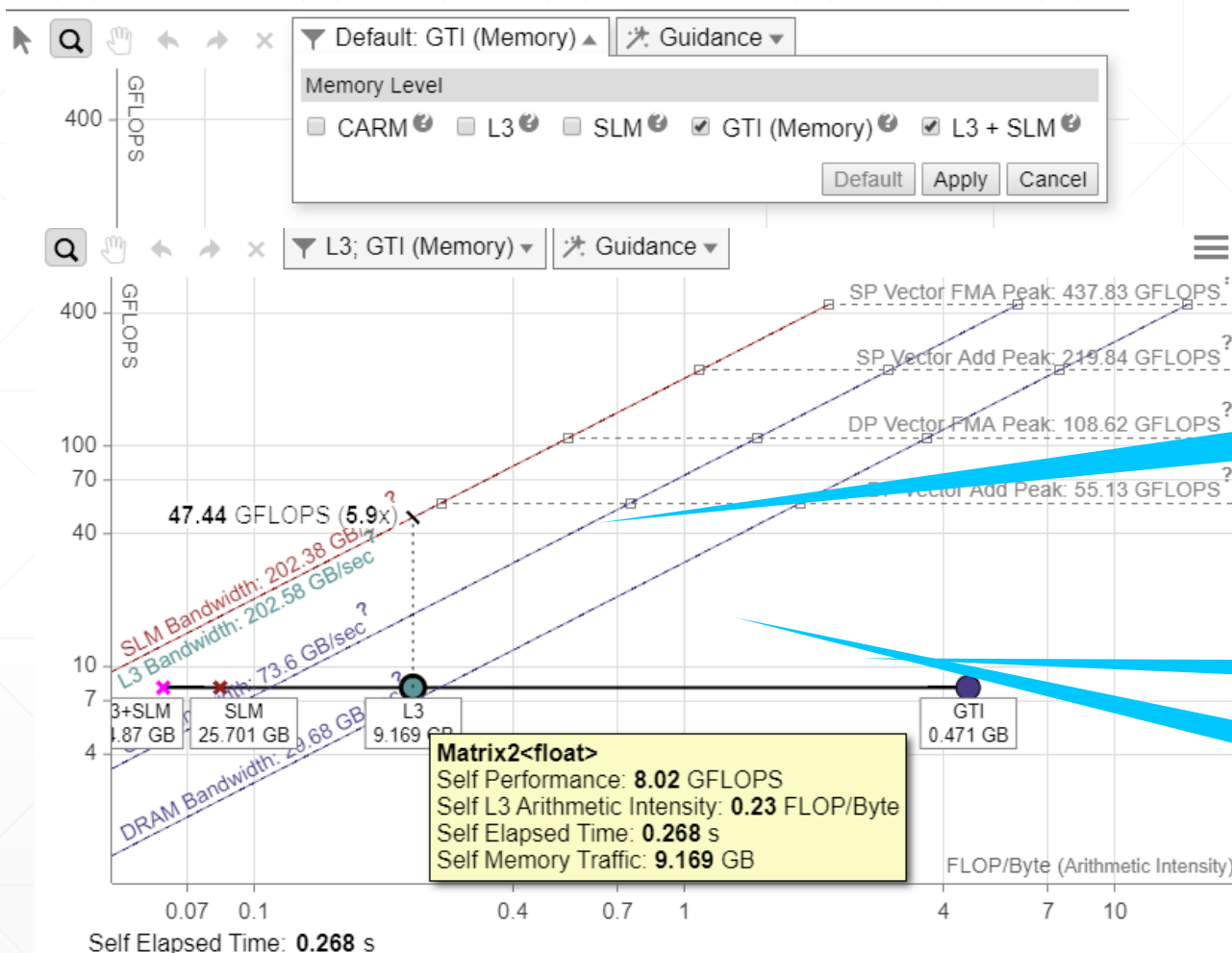
Intel® Advisor - Offload Advisor: Efficiently offload code to GPUs



Starting from an optimized binary (running on CPU)

- Identify high-impact opportunities to offload
- Detect bottlenecks and key bounding factors
- Get your code ready even before you have the hardware by modeling performance, headroom, and bottlenecks

Intel® Advisor - GPU Roofline: Find effective optimization strategies



Configure levels to display

Shows performance headroom for each loop

Likely bottlenecks

Suggests optimization next steps

Quickly find & fix performance bottlenecks, realize all the value of your hardware

- See performance headroom against hardware limitations
- Determine performance optimization strategy by identifying bottlenecks and which optimizations will pay off the most
- Visualize optimization progress

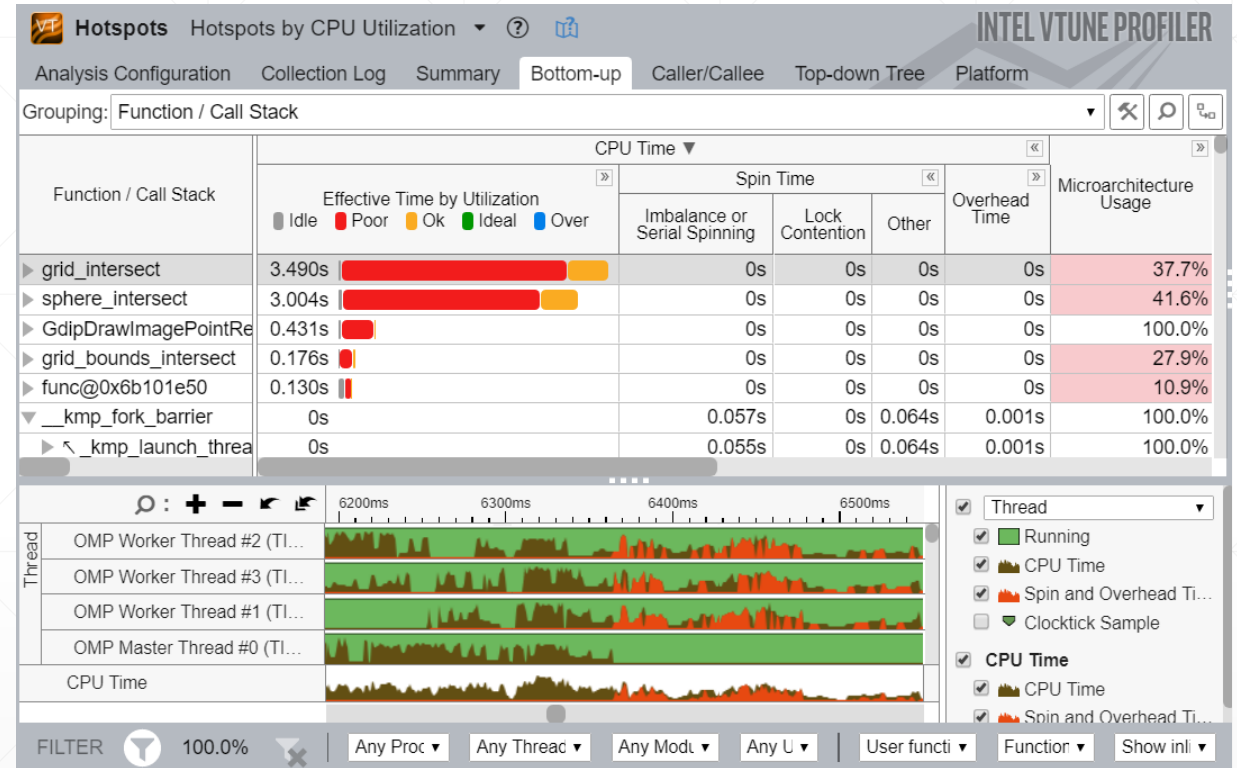
Intel® VTune™ Profiler: Analyze and Tune Application Performance

Save Time Optimizing Code

- Accurately profile C, C++, Fortran, Python, Go, Java
- Optimize CPU, threading, memory, cache, storage
- Take advantage of [Priority Support†](#)

What's New in 2021.1 Release (partial list)

- Production release of Platform Profiler
- Design & optimize for Intel® Optane™ DC Persistent Memory
- Application Performance Snapshot
 - Add communication-pattern diagnosis
 - Profile more ranks
- Linux
 - Extensive perf-enables analysis without adding drivers



Intel® VTune™ Profiler: GPU Architecture Summary

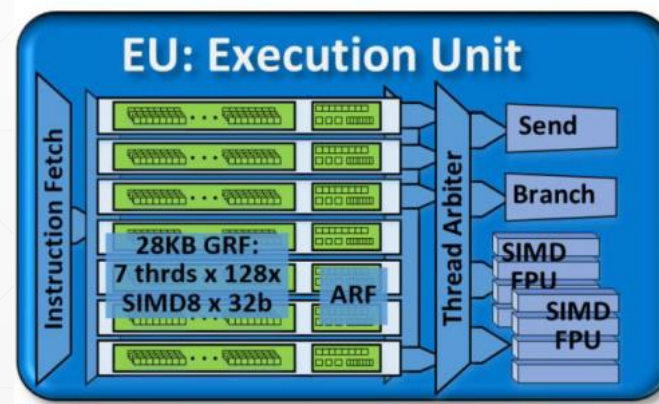
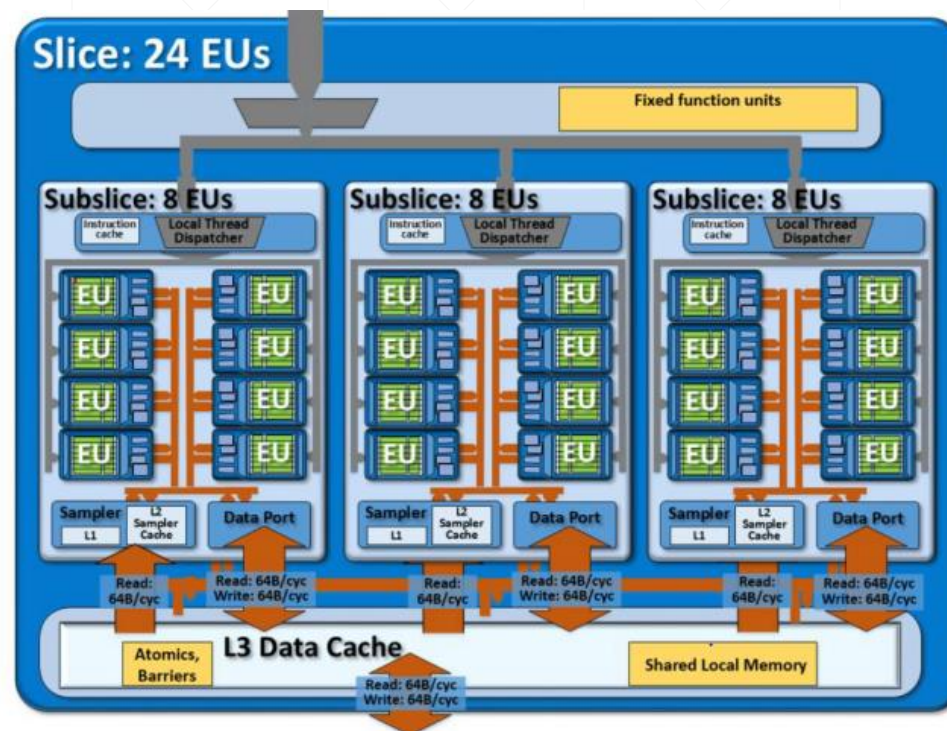
Collection and Platform Info

GPU

Name: Intel(R) UHD Graphics 620
Vendor: Intel Corporation
Driver: 27.20.100.8187
EU Count: 24
Max EU Thread Count: 7
Max Core Frequency: 1.1 GHz

GPU OpenCL Info

Version: OpenCL C 2.0
Max Compute Units: 24
Max Work Group Size: 256
Local Memory: 64 KB
SVM Capabilities: Fine-grained buffer with atomics



Intel Gen9 GPU details:

- 24EU x 7thr = 168 threads
- 128 General-Purpose Registry File (GRF) of 32 bytes
- 2 SIMD-4 FPU of 32-bit FP or INT data
- 16 MAD/cycle (ADD+MUL) x 2FPUs x SIMD-4
- 2 additional units: Branch and Send

Intel® VTune™ Profiler: Optimize Your GPU Usage

GPU Offload (Preview) GPU Offload (Preview) ▾ ? ⓘ

Analysis Configuration Collection Log Summary Graphics Platform

Elapsed Time: 2.017s

GPU Usage: 47.8%

Use this section to understand whether the GPU was utilized properly and which of the engines were utilized. Identify the amount of gaps in the GPU utilization that potentially could be loaded with some work. This metric is calculated for the engines that had at least one piece of work scheduled to them.

GPU Usage breakdown by GPU engines and work types.

GPU Engine / Packet Type	GPU Time	GPU Utilization
Render and GPGPU	0.964s	47.8%
Unknown	0.964s	47.8%

**N/A is applied to non-summable metrics.*

Packet Queue Depth Histogram

Packet Duration Histogram

Hottest GPU Computing Tasks

This section lists the most active computing tasks running on the GPU, sorted by the Total Time. Focus on the computing tasks flagged as performance-critical.

Computing Task	Total Time	Total Compute Time	Total Transfer Time (f)
Matrix<float>	3.980s	0.961s	3.019s
clEnqueueReadBufferRect	0.000s	0.000s	0.000s

**N/A is applied to non-summable metrics.*

GPU Compute/Media Hotspots (preview) GPU Compute/Media Hotspots (preview) ▾ ? ⓘ

Analysis Configuration Collection Log Summary Graphics Platform

Architecture Diagram Platform

GPU

GPU Execution Units Array: Active: 67.5%, Stalled: 32.5%, Idle: 0.0%, 94K Threads/s

Sampler: Busy: 0.0%, Bottleneck: 0.0%

L3: 8e+8 Misses/s, Miss Ratio 33.0%

GTI

System DRAM

SLM: 0.00 GB/s Read, 0.00 GB/s Write

147.41 GB/s Total

CPU Utilization: 0.0%

Grouping: Computing Task

Computing Task	Work Size		Total Time	Average Time	Computing Task		Data Transferred		EU Array		
	Global	Local			Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled
MatrixMultiply2	4096	4096	5.440s	5.440s	1	32			67.5%	32.5%	0.0%
clEnqueueReadBufferRect			0.012s	0.012s	1				35.9%	59.8%	4.4%
[Outside any task]									20.9%	10.4%	68.8%

Quickly Find & Fix Performance Bottlenecks

- Explore code execution on your platform's various CPU and GPU cores
- Identify whether your application is GPU- or CPU-bound

GPU offload:

- Identifying how effectively your application uses DPC++ or OpenCL kernels
- Exploring GPU usage and analyzing a software queue for GPU engines at each moment of time

GPU hotspots:

- Analyze the most time-consuming GPU kernels, characterize GPU usage based on GPU hardware metrics
- GPU code performance at the source-line level and kernel-assembly level

Intel® Inspector: Locate & Debug Threading, Memory Errors

Find and eliminate

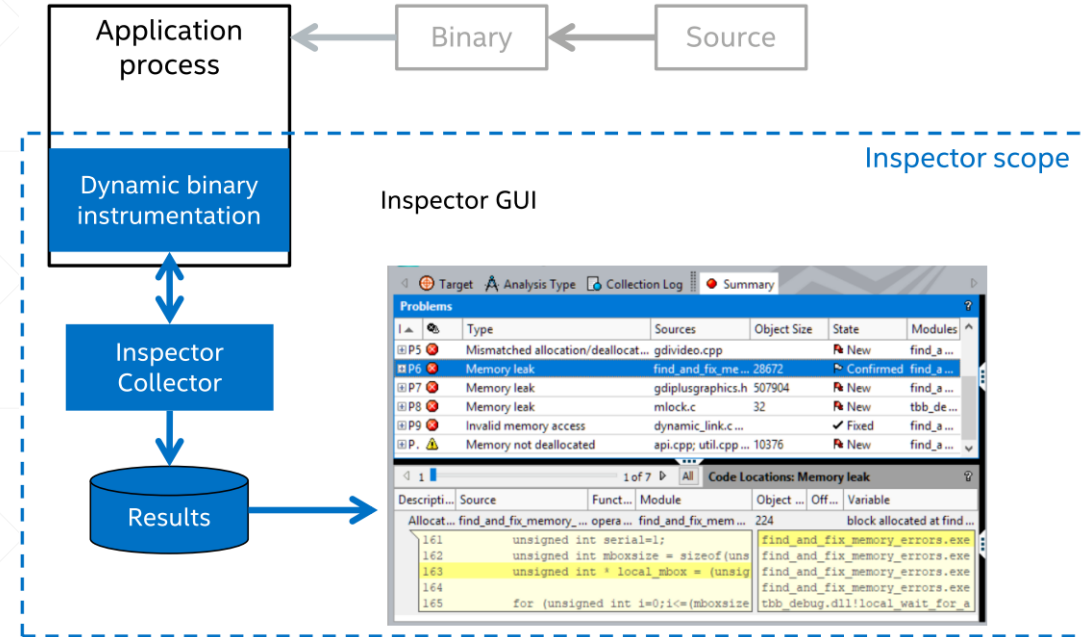
- Memory leaks, invalid access
- Persistent memory errors
- Races & deadlocks
- C, C++ and Fortran (or a mix)

Simple, Reliable, Accurate

- No special recompiles: use any build, any compiler
- Analyzes dynamically generated or linked code
- Inspects 3rd party libraries without source
- Command line for automated regression analysis

Faster Diagnosis with Debugger Breakpoints

- Breakpoint set just before the problem occurs
- Examine variables and threads with the debugger



Features	Memory Analysis	Threading Analysis	Persistence Memory
View context of problem	✓	✓	✓
Stack	✓	✓	✓
Multiple Contributing Source Locations	✓	✓	✓
Collapse multiple "sightings" to one error	✓	✓	✓
Suppress, Filter, Workflow Management	✓	✓	✓
Visual Studio Integration (Windows)	✓	✓	✓
Command line for automated tests	✓	✓	✓
Timeline visualization	✓	✓	
Memory growth during a transaction	✓		
Trigger debugger breakpoints	✓	✓	

Intel® Trace Analyzer & Collector: Profile & Analyze MPI Application

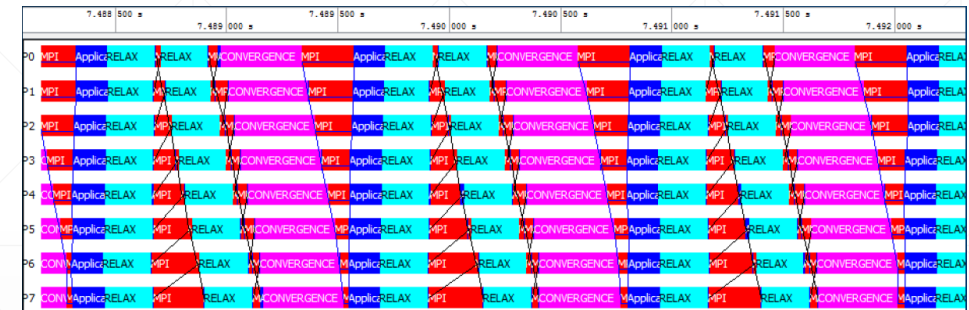
Mechanism	Advantages	Disadvantages
Run with <code>-trace</code>	Automatic collection of MPI calls No medication to source, compile or link	No collection of user code Requires dynamic link to MPI
Link with <code>-trace</code>	Automatic collection of MPI calls	No collection of user code Must be done at link time
Compile with <code>-tcollect</code>	Automatic collection of all functions entries/exits	Requires code re-compilation
Add API calls	Selective collection of desired code sections	Requires code modification

Top MPI functions

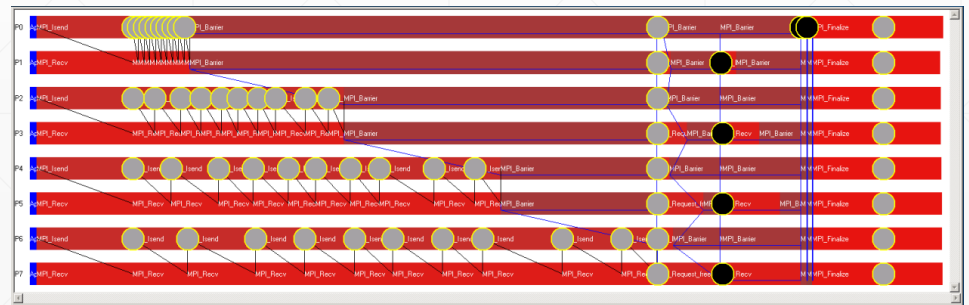
This section lists the most active MPI functions from all MPI calls in the application.



MPI event timeline



MPI correctness checking



● warnings ● errors

Debug MPI applications:

- GDB
- Allinea DDT
- gtool

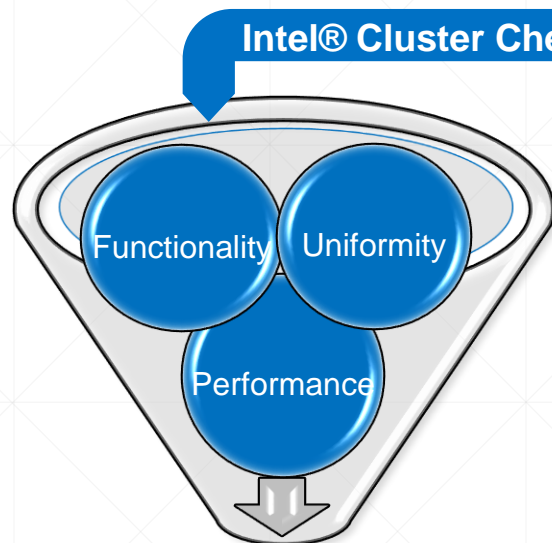
Scale MPI applications:

- Scale performance: perform on more nodes
- Scale forward: multi-core ready
- Scale efficiently: tune and debug on more nodes

Analyze, tune & optimize:

- Identify communication hotspots
- Evaluate profiling statistics and load balancing
- Visualize and understand parallel application behavior
- Analyze common MPI issues

Intel® Cluster Checker: Functionality, Uniformity & Performance Tests



Summary Output
& Log File

Functionality Tests	Uniformity Tests	Performance Tests
System-level <ul style="list-style-type: none">• Node• Connectivity• Cluster Validation <ul style="list-style-type: none">• Application platform compliance• Solution compliance	Hardware <ul style="list-style-type: none">• CPUs• Memory• Interconnect• Disks Software <ul style="list-style-type: none">• Installed packages and versions• numerous kernel and BIOS settings	Benchmarks <ul style="list-style-type: none">• DGEMM• HPCG• HPL• Intel® MPI Benchmarks• IOzone• STREAM
API Available for Integration		

Get Compact Reports, Find Problems, Validate Status

Comprehensive pre-packed cluster systems expertise out-of-the-box

- Suitable for HPC experts and those new to HPC
- Tests can be executed in selected groups on any subset of nodes

Certifying, Testing, and Troubleshooting Clusters

- Check over 100 characteristics that may affect operation and performance
- Catch issues, identify details or remedies
- Use for better uptime and productivity
- Free download, can be redistributed

Priority Support available for Intel® oneAPI Base & HPC Toolkits in commercial versions

Intel oneAPI AI Analytics Toolkit

Speed Up Development with open AI software



Machine learning ← → Deep learning



TOOLKITs

App Developers



libraries

Data Scientists



Kernels

Library Developers

Intel® Data Analytics Acceleration Library (DAAL)

Intel® Distribution for Python* (Sklearn*, Pandas*)

R (Cart, RandomForest, e1071)

Distributed (MLlib on Spark, Mahout)

ANALYTICS ZOO

Model Zoo

OpenVINO™

Intel Optimized Frameworks

TensorFlow BigDL Caffe ONNX mxnet PyTorch

More framework optimizations in progress...

Intel® Math Kernel Library (Intel® MKL)

Intel® oneAPI Collective Communication Library (Intel® oneCCL)

Deep Neural Networks Library (Intel® oneDNN)


CPU ■ GPU

Visit: www.intel.ai/technology

1 An open source version is available at: 01.org/openvintoolkit *Other names and brands may be claimed as the property of others. Developer personas show above represent the primary user base for each row, but are not mutually-exclusive. All products, computer systems, dates, and figures are preliminary based on current expectations, and are subject to change without notice.

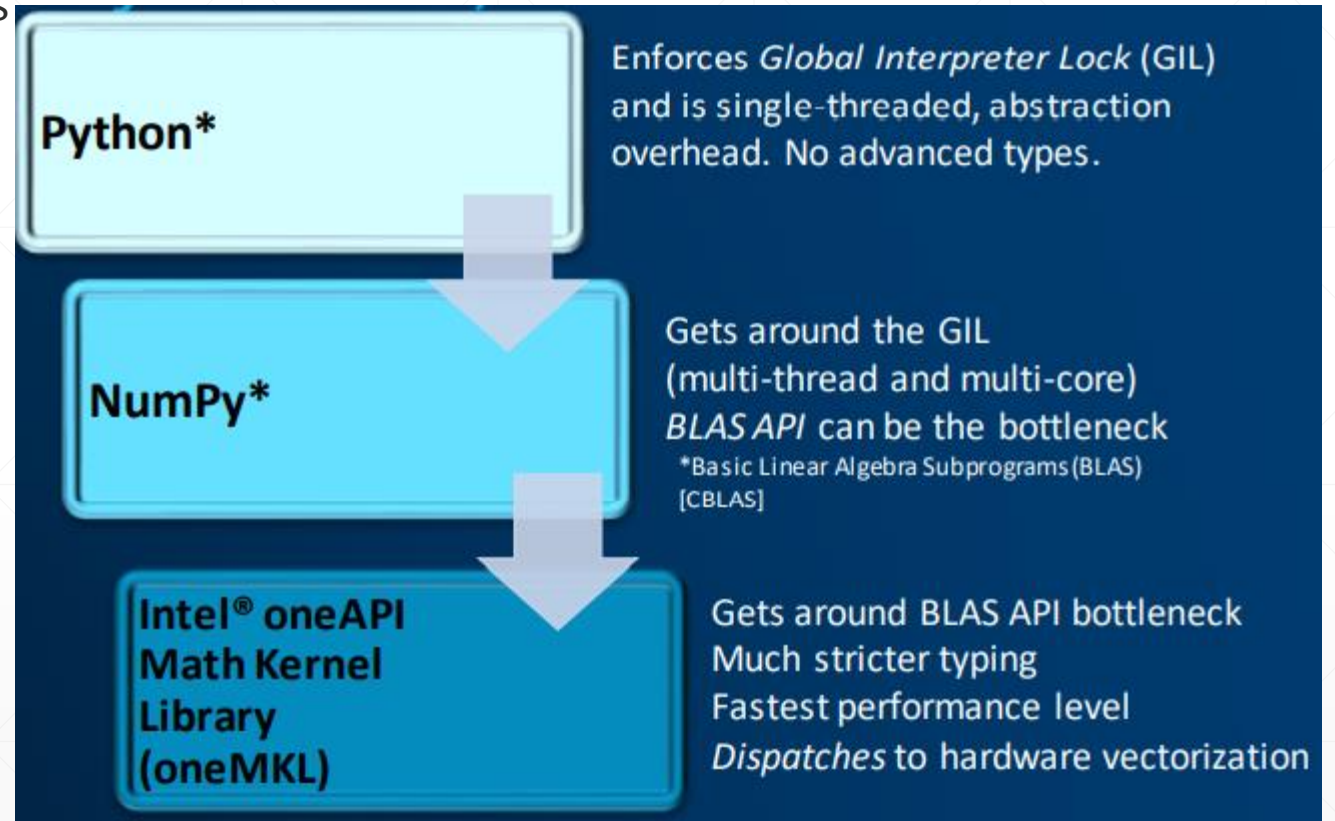
Accelerate libraries with Intel® Distribution for Python*

- High Performance Python for Scientific Computing, Data Analytics, Machine Learning

Faster Performance	Greater Productivity	Ecosystem compatibility
Performance Libraries, Parallelism, Multithreading, Language Extensions	Prebuilt & Accelerated Packages	Supports Python* 2.7 & 3.6, & 3.7 conda, pip
Accelerated NumPy*/SciPy*/scikit-learn* with oneMKL ¹ & oneDAL ² Data analytics, machine learning with scikit-learn, daal4py Optimized run-times Intel MPI®, Intel® TBB Scale with Numba* & Cython* Includes optimized mpi4py, works with Dask* & PySpark* Optimized for latest Intel® architecture	Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC & data analytics Drop-in replacement for existing Python* Usually NO code changes required! Conda build recipes included in packages Free download & free for all uses including commercial deployment	Compatible & powered by Anaconda*, supports conda & pip Distribution & individual optimized packages also available for Python* 2.7, 3.6, & 3.7 oneMKL accelerated NumPy*, and SciPy* now in Anaconda*! Optimizations upstreamed to main Python* trunk Commercial support through Intel® Parallel Studio XE
Intel® Architecture Platforms		
Operating System: Windows*, Linux*, MacOS ¹ *		

Performance Optimization

- The layers of quantitative Python*
 - The Python* language is interpreted and has many type checks to make it flexible
 - Each level has various tradeoffs; NumPy* value proposition is immediately seen
 - For best performance, escaping the Python layer early is best method



Installing Intel® Distribution for Python* 2020

Standalone Installer

Download full installer from

<https://software.intel.com/en-us/intel-distribution-for-python>

Anaconda.org

[Anaconda.org/intelchannel](https://anaconda.org/intelchannel)

```
> conda config --add channels intel  
> conda install intelpython3_full  
> conda install intelpython3_core
```

PyPI

```
pip install intel-numpy intel-scipy intel-sckit-learn  
https://software.intel.com/en-us/articles/installing-the-intel-distribution-for-python-and-intel-performance-libraries-with-pip-and
```

Docker Hub

```
docker pull intelpython/intelpython3_full
```

YUM/APT

Access for yum/apt:
<https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python>



2.7 & 3.6 &
3.7

OneAPI Deep Neural Network Library (OneDNN)

- Features:
 - API: DPC++, C++, and C
 - Training: float32, bfloat16
 - Inference: float32, float16, bfloat16, int8
 - MLPs, CNNs (1D, 2D and 3D), RNNs (plain, LSTM, GRU)
- Support matrix:
 - Compilers: Intel, GCC, CLANG, MSVC
 - OSes: Linux*, Windows*
 - CPU engine:
 - HW: Intel Atom®, Intel® Core™, Intel® Xeon®
 - Runtimes: DPC++, OpenMP, TBB
 - GPU engine:
 - HW: Intel® HD Graphics, Intel® Iris® Plus Graphics
 - Runtimes: DPC++, OpenCL™

Category	Functions
Compute intensive operations	<ul style="list-style-type: none">• (De-)Convolution• Inner Product• Vanilla RNN, LSTM, GRU
Memory bandwidth limited operations	<ul style="list-style-type: none">• Pooling• Batch Normalization• Local Response Normalization• Elementwise (ReLU, tanh, logistic etc.)• Softmax• Sum• Concat• Shuffle
Data manipulation	<ul style="list-style-type: none">• Reorder

OneAPI Deep Neural Network Library (OneDNN)

What's Inside

TensorFlow*
Pytorch*
Torch*
BigDL
Caffe* (...)

oneDNN

CPU

Intel
Atom

Intel
Core

Intel
Xeon

GPU

Intel
IGPU

Intel
dGPU

Deep learning and AI ecosystem includes edge and datacenter applications.

- Open source frameworks (TensorFlow*, Pytorch*, ONNX Runtime*)
- OEM applications (Matlab*, DL4J*)
- Cloud service providers internal workloads
- Intel deep learning products (OpenVINO™, BigDL)

oneDNN is an open source performance library for deep learning applications

- Includes **optimized** versions of **key deep learning functions**
- **Abstracts out** instruction set and other complexities of performance optimizations
- **Same API for both Intel CPU's and GPU's**, use the best technology for the job
- **Open** for community contributions

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

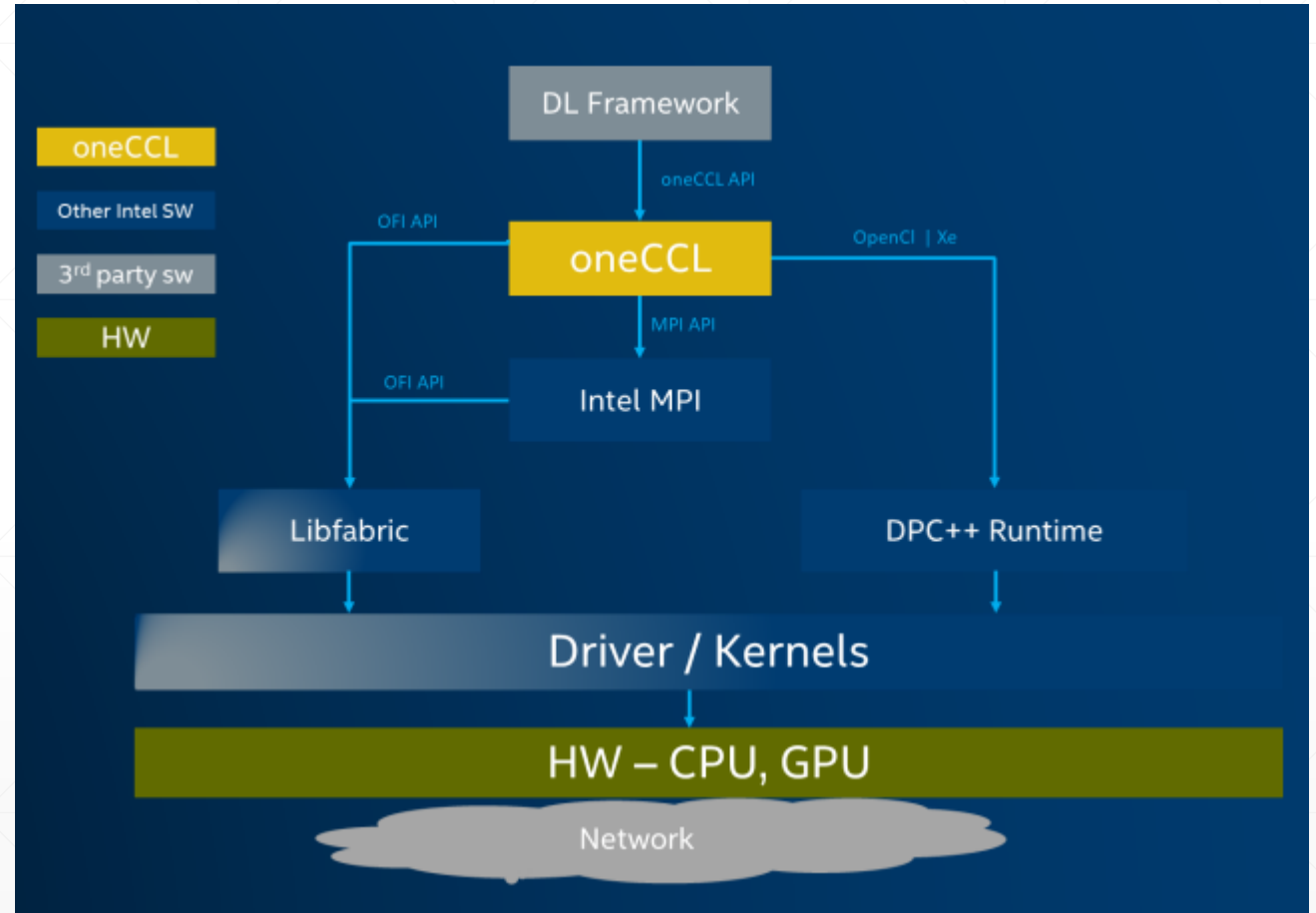
Notice Revision #20110804

Intel oneAPI Collective Communications Library (OneCCL)

- Optimized communication patterns cross nodes
 - Provides optimized communication patterns for high performance on Intel® CPUs and GPUs to distribute model training across multiple nodes
- support many interconnects
 - such as Intel® Omni-Path Architecture, InfiniBand*, and Ethernet
- On top of MPI and libfabrics
 - Built on top of lower-level communication middleware – MPI and libfabrics
- All –gather, all-reduce for Deep Learning
 - Enables efficient implementations of collectives used for **deep learning training – all-gather, all-reduce**

Intel oneAPI Collective Communications Library (OneCCL)

- Provides optimized communication patterns for high performance on Intel® CPUs and GPUs to distribute model training across multiple nodes
- Transparently supports many interconnects, such as Intel® Omni-Path Architecture, InfiniBand*, and Ethernet
- Built on top of lower-level communication middleware – MPI and libfabrics



Introduction to Intel Devcloud

- oneAPI available now on Intel® DEVCLOUD
 - A development sandbox to develop, test and run your workloads across a range of Intel CPUs, GPUs, and FPGAs using Intel's oneAPI software
 - software.intel.com/devcloud/oneapi
- A Fast Way to Start Coding
 - Try the oneAPI toolkits, compilers, performance libraries, and tools
 - Get 120 days of free access to the latest Intel® hardware and oneAPI software



The image shows the Intel DevCloud logo in the top left corner of a dark blue box. To the right of the logo is a white icon of a cloud with three small squares above it. Below the logo, there is a list of five benefits, each separated by a horizontal line. The text is white and centered within a dark red rectangular area.

- 1 Minute to Code
- No Hardware Acquisition
- No Download, Install or Configuration
- Easy Access to Samples & Tutorials
- Support for Jupyter Notebooks, Visual Studio Code

Summary

- Diverse workloads are driving the need for heterogeneous compute architectures, but each architecture has required separate programming models.
- oneAPI cross-architecture programming model provides freedom of choice. Apply your skills to the next innovation, not to rewriting software for the next hardware platform.
- Intel® oneAPI products take full advantage of accelerated compute by maximizing performance across Intel CPUs, GPUs, and FPGAs.
- Develop confidently with a proven set of cross-architecture libraries and advanced tools that interoperate with existing performance programming models.

THANK YOU