Multi-GPU Programming with oneAPI

oneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

2022. 12. 16.

MOASYS

oneAPI Smart Development Series (2021)

1. Introduction to Intel oneAPI for HPC and AI-DL
   - https://www.allshowtv.com/detail.html?idx=474
2. Benchmarking the Performance of oneAPI on Heterogeneous Computing Platforms
   - https://www.allshowtv.com/detail.html?idx=660
3. Optimization and GPU Offloading Workflow with Intel oneAPI
   - https://www.allshowtv.com/detail.html?idx=826
4. Leveraging Intel® oneDNN for AI Workload
   - https://www.allshowtv.com/detail.html?idx=909

oneAPI Smart Development Series (2022)

1. FPGA Development Flow with Intel® oneAPI Base Toolkit
   - https://www.allshowtv.com/detail.html?idx=995
2. OpenMP Offload with Intel® oneAPI HPC Toolkit
   - https://www.allshowtv.com/detail.html?idx=1041
3. DPC++ Essentials: Advanced Concepts
   - https://www.allshowtv.com/detail.html?idx=1112
4. Multi-GPU Programming with oneAPI
   - https://www.allshowtv.com/detail.html?idx=1188

Contents

Overview of oneAPI for Heterogenous Computing

HPC wire 2021 Readers' Choice Awards

Best HPC Programming Tool or Technology

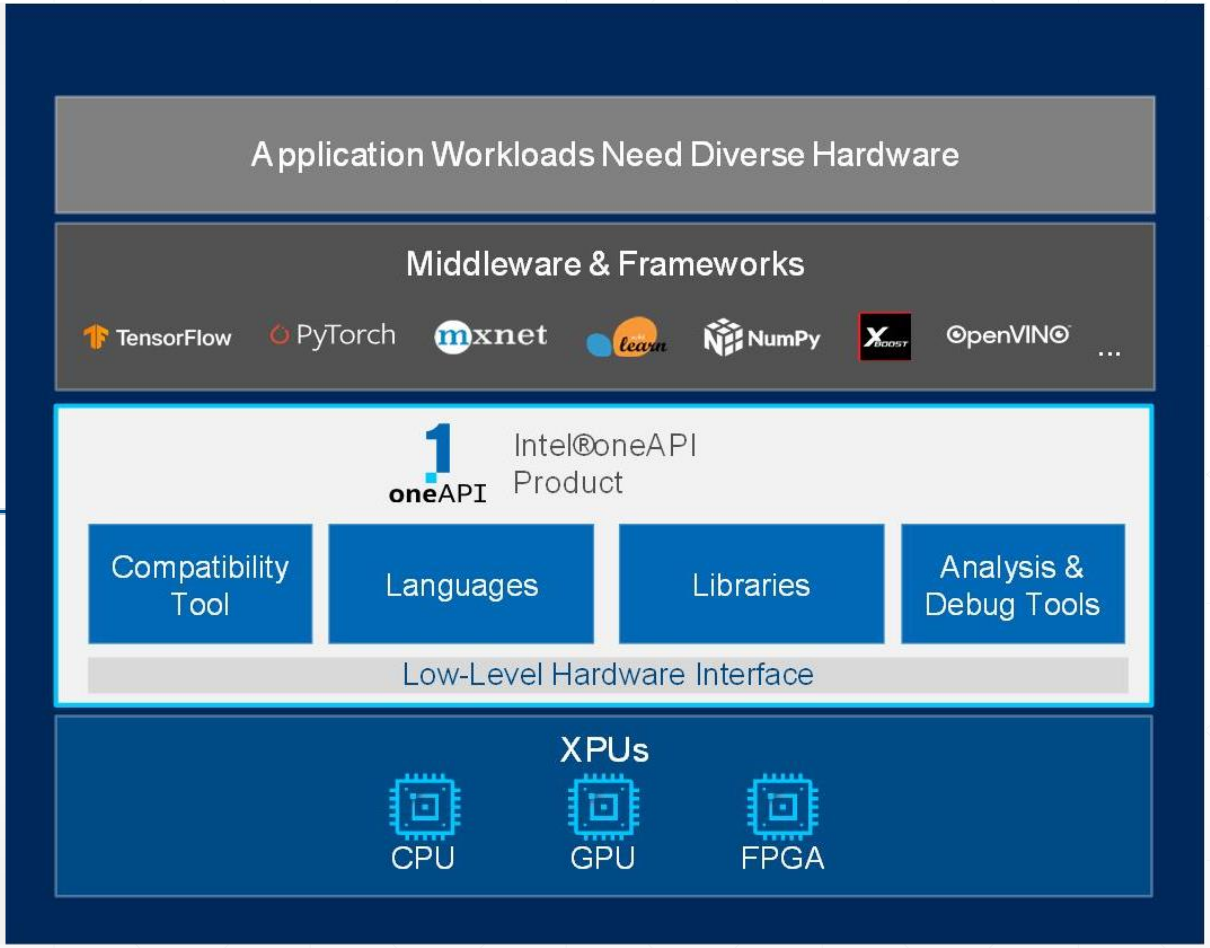
oneAPI

Open, Standards-Based Unified Software Stack

Freedom from proprietary programming models

Full performance from the hardware

Piece of mind for developers



- Support diverse accelerator devices (XPU) such as CPU, GPU and FPGA
- Provide optimized libraries for high performance computing and machine learning applications

Comparison of Heterogenous Programming Models

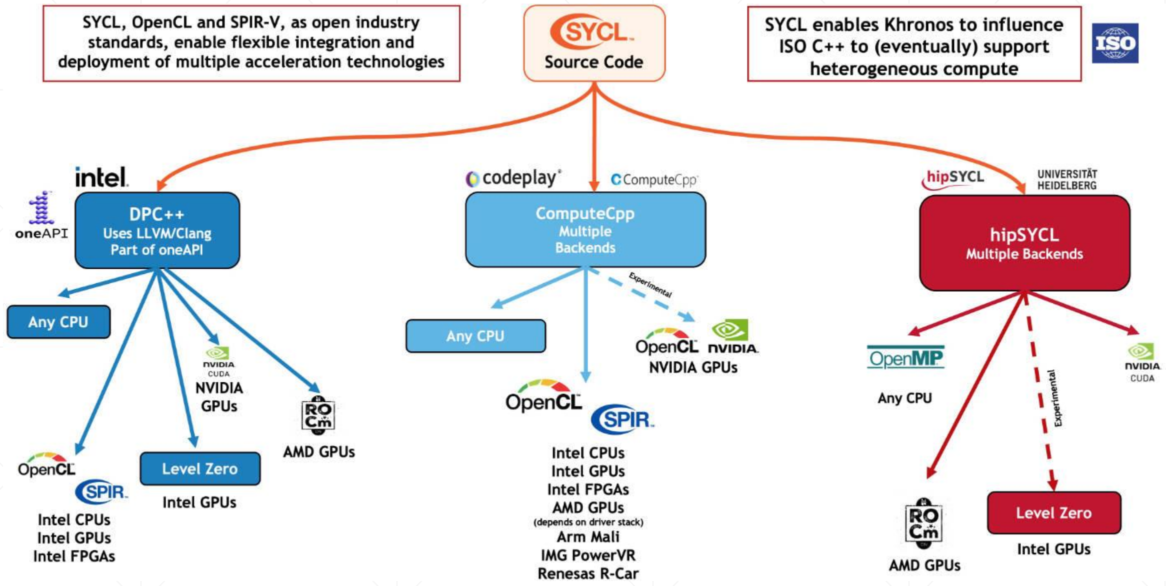
	CUDA	HIP	OpenACC	OpenMP	SYCL/DPC++
Languages	**C/C++/Fortran**	C/C++/Fortran	C/C++/Fortran	C/C++/Fortran	**C++**
Abstraction	**Low**	Low	High	High	**Medium**
Coding	**-**	-	Directive-based	Directive-based	**C++ lambda**
Parallelism	**SIMT**	SIMT	Fork-join SIMD	Fork-join SIMD	**OpenCL**
Offload	**GPU (NVIDIA)**	GPU (NVIDIA/AMD)	GPU (NVIDIA)	CPU/GPU (NVIDIA/AMD/Intel)	**CPU/GPU/FPGA (NVIDIA/AMD/Intel)**
Compiler	**Proprietary**	LLVM	PGI/CCE/GCC	PGI/CCE/GCC/LLVM/XL/Intel	**LLVM**
License	**Proprietary**	Open-source	Open-source	Open-source	**Open-source**

- Write once, run everywhere with SYCL/DPC++
  - ISO C++17 and SYCL standards and extensions
  - Single-source style framework
  - Asynchronous execution model
  - Open, cross-architecture and cross-vendor

oneAPI for Data Center and AI

Intel data center GPU codenamed
Ponte Vecchio
>100B Transistors
128GB HBM2e
Aurora >2 Exaflops of peak compute

International Supercomputing Conference (2022)



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

- Intel DPC++ Compiler: oneAPI Base Toolkit
  - OpenCL backend: optimized for Intel CPUs, GPUs (Gen9, 11, Xe) and FPGA (Stratix, Aria)
  - Level Zero backend: low-level offloading API for Intel GPUs
- Intel LLVM Compiler: open source project
  - CUDA backend: experimental support for NVIDIA GPUs
  - HIP backend: experimental support for AMD GPUs via ROCm 4.x

Basic Structure of DPC++ Program

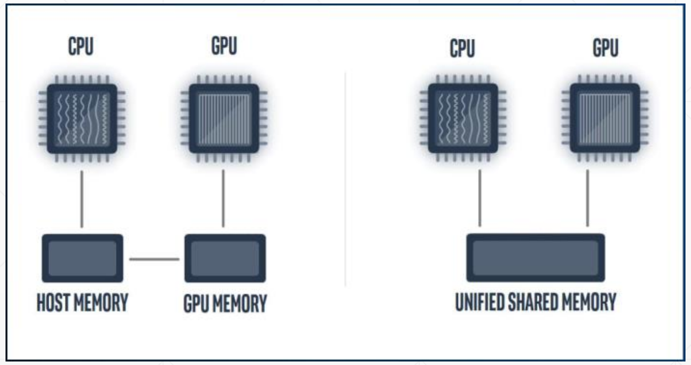
```
#include <iostream>
#include <CL/sycl.hpp>
#define N 1000

using namespace sycl;

int main() {
    queue q{default_selector()};

    double* vec = new double[N];

    // initialization on host
    for (auto i = 0; i < N; i++) vec[i] = 0.0;

    { // scope opens
        buffer<double, 1> vec_buf(vec, size<1> N);

        q.submit([&](handler& h) {
            accessor vec_acc(vec_buf, h);

            h.parallel_for(N, [=](auto i) {
                vec_acc[i] = 2*i;
            });
        });
    } // scope closes, buffer destruction

    delete vec;
}
```

- sycl::queue:
  - Offload code submitted to device via queue
  - One queue maps to exactly one device to avoid runtime ambiguity
  - Multiple queues can use same device
- sycl::buffer:
  - Initialized from already allocated memory
  - Buffer destruction via scope closing is a blocking call
- sycl::accessor:
  - How memory is accessed: *read_only, write_only, read_write*
  - Where memory is accessed: *global_memory, local_memory*
- sycl::handler:
  - Constructed at runtime, as argument to lambda function of *submit*
  - How code is submitted to queue: *single_task (serial), paralle_for (SIMT)*
  - Kernel code as callable lambda functions
  - No dynamic memory allocation

Basic Structure of DPC++ Program: SYCL 2020

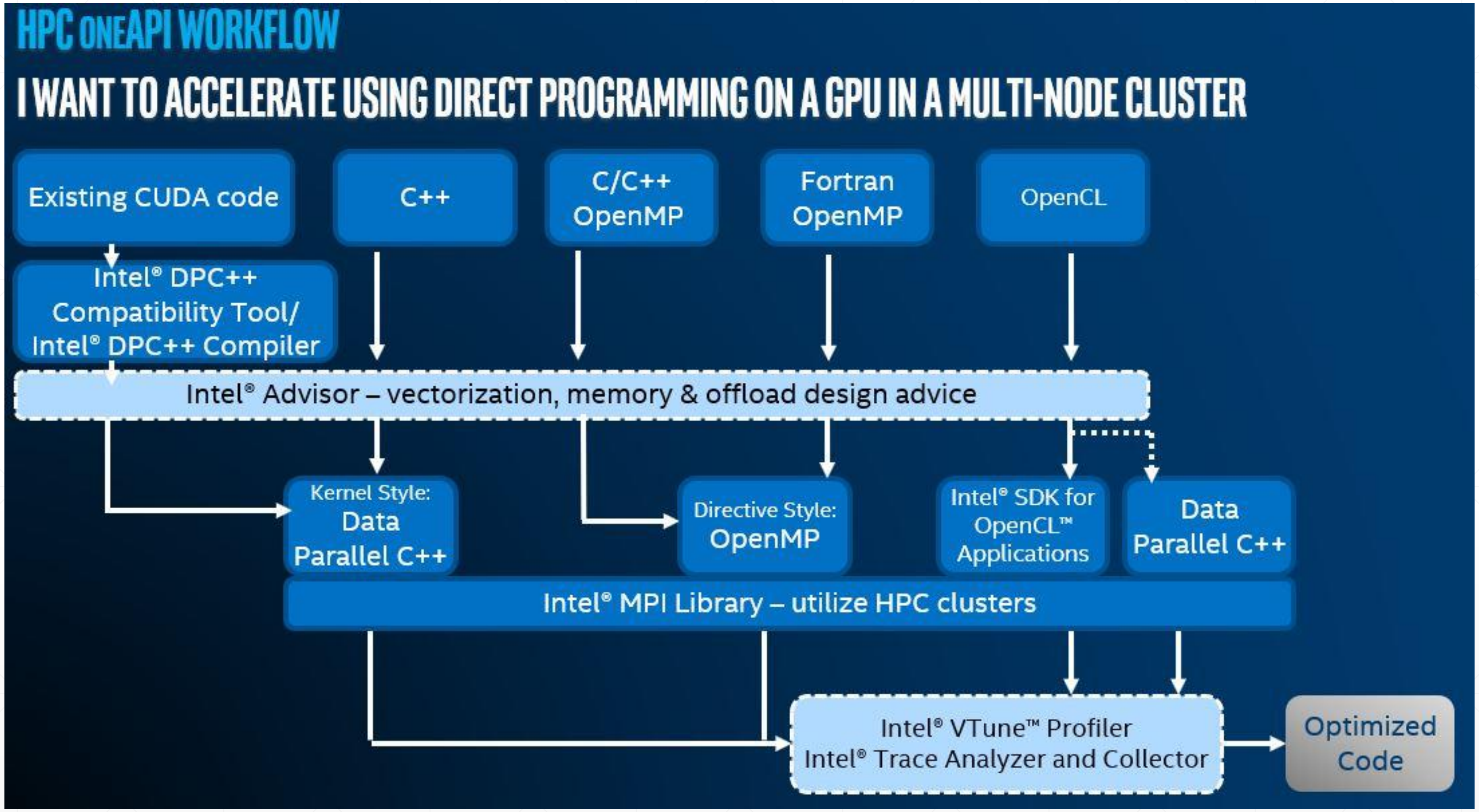
```
#include <iostream>
#include <CL/sycl.hpp>
#define N 1000

using namespace sycl;

int main() {
    queue q{default_selector()};

    auto vec = malloc_shared<double>(N, q);

    // initialization on host
    for (auto i = 0; i < N; i++) vec[i] = 0.0;

    q.parallel_for(N, [=](auto i) {
        vec[i] = 2*i;
    }).wait();

    free vec;
}
```

- A pointer-based approach to memory allocation
  - Simplify porting to accelerators with minimal changes
- sycl::malloc::host()
  - Return a pointer to memory physically located on the host
  - Device can access host allocated memory via PCIe buses
- sycl::malloc::device()
  - Return a pointer to memory physically located on the device
- sycl::malloc::shared()
  - Return a pointer to the *unified virtual address* (UVA) space
  - SYCL runtimes automatically handle data movements between host/device



Allocation	Host accessible ?	Device accessible ?	Memory Space
malloc_host	Yes	Yes	Host
malloc_device	No	Yes	Device
malloc_shared	Yes	Yes	UVA

oneAPI HPC Workflow



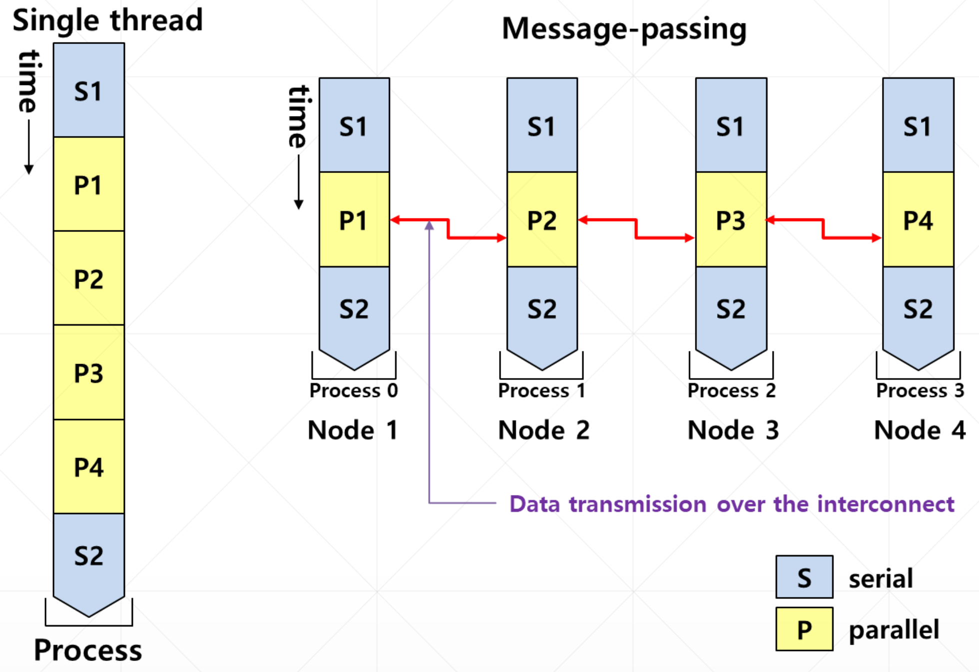
HPC ONEAPI WORKFLOW
I WANT TO ACCELERATE USING DIRECT PROGRAMMING ON A GPU IN A MULTI-NODE CLUSTER
Existing CUDA code
C++
C/C++ OpenMP
Fortran OpenMP
OpenCL
Intel® DPC++ Compatibility Tool/ Intel® DPC++ Compiler
Intel® Advisor – vectorization, memory & offload design advice
Kernel Style: Data Parallel C++
Directive Style: OpenMP
Intel® SDK for OpenCL™ Applications
Data Parallel C++
Intel® MPI Library – utilize HPC clusters
Intel® VTune™ Profiler Intel® Trace Analyzer and Collector
Optimized Code

- DPC++ compatibility tool is now open source: https://github.com/oneapi-src/SYCLomatic

Basic Structure of MPI Program: Hello, World!

```
#include <cstdio>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int i, myid, ntasks, namelen;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ntasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    if (myid == 0) {
        printf("In total there are %i tasks\n", ntasks);
    }

    printf("Hello from rank %i\n", myid);

    MPI_Finalize();
}
```

```
$ mpiicpc hello.ccp -o hello.x
$ mpirun -np 8 ./hello.x
Hello from rank 5
Hello from rank 2
Hello from rank 6
Hello from rank 1
In total there are 8 tasks
Hello from rank 0
Hello from rank 4
Hello from rank 3
Hello from rank 7
```



- What is MPI ?
  - Message Passing Interface
  - A library, not a language
  - For inter-process communication and data exchange
  - Use for distributed memory computing

Basic Structure of MPI Program: Initialization

```
#include <cstdio>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int i, myid, ntasks, namelen;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    if (iproc == 0) {
        printf("In total there are %i tasks\n", ntasks);
    }

    printf("Hello from rank %i\n", iproc);

    MPI_Finalize();
}
```

```
$ mpiicpc hello.ccp -o hello.x
$ mpirun -np 8 ./hello.x
Hello from rank 5
Hello from rank 2
Hello from rank 6
Hello from rank 1
In total there are 8 tasks
Hello from rank 0
Hello from rank 4
Hello from rank 3
Hello from rank 7
```

int MPI_Init(int *argc, char ***argv)

- ❑ Initialize the MPI execution environment
  - o argc: pointer to the number of arguments (IN)
  - o argv: pointer to the argument vector (IN)

int MPI_Comm_size(MPI_Comm comm, int *size)

- ❑ Determine the size of communicator
  - o comm: communicator handle (IN)
  - o size: number of process in the comm (OUT)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

- ❑ Determine the rank of calling process in the communicator
  - o comm: communicator handle (IN)
  - o rank: number of process in the comm (OUT)

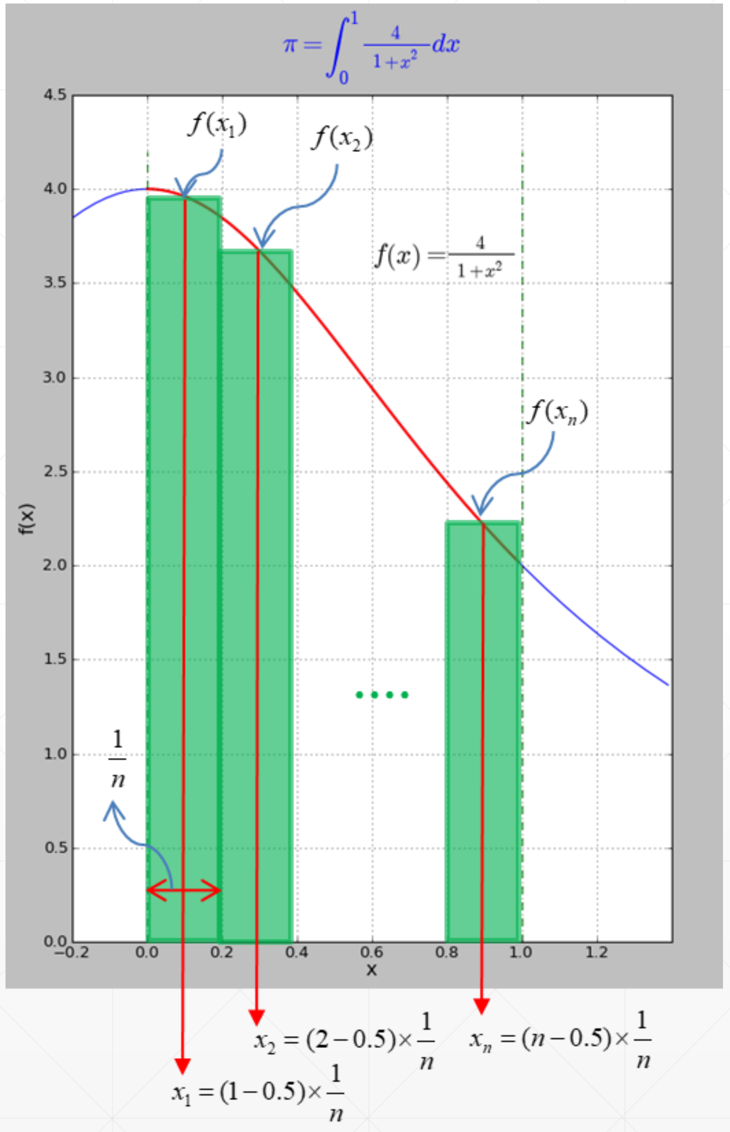
int MPI_Comm_rank(void)

- ❑ Terminates MPI execution environment

Estimation of Pi via Numerical Integration

```
#include <cstdio>
#include <cmath>

constexpr int n = 1000;

double integrate(int, int);

int main(int argc, char** argv)
{
    double pi = 0.0;
    double h  = 1.0 / n;

    for (int i = 1; i < n; i += 1) {
        double x = h * (i - 0.5);

        pi += 4.0 * h / (1.0 + x * x);
    }

    printf("Approximate pi=%.6f with error=%.6e\n", pi, fabs(pi-M_PI));

    return 0;
}
```

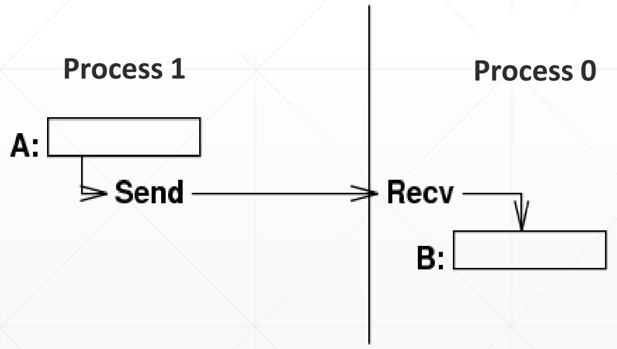
```
$ icpx pi_serial.cpp -o pi.x
$ ./pi.x
Approximate pi=3.141393 with error=2.000092e-04
```



Estimation of Pi: MPI P2P Communication

```
#include <cstdio>
#include <cmath>
#include <mpi.h>

constexpr int n = 10000;

double integrate(int, int);

int main(int argc, char** argv)
{
    int iproc, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    double ipi = integrate(iproc, nprocs);

    if (iproc == 0) {
       double pi = ipi;
       for (int i = 1; i < nprocs; i++) {
            MPI_Recv(&ipi, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            pi += ipi;
        }
        printf("Approximate pi=%.6f with error=%.6e\n", pi, fabs(pi-M_PI));
    } else {
        MPI_Send(&ipi, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }

    MPI_Finalize();
}
```

```
double integrate(int iproc, int nprocs) {
    double ipi = 0.0;
    double h   = 1.0 / n;

    for (int i=iproc+1; i <= n; i += nprocs) {
        double x = h * (i - 0.5);

        ipi += 4.0 * h / (1.0 + x*x);
    }

    return ipi;
}
```



Where to send ?
What to send ?
How many to send ?

Where to receive ?
What to receive ?
How many to receive ?

MPI_Send/MPI_Recv API

*int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)*

- ❑ Perform blocking send
  - buf: initial address of send buffer (IN)
  - count: number of elements in send buffer (IN)
  - datatype: datatype of each send buffer element (IN)
  - dest: rank of destination (IN)
  - tag: message tag (IN)
  - comm: communicator (IN)

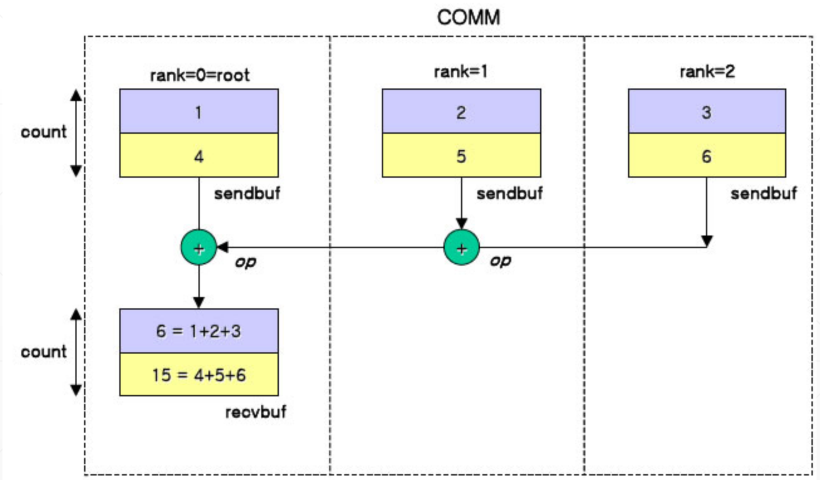
*int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)*

- ❑ Perform blocking receive
  - buf: initial address of send buffer (OUT)
  - count: maximum number of elements in receive buffer (IN)
  - datatype: datatype of each receive buffer element (IN)
  - source: rank of source (IN)
  - tag: message tag (IN)
  - comm: communicator (IN)
  - status: status object (OUT)

Estimation of Pi: MPI Collective Communication

```
#include <cstdio>
#include <cmath>
#include <mpi.h>

constexpr int n = 10000;

double integrate(int, int);

int main(int argc, char** argv)
{
    int iproc, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    double pi  = 0.0;
    double ipi = integrate(iproc, nprocs);

    MPI_Reduce(&ipi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (iproc == 0) {
        printf("Approximate pi=%.6f with error=%.6e\n", pi, fabs(pi-M_PI));
    }

    MPI_Finalize();
}
```

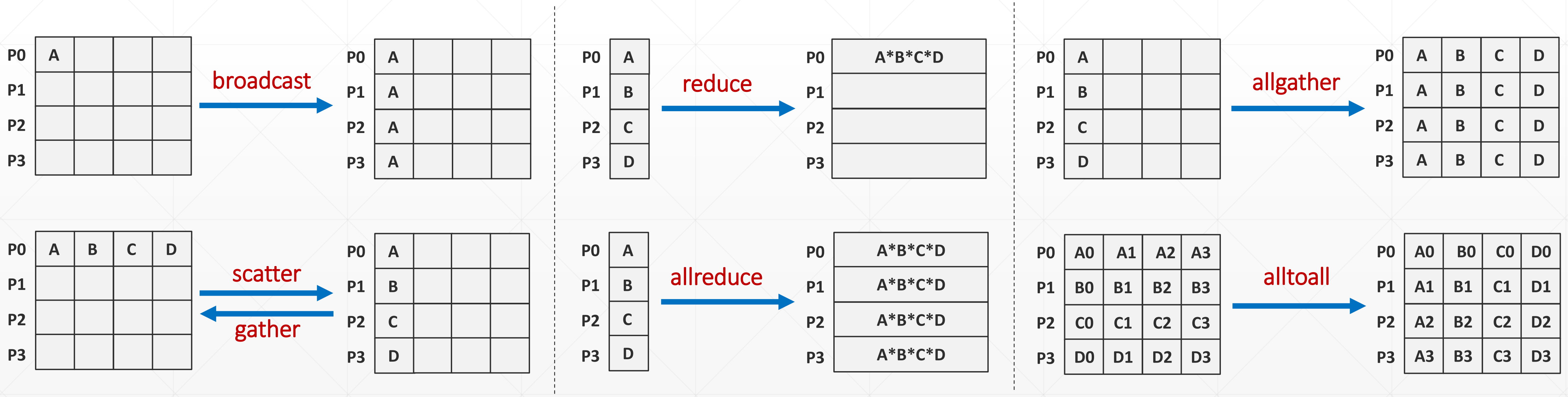
```
double integrate(int iproc, int nprocs) {
    double ipi = 0.0;
    double h   = 1.0 / n;

    for (int i=iproc+1; i <= n; i += nprocs) {
        double x = h * (i - 0.5);

        ipi += 4.0 * h / (1.0 + x*x);
    }

    return ipi;
}
```



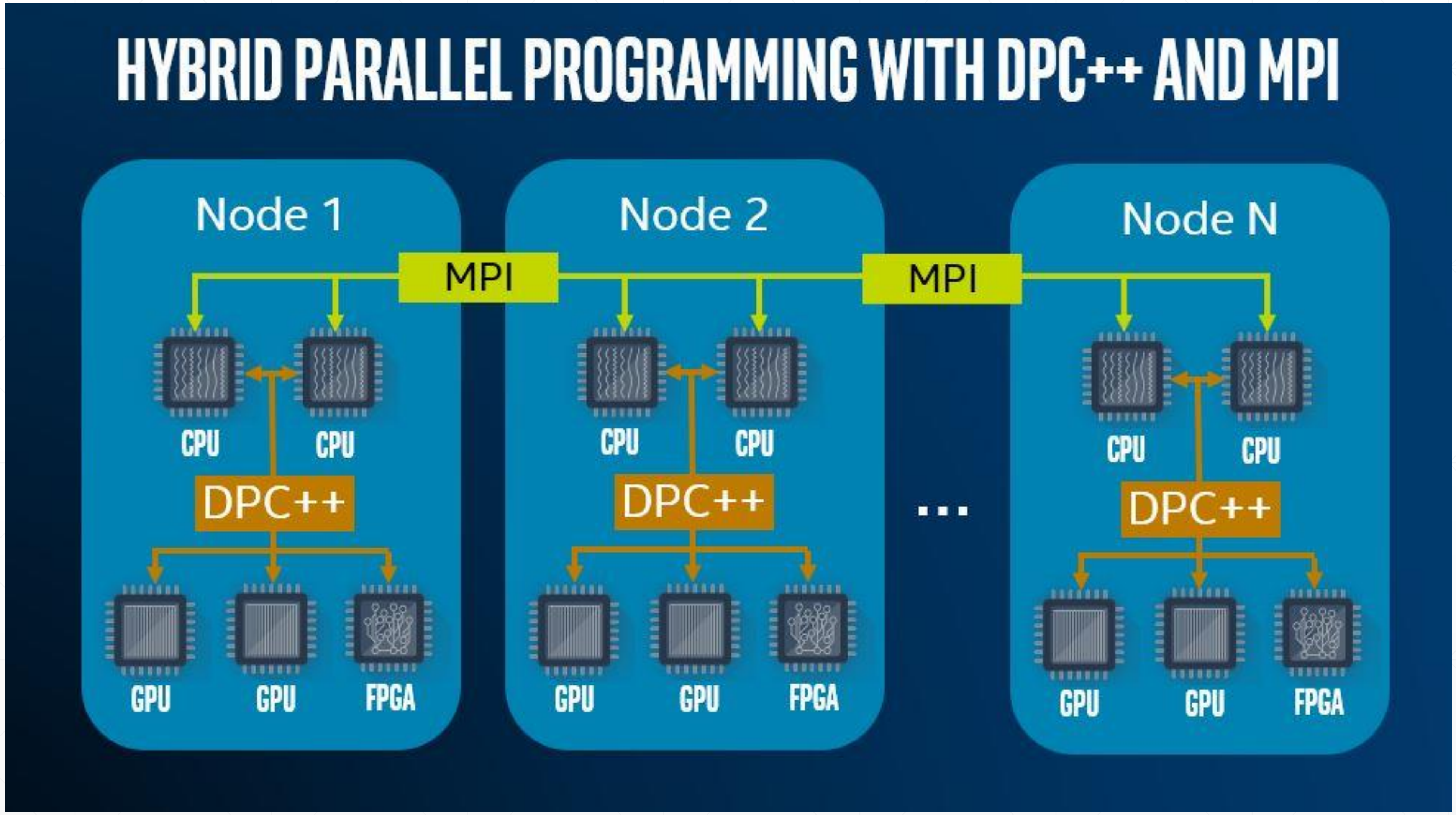
MPI_Reduce API

int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- Reduces values on all processes to a single value
  - sendbuf: address of send buffer (IN)
  - recvbuf: address of receive buffer (OUT)
  - count: number of elements in send buffer (IN)
  - datatype: data type of elements of send buffer (IN)
  - op: reduce operation (IN)
  - root: rank of root process (IN)
  - comm: communicator (IN)



MPI and DPC++ Interoperatibility



Interoperability between MPI and oneAPI

```
#include <cstdio>
#include <cmath>
#include <mpi.h>
#include <CL/sycl.hpp>

using namespace sycl;
constexpr int n = 1000;

void integrate(double*, int, int, queue&);

int main(int argc, char** argv) {
    int iproc, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    queue q{default_selector()};

    double pi      = 0.0;
    int n_per_rank = n / nprocs;

    double* ipi = new double[n_per_rank];
    for (int i = 0; i < n_per_rank; i++) ipi[i] = 0.0;

    integrate(ipi, iproc, nprocs, q);

    MPI_Reduce(&ipi[0], &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

Include MPI header file

Include SYCL header file

Initialize MPI communicator

Create a SYCL queue

Each *iproc* allocates its own *ipi[]* array

Each *iprocs* calculate *ipi[]* on SYCL devices

Root process gather values Pi via MPI_SUM operator

MPI termination

Estimation of PI on Device (Buffer)

```
void integrate(double* ipi, int iproc, int nprocs, queue& q) {
    double dx  = 1.0 / n;

    buffer<double,1> pi_buf(ipi, range<1>(n/nprocs));

    q.submit([&](handler& h) {
        accessor pi_acc(pi_buf, h, write_only);

        h.parallel_for(n / nprocs,[=](id<1> i) {
            double x  = (double) iproc / (double) nprocs + ((double) i - 0.5) * dx;
            pi_acc[i] = 4.0 * dx / (1.0 + x*x);
        });
    }).wait();

    q.submit([&](handler& h) {
        accessor pi_acc(pi_buf, h);

        h.single_task([=]() {
            for (int i = 1; i < n/nprocs; i++) pi_acc[0] += pi_acc[i];
        });
    }).wait();
}
```

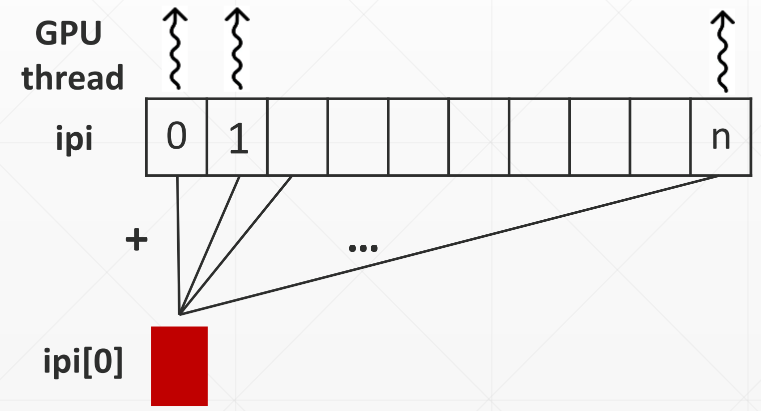
Create buffer to ipi

Create accessor to ipi's buffer

Calculate value of pi each data point

Create accessor to ipi's buffer

Accumulate pi to first element of pi_per_rank



- https://www.allshowtv.com/detail.html?idx=1112
  - Workgroup reduction
  - Subgroup reduction
  - Simplified reduction

Estimation of PI on Device (USM)

```
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    queue q{default_selector()};

    double pi     = 0.0;
    int n_per_rank = n / nprocs;

    auto ipi = malloc_shared<double>(n_per_rank, q);
    for (int i = 0; i < n_per_rank; i++) ipi[i] = 0.0;

    integrate(ipi, iproc, nprocs, q);

    MPI_Reduce(&ipi[0], &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

Initialize MPI communicator

Create a SYCL queue

Allocate shared memory between host and device

Each *iprocs* calculate *ipi* on SYCL devices

Root process gather values Pi via MPI_SUM operator

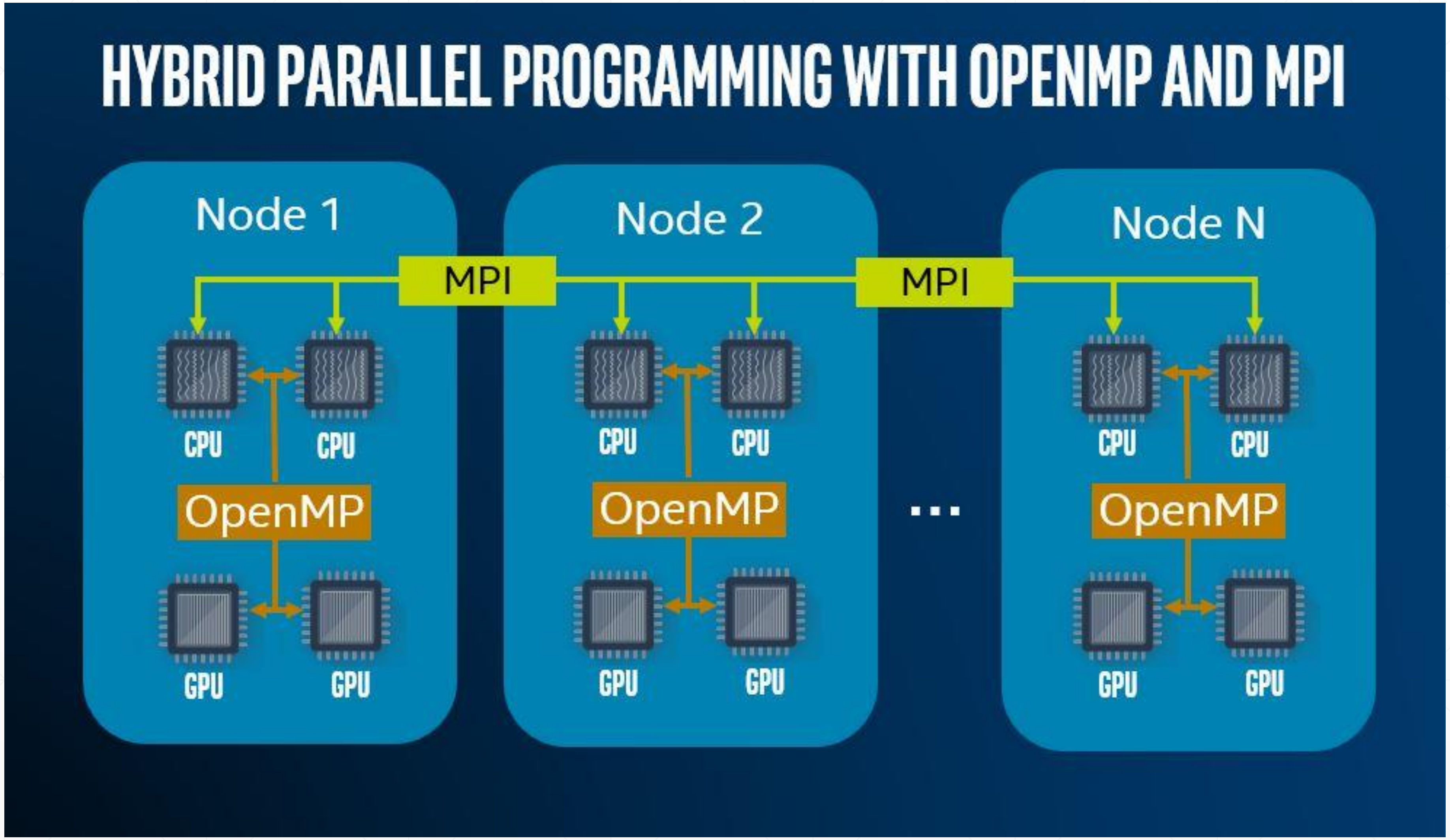
MPI termination

```
void integrate(double* ipi, int iproc, int nprocs, queue& q) {
    double dx  = 1.0 / n;

    q.parallel_for(n / nprocs,[=](id<1> i) {
        double x  = (double) iproc / (double) nprocs + ((double) i - 0.5) * dx;
        ipi[i] = 4.0 * dx / (1.0 + x*x);
    }).wait();

    q.single_task([=]() {
        for (int i = 1; i < n/nprocs; i++) ipi[0] += ipi[i];
    }).wait();
}
```

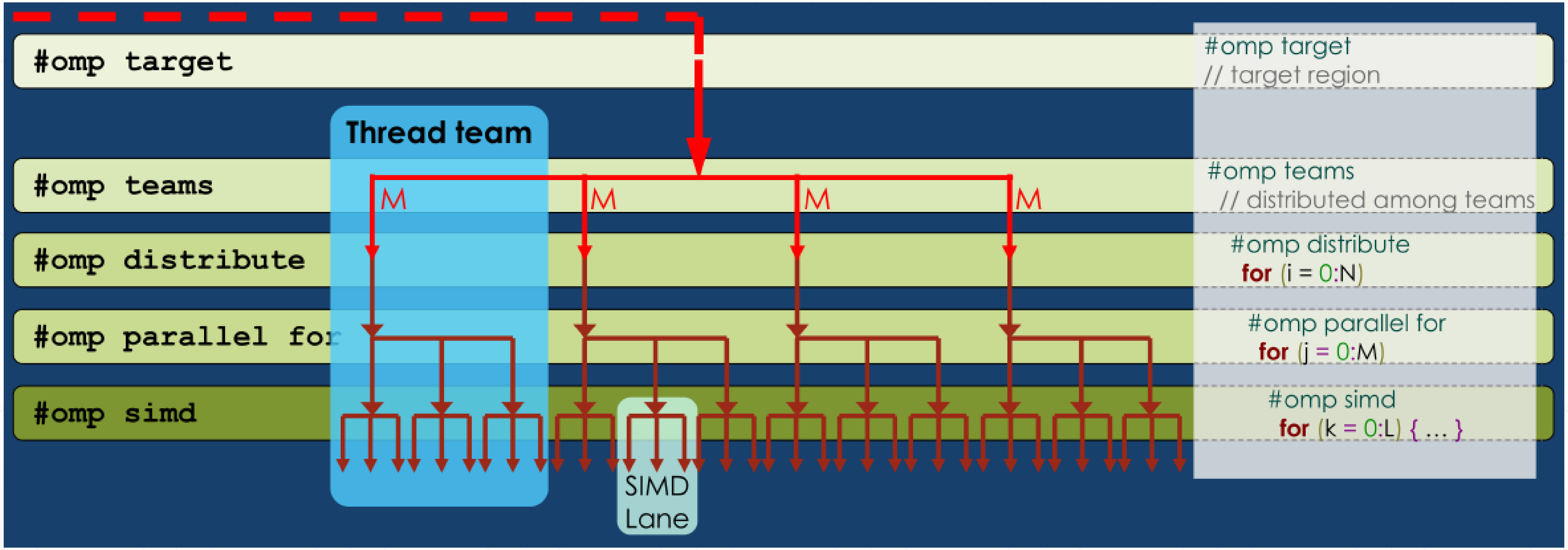
Calculate value of pi each data point

Accumulate pi to first element of ipi

MPI and OpenMP Interoperatibility



OpenMP Hierarchical Parallelism



- target: create a single device thread
- teams: create a group of master threads, mapped to iGPU subslice
- distribute: assign chunks of loop iterations to teams
- parallel for simd: run each chunk of iterations on EU threads and SIMD lanes

Estimation of PI via OpenMP Offload

```
#include <cstdio>
#include <cmath>
#include <mpi.h>

constexpr int n = 1000;

void integrate(double*, int, int);

int main(int argc, char** argv) {
    int iproc, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    int n_per_rank = n / nprocs;

    double* ipi = new double[n_per_rank];
    for (int i = 0; i < n_per_rank; i++) ipi[i] = 0.0;

    integrate(ipi, iproc, nprocs);

    double  pi = 0.0;
    double ipi = 0.0;

    for (auto i = 0; i < n_per_rank; i++) ipi += pi_per_rank[i];

    MPI_Reduce(&ipi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

Include MPI header file

Initialize MPI communicator

Each *iproc* allocates its own *pi_per_rank[]* array

Each *iprocs* calculate *pi_per_rank* using OpenMP

Root process gather values Pi via MPI_SUM operator

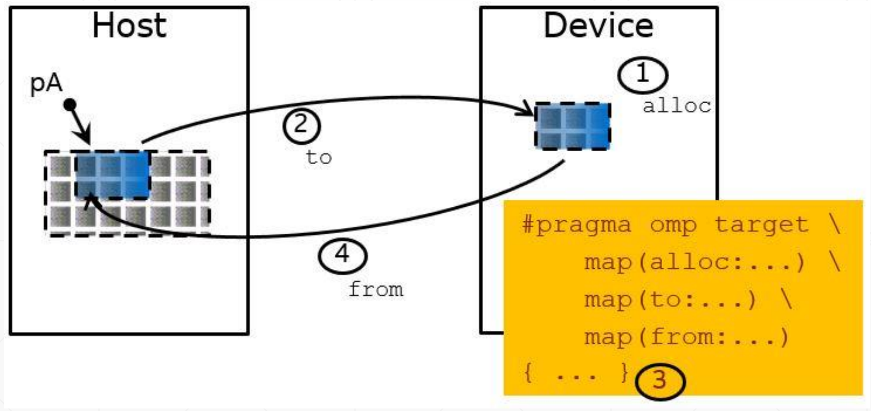
Root process gather values Pi via MPI_SUM operator

MPI termination

Calculation of PI

```
void integrate(double* pi, int iproc, int nprocs) {
    double dx     = 1.0 / n;
    int num_step = n / nprocs;

    #pragma omp target map(from:pi[0:num_step])
    {
        #pragma omp teams distribute parallel for simd
        for (int i=0; i< num_step; i++) {
            double x = (double) iproc / (double) nprocs + ((double) i - 0.5) * dx;
            pi[i] = 4.0 * dx / (1.0 + x*x);
        }
    }
}
```

- map: explicit control movement of data
- Available map type:
  - *alloc*: allocated but not initialized
  - *to*: host → device copy on target entry
  - *from*: device → host copy on target exit
  - *tofrom*: both *to* and *from* (default)
- Pointers (C/C++) and dynamically allocated arrays (Fortran): dimensions are required

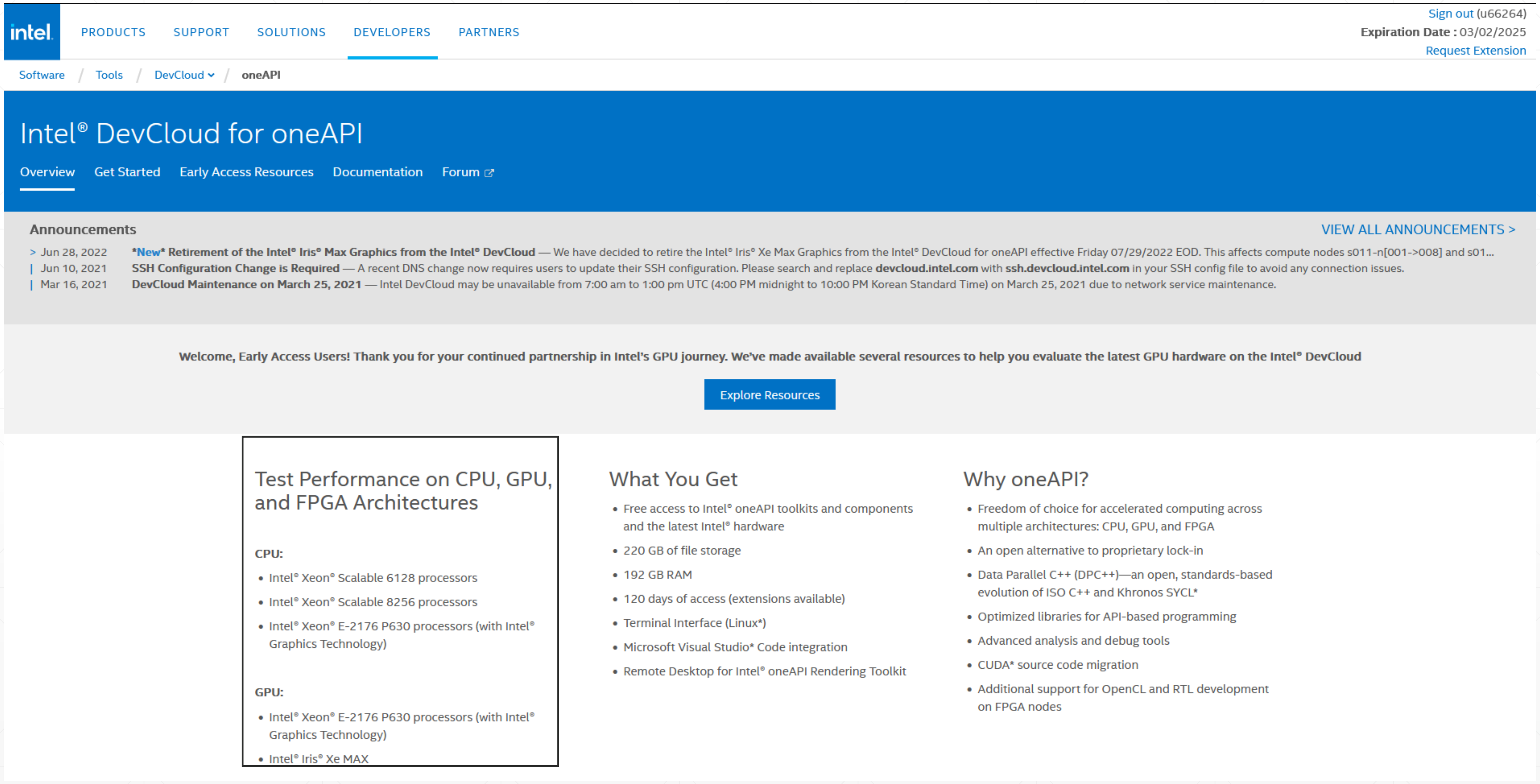


```
#pragma omp target \
    map(alloc:...) \
    map(to:...) \
    map(from:...)
{ ... }
```

Conclusions

- Basic MPI concepts:
  - Communicator functions: MPI_Init(), MPI_Comm_size(), MPI_Comm_rank()
  - P2P functions: MPI_Send(), MPI_Recv()
  - Collective functions: MPI_Reduce()
- Interoperability between MPI and DPC++
  - Buffer method
  - USM method
- Interoperability between MPI and OpenMP offload
- Contact: sales@moasys.com
  - GPU and FPGA code migration
  - Code optimization and parallelization consultant
  - Specialized HPC education

https://devcloud.intel.com/oneapi/



PRODUCTS SUPPORT SOLUTIONS DEVELOPERS PARTNERS

Sign out (u66264)
Expiration Date : 03/02/2025
Request Extension

Software / Tools / DevCloud / **oneAPI**

Intel® DevCloud for oneAPI

Overview Get Started Early Access Resources Documentation Forum

Announcements VIEW ALL ANNOUNCEMENTS >

> Jun 28, 2022 ***New* Retirement of the Intel® Iris® Max Graphics from the Intel® DevCloud** — We have decided to retire the Intel® Iris® Xe Max Graphics from the Intel® DevCloud for oneAPI effective Friday 07/29/2022 EOD. This affects compute nodes s011-n[001->008] and s01...
| Jun 10, 2021 **SSH Configuration Change is Required** — A recent DNS change now requires users to update their SSH configuration. Please search and replace **devcloud.intel.com** with **ssh.devcloud.intel.com** in your SSH config file to avoid any connection issues.
| Mar 16, 2021 **DevCloud Maintenance on March 25, 2021** — Intel DevCloud may be unavailable from 7:00 am to 1:00 pm UTC (4:00 PM midnight to 10:00 PM Korean Standard Time) on March 25, 2021 due to network service maintenance.

Welcome, Early Access Users! Thank you for your continued partnership in Intel's GPU journey. We've made available several resources to help you evaluate the latest GPU hardware on the Intel® DevCloud

Explore Resources

Test Performance on CPU, GPU, and FPGA Architectures

CPU:

- Intel® Xeon® Scalable 6128 processors
- Intel® Xeon® Scalable 8256 processors
- Intel® Xeon® E-2176 P630 processors (with Intel® Graphics Technology)

GPU:

- Intel® Xeon® E-2176 P630 processors (with Intel® Graphics Technology)
- Intel® Iris® Xe MAX

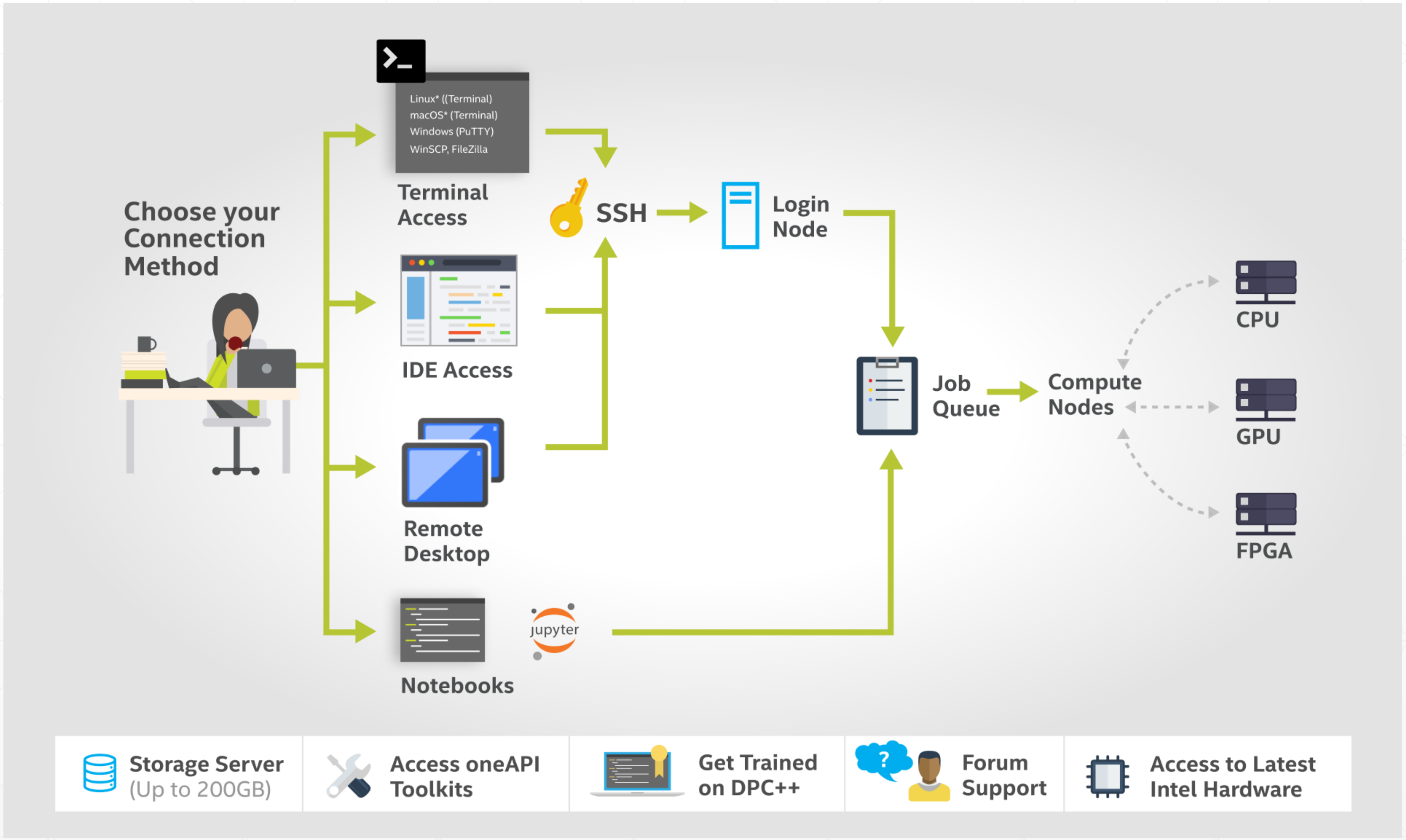
What You Get

- Free access to Intel® oneAPI toolkits and components and the latest Intel® hardware
- 220 GB of file storage
- 192 GB RAM
- 120 days of access (extensions available)
- Terminal Interface (Linux*)
- Microsoft Visual Studio* Code integration
- Remote Desktop for Intel® oneAPI Rendering Toolkit

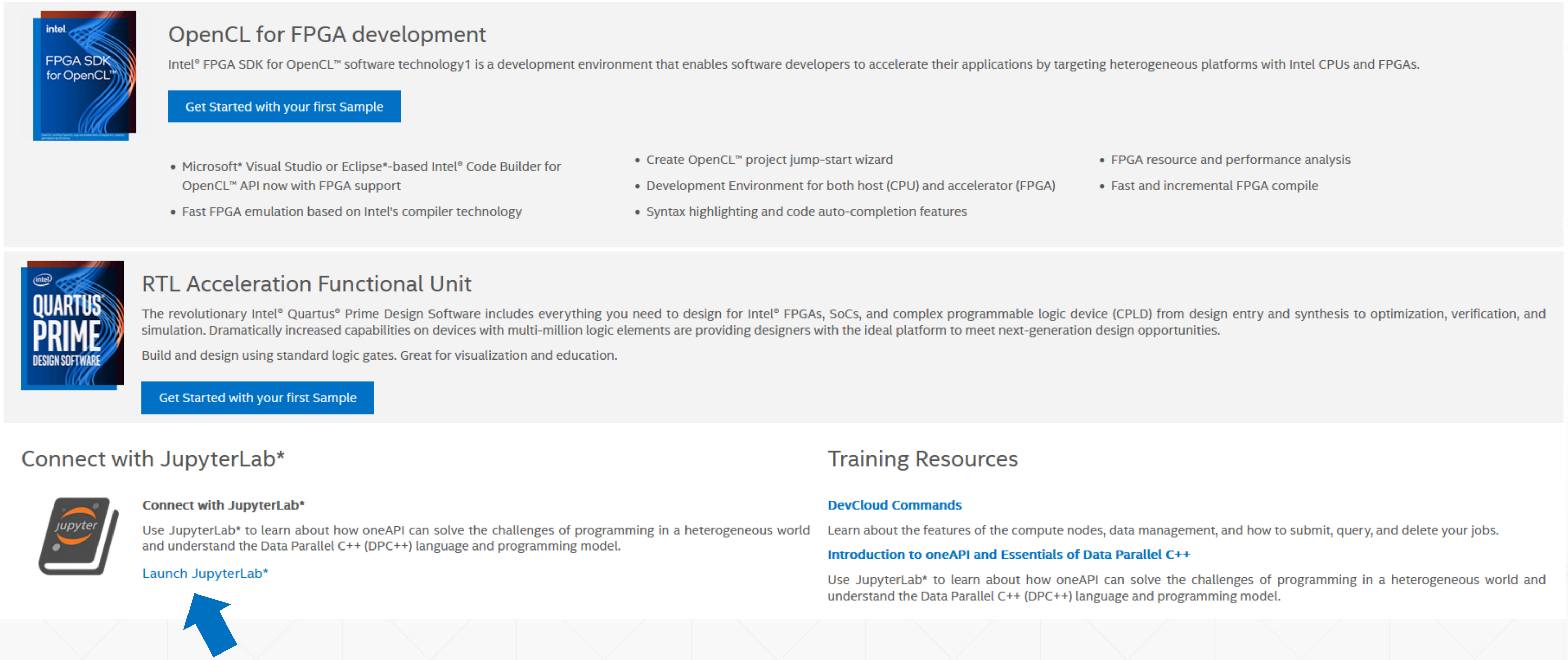
Why oneAPI?

- Freedom of choice for accelerated computing across multiple architectures: CPU, GPU, and FPGA
- An open alternative to proprietary lock-in
- Data Parallel C++ (DPC++)—an open, standards-based evolution of ISO C++ and Khronos SYCL*
- Optimized libraries for API-based programming
- Advanced analysis and debug tools
- CUDA* source code migration
- Additional support for OpenCL and RTL development on FPGA nodes

Connection Methods



https://devcloud.intel.com/oneapi/get_started/



OpenCL for FPGA development

Intel® FPGA SDK for OpenCL™ software technology1 is a development environment that enables software developers to accelerate their applications by targeting heterogeneous platforms with Intel CPUs and FPGAs.

Get Started with your first Sample

- Microsoft* Visual Studio or Eclipse*-based Intel® Code Builder for OpenCL™ API now with FPGA support
- Fast FPGA emulation based on Intel's compiler technology
- Create OpenCL™ project jump-start wizard
- Development Environment for both host (CPU) and accelerator (FPGA)
- Syntax highlighting and code auto-completion features
- FPGA resource and performance analysis
- Fast and incremental FPGA compile

RTL Acceleration Functional Unit

The revolutionary Intel® Quartus® Prime Design Software includes everything you need to design for Intel® FPGAs, SoCs, and complex programmable logic device (CPLD) from design entry and synthesis to optimization, verification, and simulation. Dramatically increased capabilities on devices with multi-million logic elements are providing designers with the ideal platform to meet next-generation design opportunities.

Build and design using standard logic gates. Great for visualization and education.

Get Started with your first Sample

Connect with JupyterLab*

Connect with JupyterLab*

Use JupyterLab* to learn about how oneAPI can solve the challenges of programming in a heterogeneous world and understand the Data Parallel C++ (DPC++) language and programming model.

Launch JupyterLab*

Training Resources

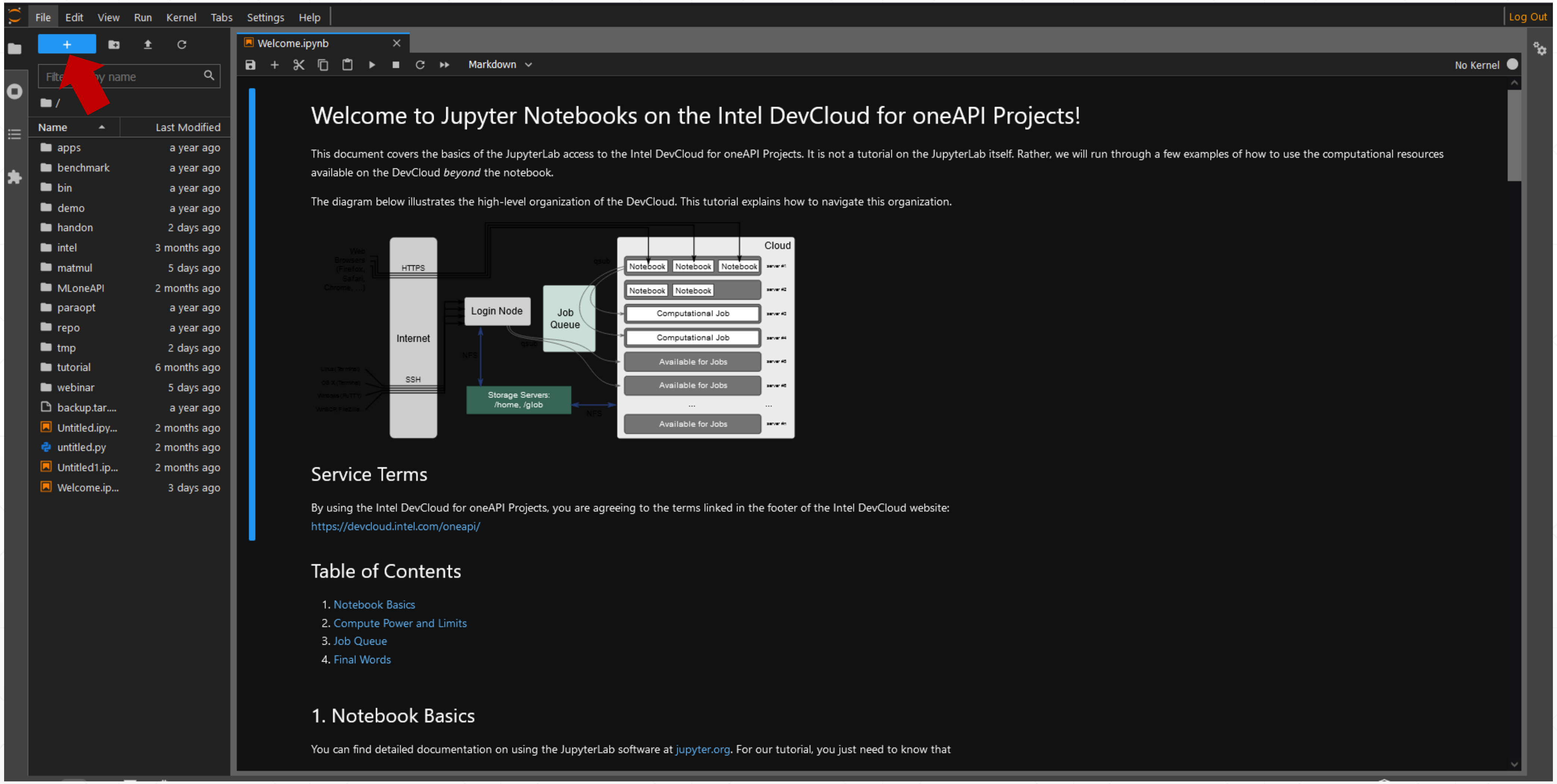
DevCloud Commands

Learn about the features of the compute nodes, data management, and how to submit, query, and delete your jobs.

Introduction to oneAPI and Essentials of Data Parallel C++

Use JupyterLab* to learn about how oneAPI can solve the challenges of programming in a heterogeneous world and understand the Data Parallel C++ (DPC++) language and programming model.

Jupyter Hub Interface

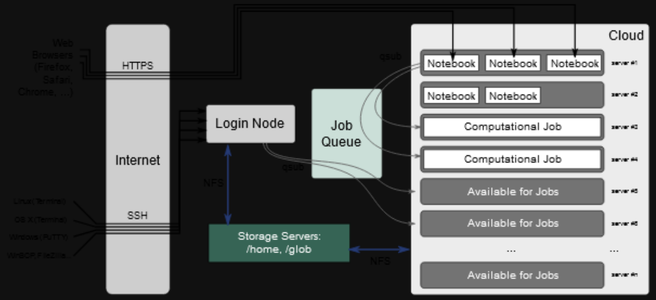


Name	Last Modified
apps	a year ago
benchmark	a year ago
bin	a year ago
demo	a year ago
handon	2 days ago
intel	3 months ago
matmul	5 days ago
MLoneAPI	2 months ago
paraopt	a year ago
repo	a year ago
tmp	2 days ago
tutorial	6 months ago
webinar	5 days ago
backup.tar....	a year ago
Untitled.ipy...	2 months ago
untitled.py	2 months ago
Untitled1.ip...	2 months ago
Welcome.ip...	3 days ago

Welcome to Jupyter Notebooks on the Intel DevCloud for oneAPI Projects!

This document covers the basics of the JupyterLab access to the Intel DevCloud for oneAPI Projects. It is not a tutorial on the JupyterLab itself. Rather, we will run through a few examples of how to use the computational resources available on the DevCloud *beyond* the notebook.

The diagram below illustrates the high-level organization of the DevCloud. This tutorial explains how to navigate this organization.



Service Terms

By using the Intel DevCloud for oneAPI Projects, you are agreeing to the terms linked in the footer of the Intel DevCloud website: https://devcloud.intel.com/oneapi/

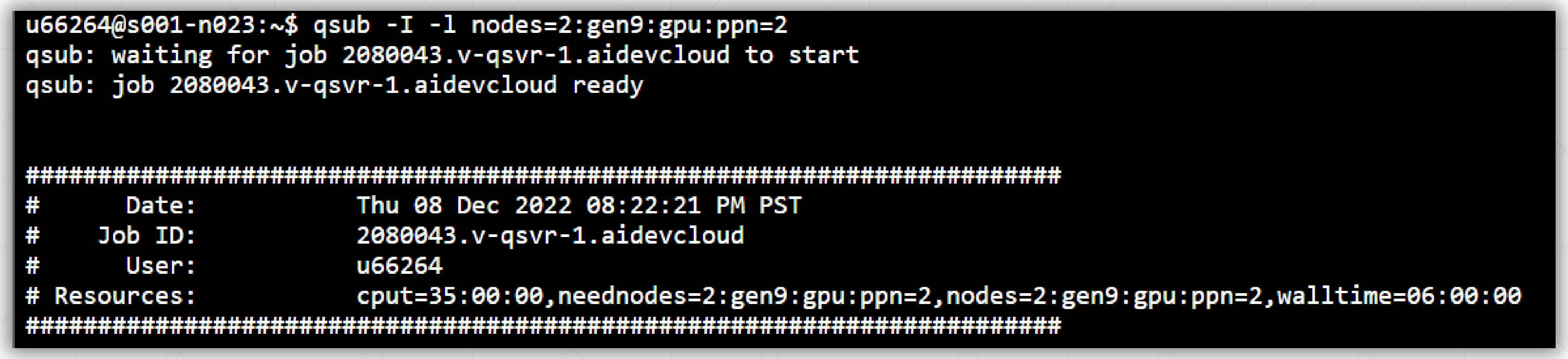
Table of Contents

1. Notebook Basics

You can find detailed documentation on using the JupyterLab software at jupyter.org. For our tutorial, you just need to know that

- New Launcher -> Terminal

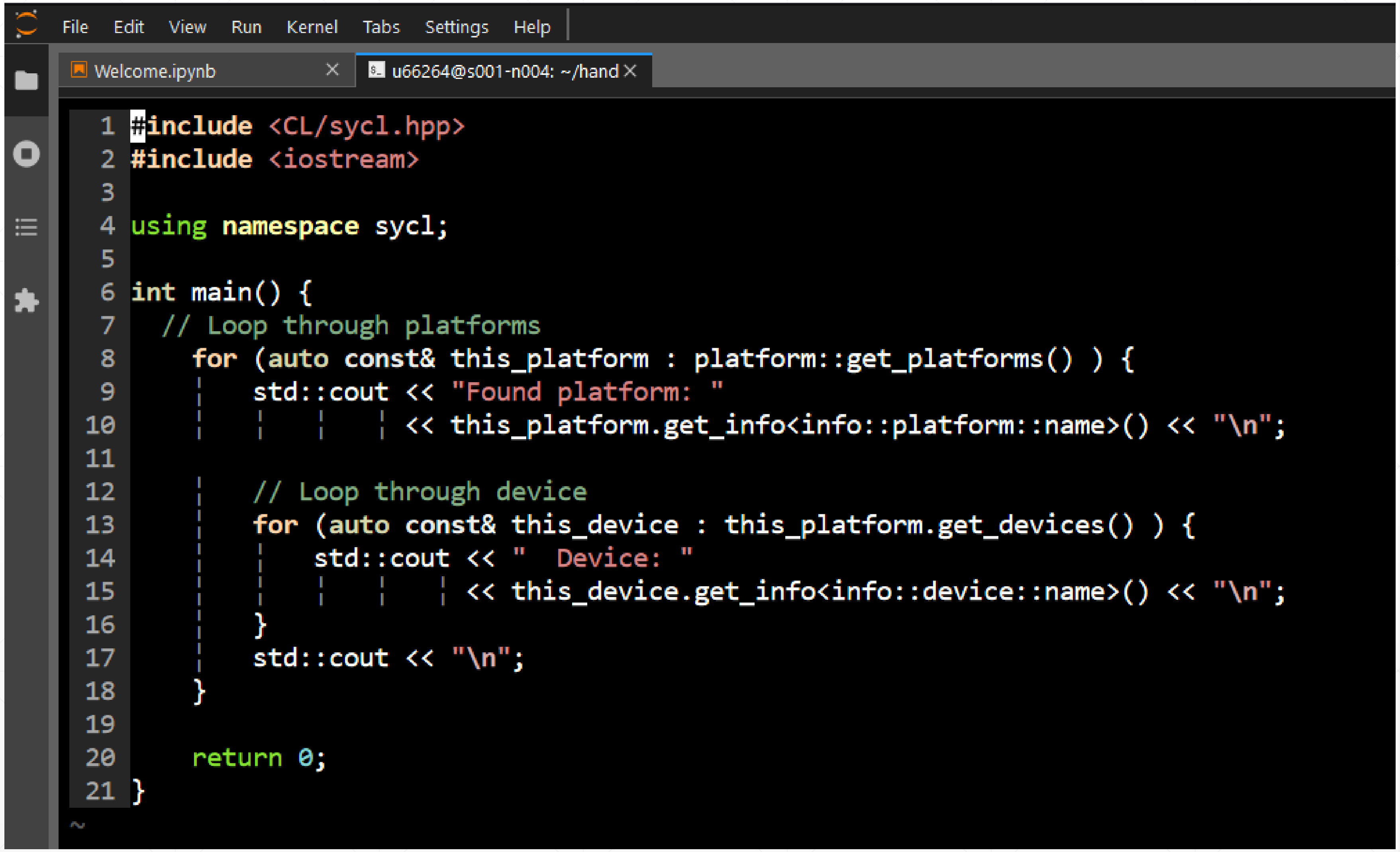
Request Interactive Jobs via PBS



```
u66264@s001-n023:~$ qsub -I -l nodes=2:gen9:gpu:ppn=2
qsub: waiting for job 2080043.v-qsvr-1.aidevcloud to start
qsub: job 2080043.v-qsvr-1.aidevcloud ready


########################################################################
#      Date:           Thu 08 Dec 2022 08:22:21 PM PST
#    Job ID:           2080043.v-qsvr-1.aidevcloud
#      User:           u66264
# Resources:           cput=35:00:00,neednodes=2:gen9:gpu:ppn=2,nodes=2:gen9:gpu:ppn=2,walltime=06:00:00
########################################################################
```

- Request node based on device properties
  - `qsub -I -l nodes=[nnodes]:[props]:ppn=[process_per_node]`
- Properties describing device class:
  - core / xeon / gpu / fpga
- Properties describing device name:
  - gen9 / gen 11 / aria10 / stratix10 / gold6128 / i9-10920x
- Properties describing purpose:
  - fpga_compile / fpga_runtime / renderkit

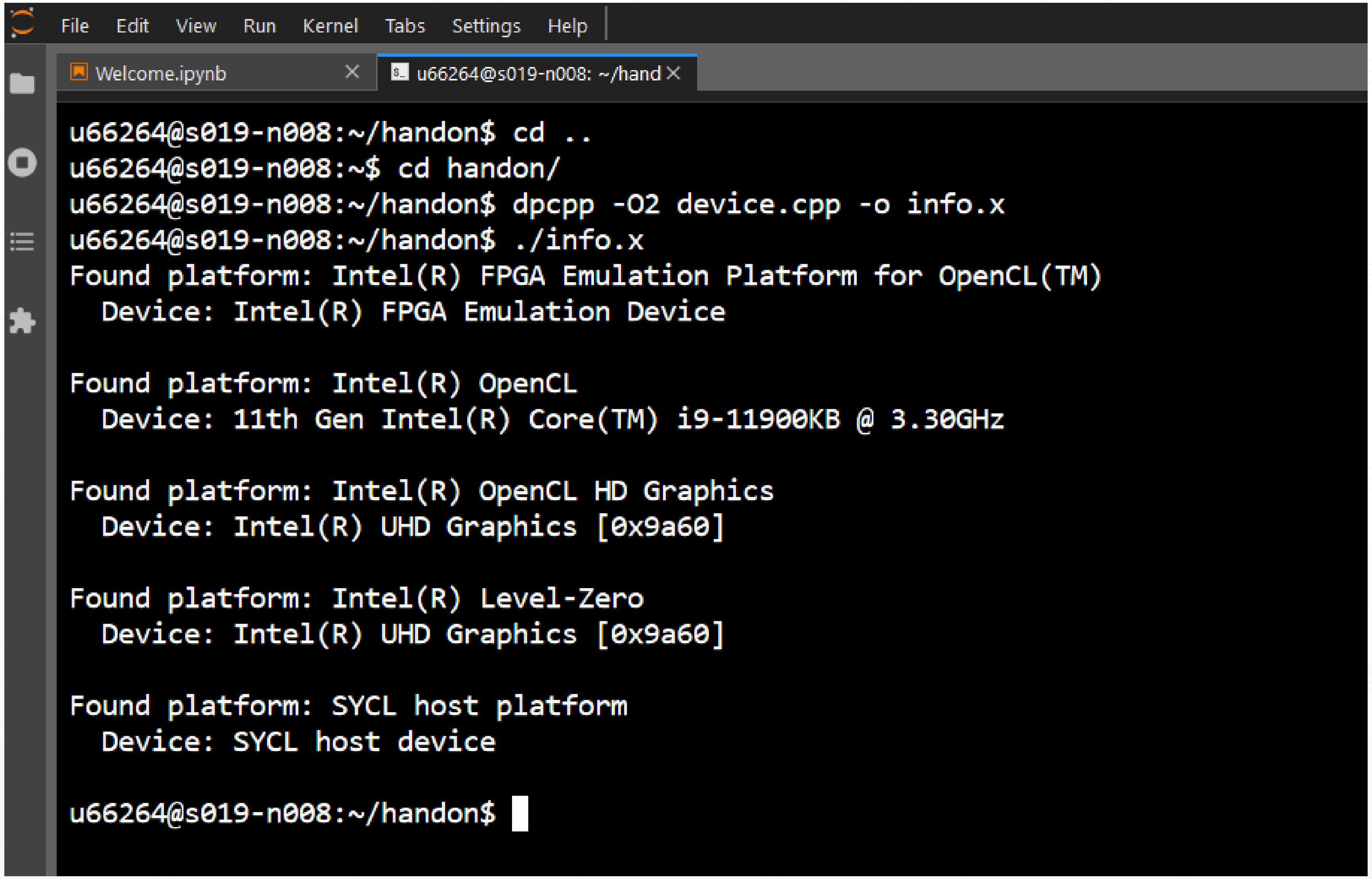
Device and Platform Discovery



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

using namespace sycl;

int main() {
  // Loop through platforms
    for (auto const& this_platform : platform::get_platforms() ) {
        std::cout << "Found platform: "
                  << this_platform.get_info<info::platform::name>() << "\n";

        // Loop through device
        for (auto const& this_device : this_platform.get_devices() ) {
            std::cout << "  Device: "
                      << this_device.get_info<info::device::name>() << "\n";
        }
        std::cout << "\n";
    }

    return 0;
}
```

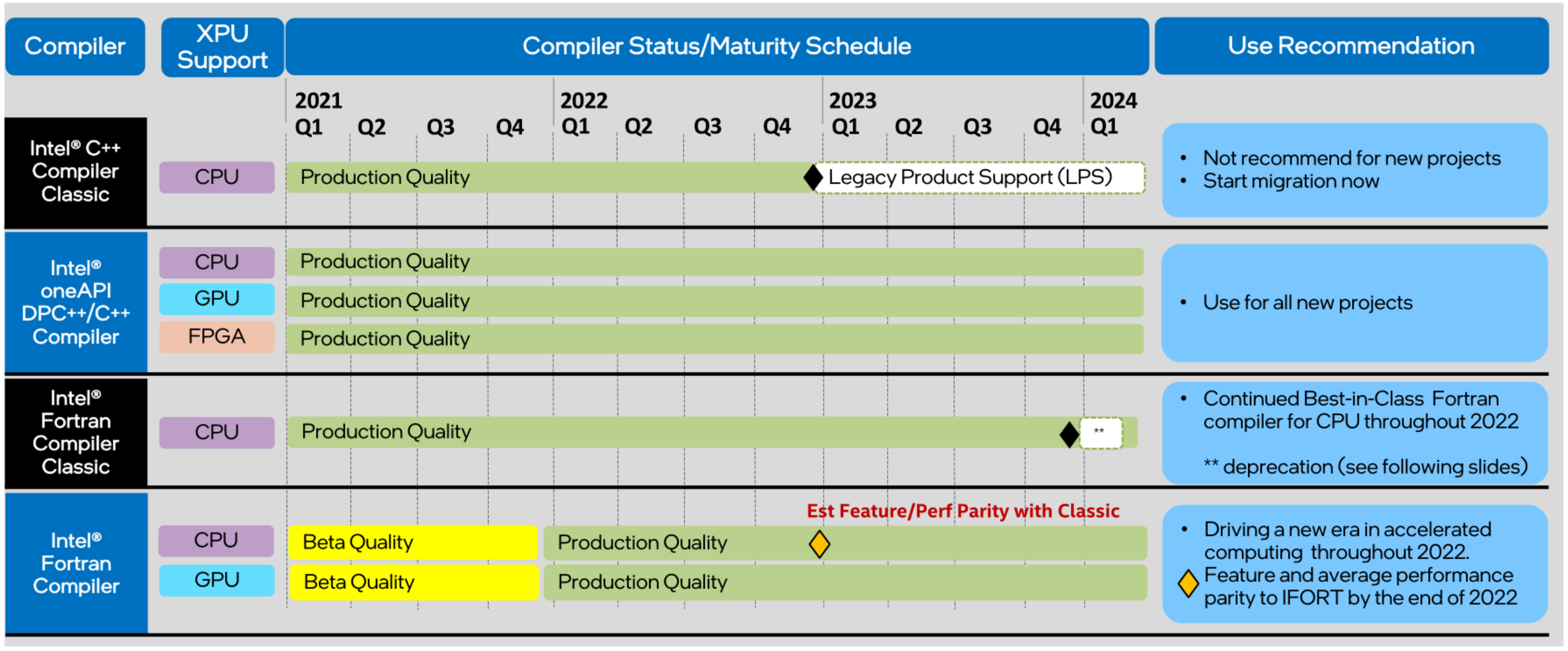
https://github.com/Apress/data-parallel-CPP/tree/main/samples/Ch12_device_information

Device and Platform Discovery (Gen 9)



```
u66264@s019-n008:~/handon$ cd ..
u66264@s019-n008:~$ cd handon/
u66264@s019-n008:~/handon$ dpcpp -O2 device.cpp -o info.x
u66264@s019-n008:~/handon$ ./info.x
Found platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
  Device: Intel(R) FPGA Emulation Device

Found platform: Intel(R) OpenCL
  Device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz

Found platform: Intel(R) OpenCL HD Graphics
  Device: Intel(R) UHD Graphics [0x9a60]

Found platform: Intel(R) Level-Zero
  Device: Intel(R) UHD Graphics [0x9a60]

Found platform: SYCL host platform
  Device: SYCL host device

u66264@s019-n008:~/handon$ 
```

Hands-on

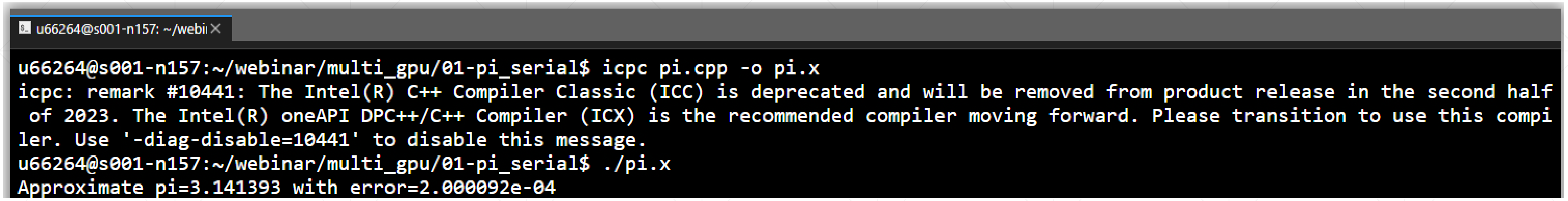


Compiler	XPU Support	Compiler Status/Maturity Schedule (2021 Q1 – 2024 Q1)	Use Recommendation
Intel® C++ Compiler Classic	CPU	Production Quality (2021 Q1 – 2022 Q4) ◆ Legacy Product Support (LPS) (2023 Q1 – 2024 Q1)	• Not recommend for new projects • Start migration now
Intel® oneAPI DPC++/C++ Compiler	CPU	Production Quality	• Use for all new projects
	GPU	Production Quality	
	FPGA	Production Quality	
Intel® Fortran Compiler Classic	CPU	Production Quality ◆ **	• Continued Best-in-Class Fortran compiler for CPU throughout 2022 ** deprecation (see following slides)
Intel® Fortran Compiler	CPU	Beta Quality (2021) / Production Quality (2022 –); Est Feature/Perf Parity with Classic ◇ (end of 2022)	• Driving a new era in accelerated computing throughout 2022. ◇ Feature and average performance parity to IFORT by the end of 2022
	GPU	Beta Quality (2021) / Production Quality (2022 –)	

https://www.ixpug.org/resources/intel-fortran-compilers-a-tradition-of-trusted-application-performance

- **Production quality**: ifx/icx/dpcpp++ have passed performance and validation tests, and are ready to use

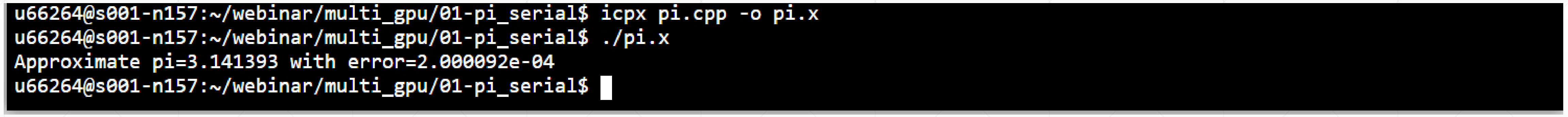
Estimation of PI: Serial



u66264@s001-n157: ~/webii

```
u66264@s001-n157:~/webinar/multi_gpu/01-pi_serial$ icpc pi.cpp -o pi.x
icpc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is deprecated and will be removed from product release in the second half
 of 2023. The Intel(R) oneAPI DPC++/C++ Compiler (ICX) is the recommended compiler moving forward. Please transition to use this compi
ler. Use '-diag-disable=10441' to disable this message.
u66264@s001-n157:~/webinar/multi_gpu/01-pi_serial$ ./pi.x
Approximate pi=3.141393 with error=2.000092e-04
```

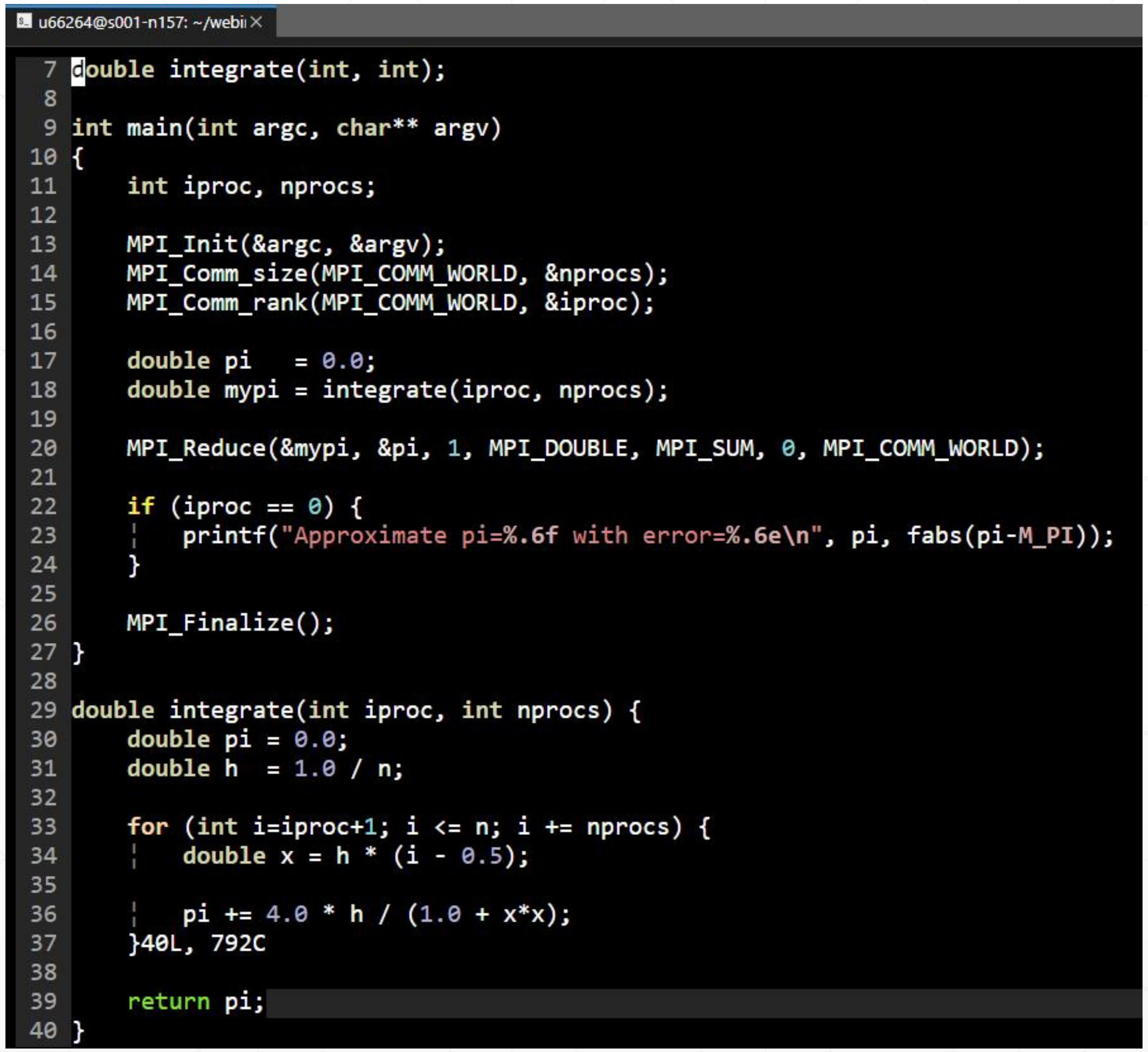
- Intel C++ compiler (ICPC) will be removed in future release.



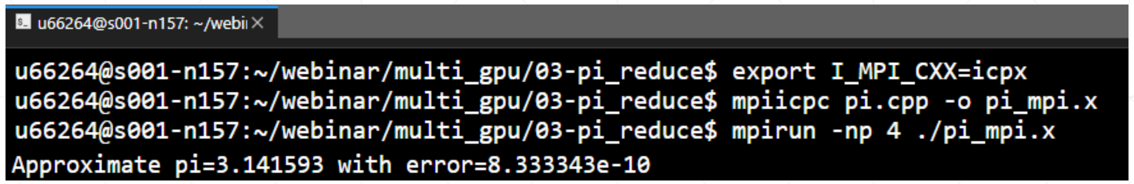
```
u66264@s001-n157:~/webinar/multi_gpu/01-pi_serial$ icpx pi.cpp -o pi.x
u66264@s001-n157:~/webinar/multi_gpu/01-pi_serial$ ./pi.x
Approximate pi=3.141393 with error=2.000092e-04
u66264@s001-n157:~/webinar/multi_gpu/01-pi_serial$
```

- New Intel C++ compiler (ICPX) is recommended

Estimation of PI: MPI_Reduce



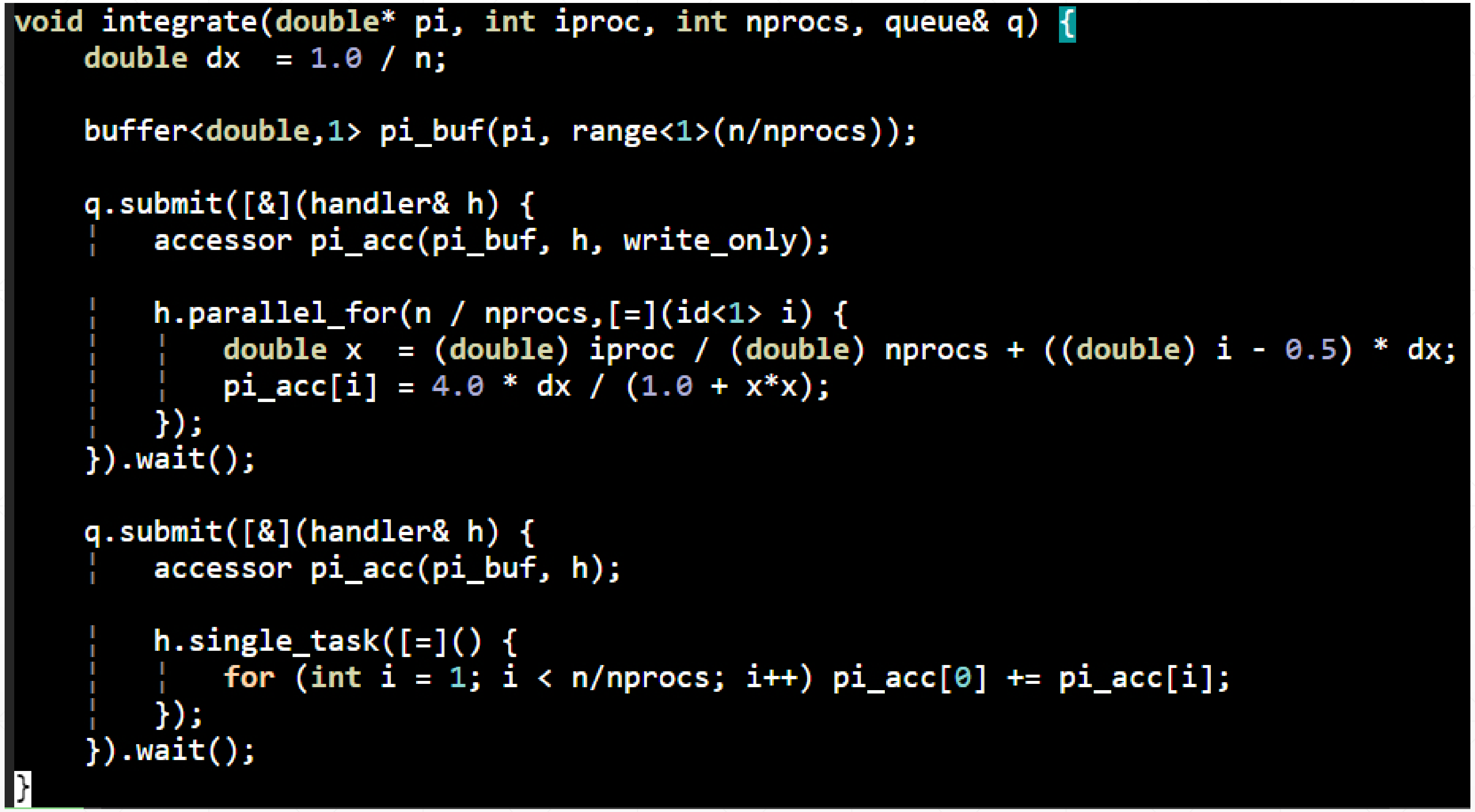
```
double integrate(int, int);

int main(int argc, char** argv)
{
    int iproc, nprocs;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);

    double pi   = 0.0;
    double mypi = integrate(iproc, nprocs);

    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (iproc == 0) {
        printf("Approximate pi=%.6f with error=%.6e\n", pi, fabs(pi-M_PI));
    }

    MPI_Finalize();
}

double integrate(int iproc, int nprocs) {
    double pi = 0.0;
    double h  = 1.0 / n;

    for (int i=iproc+1; i <= n; i += nprocs) {
        double x = h * (i - 0.5);

        pi += 4.0 * h / (1.0 + x*x);
    }40L, 792C

    return pi;
}
```



```
u66264@s001-n157:~/webinar/multi_gpu/03-pi_reduce$ export I_MPI_CXX=icpx
u66264@s001-n157:~/webinar/multi_gpu/03-pi_reduce$ mpiicpc pi.cpp -o pi_mpi.x
u66264@s001-n157:~/webinar/multi_gpu/03-pi_reduce$ mpirun -np 4 ./pi_mpi.x
Approximate pi=3.141593 with error=8.333343e-10
```

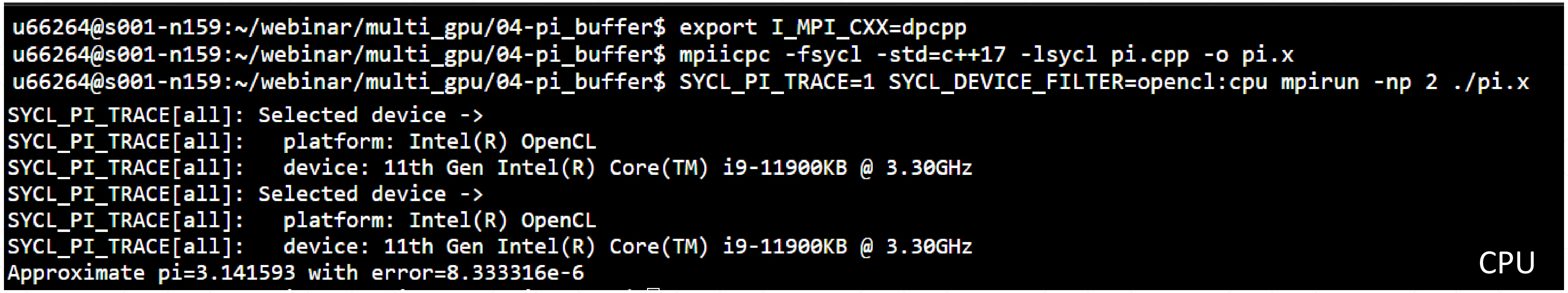
- Compiler wrapper for Intel MPI:
  - mpicc: GNU C compiler
  - mpicxx: GNU C++ compiler
  - mpif90: GNU Fortran compiler
  - mpiicc: Intel C compiler
  - mpiifort: Intel Fortran compiler
  - mpiicpc: Intel C++ compiler:
    - export I_MPI_CXX=[icpc|icpx|dpcpp]

Estimation of PI: DPC++ Buffer



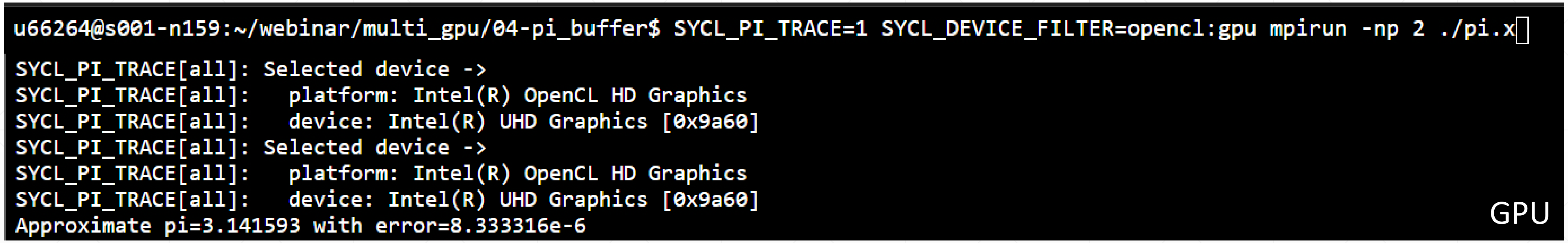
```
void integrate(double* pi, int iproc, int nprocs, queue& q) {
    double dx  = 1.0 / n;

    buffer<double,1> pi_buf(pi, range<1>(n/nprocs));

    q.submit([&](handler& h) {
        accessor pi_acc(pi_buf, h, write_only);

        h.parallel_for(n / nprocs,[=](id<1> i) {
            double x  = (double) iproc / (double) nprocs + ((double) i - 0.5) * dx;
            pi_acc[i] = 4.0 * dx / (1.0 + x*x);
        });
    }).wait();

    q.submit([&](handler& h) {
        accessor pi_acc(pi_buf, h);

        h.single_task([=]() {
            for (int i = 1; i < n/nprocs; i++) pi_acc[0] += pi_acc[i];
        });
    }).wait();
}
```

Estimation of PI: Comping MPI/DPC++ Code



```
u66264@s001-n159:~/webinar/multi_gpu/04-pi_buffer$ export I_MPI_CXX=dpcpp
u66264@s001-n159:~/webinar/multi_gpu/04-pi_buffer$ mpiicpc -fsycl -std=c++17 -lsycl pi.cpp -o pi.x
u66264@s001-n159:~/webinar/multi_gpu/04-pi_buffer$ SYCL_PI_TRACE=1 SYCL_DEVICE_FILTER=opencl:cpu mpirun -np 2 ./pi.x
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Approximate pi=3.141593 with error=8.333316e-6
```

CPU

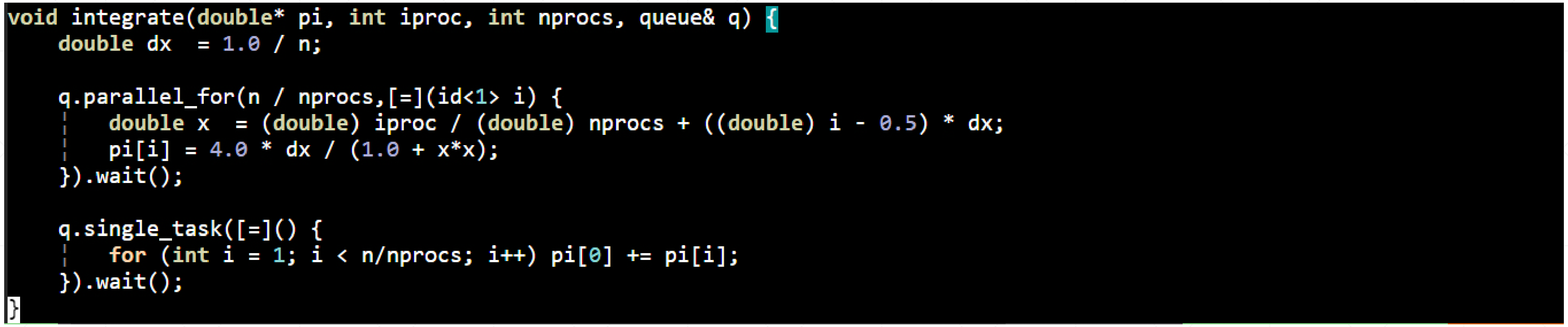


```
u66264@s001-n159:~/webinar/multi_gpu/04-pi_buffer$ SYCL_PI_TRACE=1 SYCL_DEVICE_FILTER=opencl:gpu mpirun -np 2 ./pi.x
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL HD Graphics
SYCL_PI_TRACE[all]:   device: Intel(R) UHD Graphics [0x9a60]
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL HD Graphics
SYCL_PI_TRACE[all]:   device: Intel(R) UHD Graphics [0x9a60]
Approximate pi=3.141593 with error=8.333316e-6
```

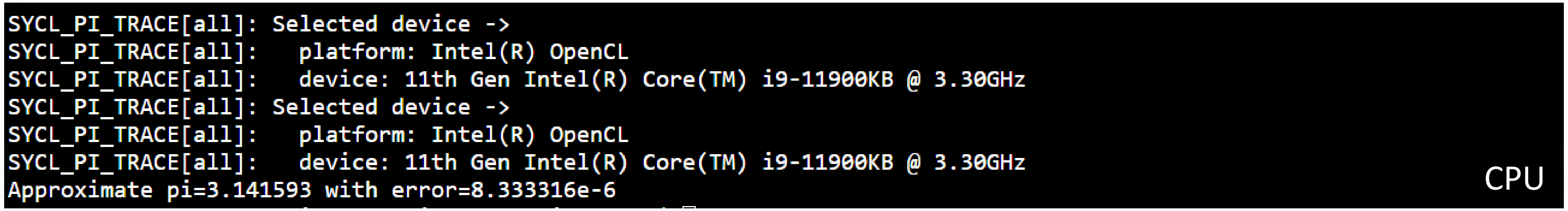
GPU

- Debug message:
  - SYCL_PI_TRACE=1
- Device selection:
  - SYCL_DEVICE_FILTER= [backend:device]
  - Supported backend: level_zero | opencl | cuda | hip
  - Supoorted device: cpu | gpu | acc

Estimation of PI: DPC++ USM

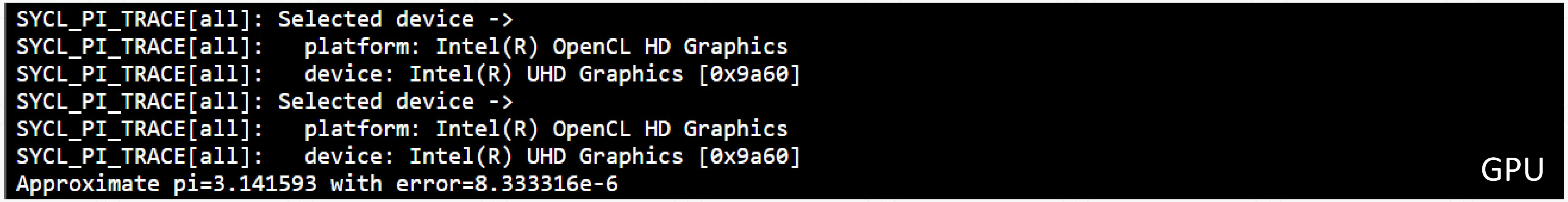


```
void integrate(double* pi, int iproc, int nprocs, queue& q) {
    double dx  = 1.0 / n;

    q.parallel_for(n / nprocs,[=](id<1> i) {
        double x  = (double) iproc / (double) nprocs + ((double) i - 0.5) * dx;
        pi[i] = 4.0 * dx / (1.0 + x*x);
    }).wait();

    q.single_task([=]() {
        for (int i = 1; i < n/nprocs; i++) pi[0] += pi[i];
    }).wait();
}
```



```
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL
SYCL_PI_TRACE[all]:   device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Approximate pi=3.141593 with error=8.333316e-6
```

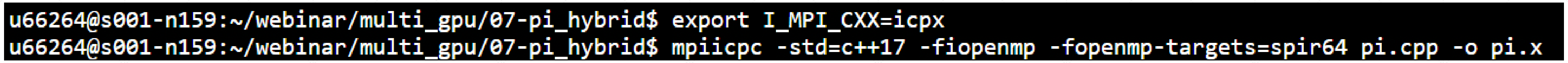
CPU



```
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL HD Graphics
SYCL_PI_TRACE[all]:   device: Intel(R) UHD Graphics [0x9a60]
SYCL_PI_TRACE[all]: Selected device ->
SYCL_PI_TRACE[all]:   platform: Intel(R) OpenCL HD Graphics
SYCL_PI_TRACE[all]:   device: Intel(R) UHD Graphics [0x9a60]
Approximate pi=3.141593 with error=8.333316e-6
```

GPU

Estimation of PI: MPI + OpenMP Offload



```
u66264@s001-n159:~/webinar/multi_gpu/07-pi_hybrid$ export I_MPI_CXX=icpx
u66264@s001-n159:~/webinar/multi_gpu/07-pi_hybrid$ mpiicpc -std=c++17 -fiopenmp -fopenmp-targets=spir64 pi.cpp -o pi.x
```

- Select target device
  - OMP_TARGET_OFFLOAD = mandatory | disable | default
    - mandatory: run target region on GPU
    - disabled: run target regions on CPU
    - default: run target region on GPU if available, else false back to CPU
- Select OpenMP backend
  - LIBOMPTARGET_PLUGIN = opencl | level0
- Performance profiling on GPU
  - LIBOMPTARGET_PLUGIN_PROFILE = T | F

Porting Guide

Compiler options	icpc	icx
Disable optimization	-O0	-O0
Optimization for speed (no code size increase)	-O1	-O1
Optimization for speed	-O2	-O2 **-xSAPPHIRERAPIDS**
High-level loop optimization	-O3	-O3 **-xSAPPHIRERAPIDS**
AVX512 vector width	-qopt-zmm-usage=high	**-mprefer-vector-width=512**
Floating point optimization	-fp model fast=2	-fp model fast=2 **-assume nan_compares**
Micoarchitecture optimization	-xSAPPHIRERAPIDS	-xSAPPHIRERAPIDS
Interprocedural optimization	-ipo	**-flto**
OpenMP support	-qopenmp	**-fiopenmp**
Just-in-time compilation	N/A	**-fopenmp-targets=spir64**
Ahead-of-time compilation	N/A	**-fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device xehp"**
Optimization report	-qopt-report=5	**-qopt-report=3**
Optimization phase report	-qopt-report-phase=loop	N/A
Optimization report filter	-qopt-report-routine=stencile	N/A

- https://www.intel.com/content/www/us/en/developer/articles/guide/porting-guide-for-icc-users-to-dpcpp-or-icx.html