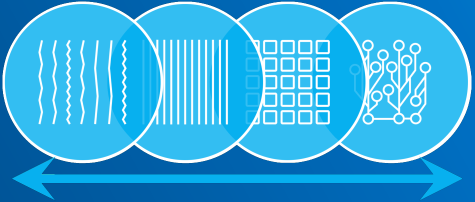
ONEAPI

SINGLE PROGRAMMING MODEL TO DELIVER CROSS-ARCHITECTURE PERFORMANCE



MODULE 1
GETTING STARTED WITH ONEAPI

COLFAX RESEARCH

intel®

ONEAPI TRAINING SERIES

- Module 1: Getting Started with oneAPI
- Module 2: Introduction to DPC++
- Module 3: Fundamentals of DPC++, part 1 of 2
- Module 4: Fundamentals of DPC++, part 2 of 2
- Modules 5+: Deeper dives into specific DPC++ features, oneAPI libraries and tools

https://oneapi.com

https://software.intel.com/en-us/oneapi

https://tinyurl.com/book-dpcpp

http://tinyurl.com/oneapimodule?1

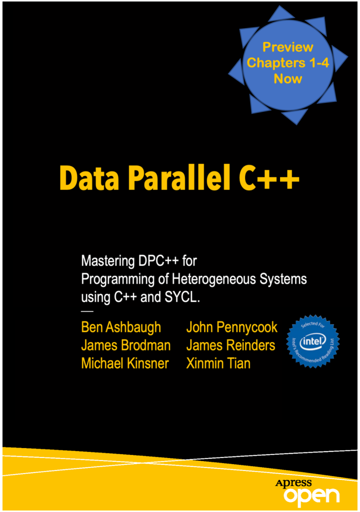
1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

RESOURCES

- ▹ Book (Chapters 1-4 Preview)
- ▹ oneAPI Toolkit(s)
- ▹ Training, Support, Forums, Example Code

All available
Free

https://software.intel.com/en-us/oneapi



Preview Chapters 1-4 Now

Data Parallel C++

Mastering DPC++ for
Programming of Heterogeneous Systems
using C++ and SYCL.

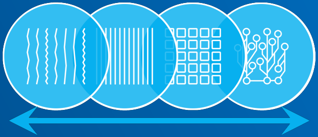
Ben Ashbaugh
James Brodman
Michael Kinsner
John Pennycook
James Reinders
Xinmin Tian

Selected For Intel's Recommended Reading List

Apress open

https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?1

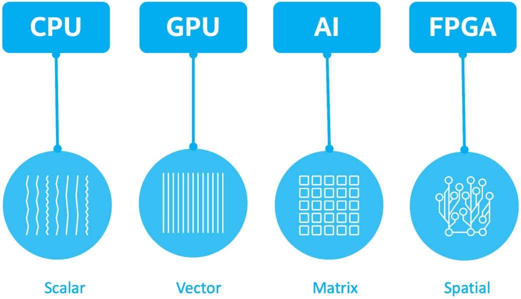
§1. PROGRAMMING IN A HETEROGENEOUS WORLD



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

DIVERSE WORKLOADS DEMAND DIVERSE ARCHITECTURES

The future is a diverse mix of scalar, vector, matrix, and spatial architectures deployed in CPU, GPU, AI, FPGA, and other accelerators.



CHALLENGE: PROGRAMMING IN A HETEROGENEOUS WORLD

- Diverse set of data-centric hardware
- No common programming language or APIs
- Inconsistent tool support across platforms
- Proprietary solutions on individual platforms
- Each platform requires unique software investment

CHALLENGE: PROGRAMMING IN A HETEROGENEOUS WORLD

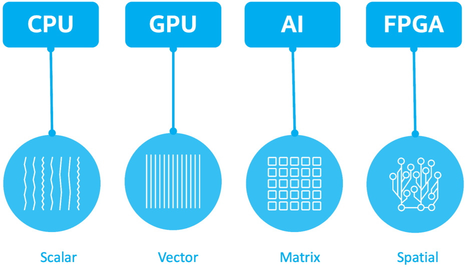
- Diverse set of data-centric hardware
- No common programming language or APIs
- Inconsistent tool support across platforms
- Proprietary solutions on individual platforms
- Each platform requires unique software investment

CHALLENGE: PROGRAMMING IN A HETEROGENEOUS WORLD

- Diverse set of data-centric hardware
- No common programming language or APIs
- Inconsistent tool support across platforms
- Proprietary solutions on individual platforms
- Each platform requires unique software investment

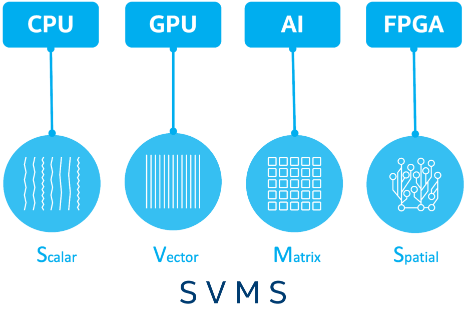
CHALLENGE: PROGRAMMING IN A HETEROGENEOUS WORLD

- Diverse set of data-centric hardware
- No common programming language or APIs
- Inconsistent tool support across platforms
- Proprietary solutions on individual platforms
- Each platform requires unique software investment

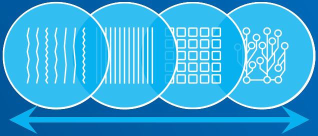


CHALLENGE: PROGRAMMING IN A HETEROGENEOUS WORLD

- Diverse set of data-centric hardware
- No common programming language or APIs
- Inconsistent tool support across platforms
- Proprietary solutions on individual platforms
- Each platform requires unique software investment



§2. HOW ONEAPI ADDRESSES OUR HETEROGENEOUS WORLD



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

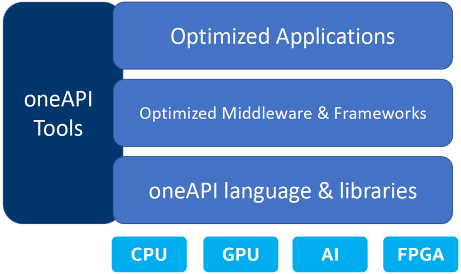
INTEL'S ONEAPI CORE CONCEPT

- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

INTEL'S ONEAPI CORE CONCEPT

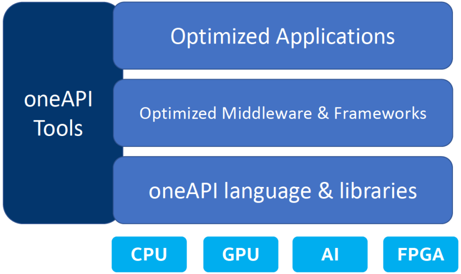
- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism


oneAPI Tools
Optimized Applications
Optimized Middleware & Frameworks
oneAPI language & libraries
CPU
GPU
AI
FPGA

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

INTEL'S ONEAPI CORE CONCEPT

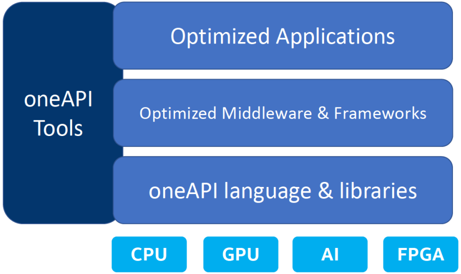
- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism


oneAPI Tools
Optimized Applications
Optimized Middleware & Frameworks
oneAPI language & libraries
CPU
GPU
AI
FPGA

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

INTEL'S ONEAPI CORE CONCEPT

- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism



oneAPI Tools

Optimized Applications

Optimized Middleware & Frameworks

oneAPI language & libraries

CPU

GPU

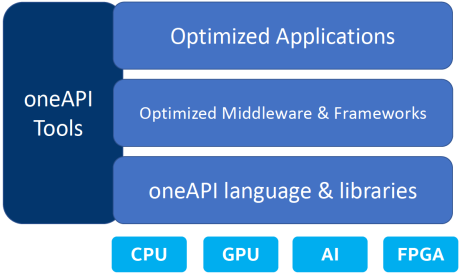
AI

FPGA

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

INTEL'S ONEAPI CORE CONCEPT

- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism

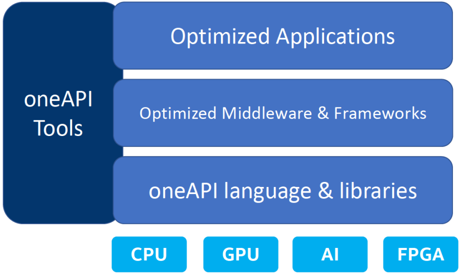


oneAPI Tools
Optimized Applications
Optimized Middleware & Frameworks
oneAPI language & libraries
CPU
GPU
AI
FPGA

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

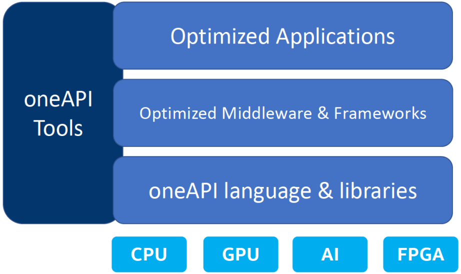
INTEL'S ONEAPI CORE CONCEPT

- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism


oneAPI Tools
Optimized Applications
Optimized Middleware & Frameworks
oneAPI language & libraries
CPU
GPU
AI
FPGA

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

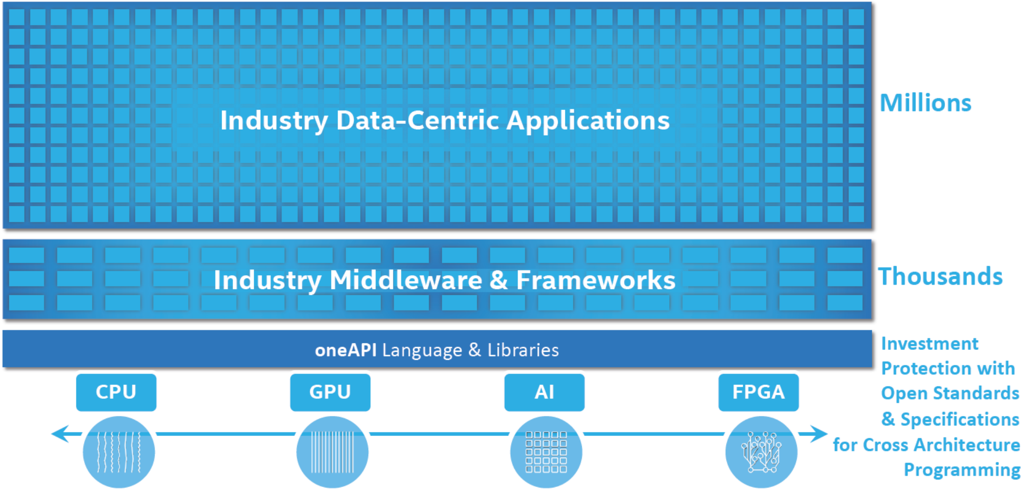
INTEL'S ONEAPI CORE CONCEPT


oneAPI Tools
Optimized Applications
Optimized Middleware & Frameworks
oneAPI language & libraries
CPU
GPU
AI
FPGA

- Project oneAPI delivers a unified programming model to simplify development across diverse architectures
- Common developer experience across SVMS
- Uncompromised native high-level language performance
- Unified language and libraries for expressing parallelism

- Support for CPU, GPU, AI, and FPGA
- Based on industry standards and open specifications

PROTECT PROGRAMMING INVESTMENTS



Industry Data-Centric Applications

Millions

Industry Middleware & Frameworks

Thousands

oneAPI Language & Libraries

CPU

GPU

AI

FPGA

Investment
Protection with
Open Standards
& Specifications
for Cross Architecture
Programming

GOOD PLAN: LET ALL LIGHTS SHINE

- Allowing all ?PUs to shine should yield better results than programming approaches that focus on highlighting a particular PU over all others.
- Programmers want to write a single portable program that uses ALL resources in the heterogeneous platform.

GOOD PLAN: LET ALL LIGHTS SHINE

- Allowing all ?PUs to shine should yield better results than programming approaches that focus on highlighting a particular PU over all others.
- Programmers want to write a single portable program that uses ALL resources in the heterogeneous platform.

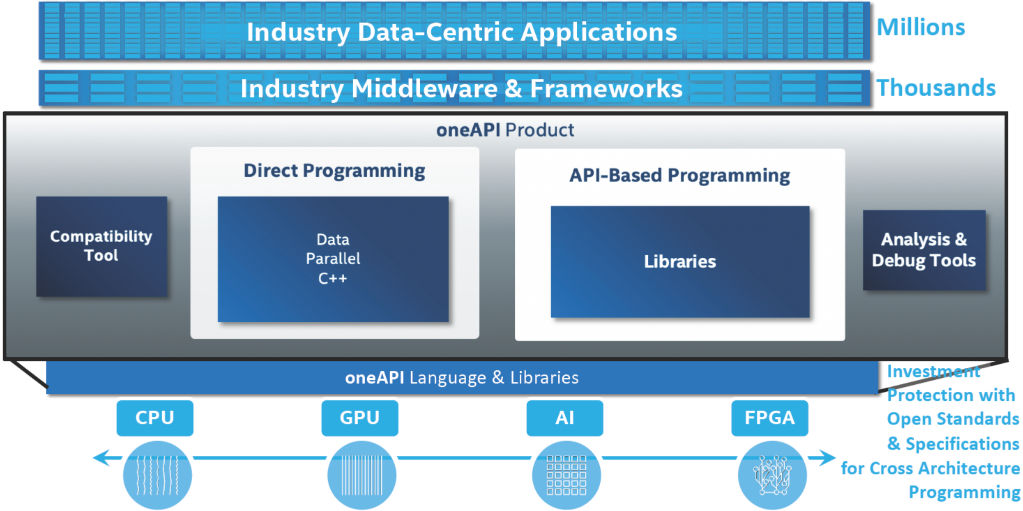
GOOD PLAN: ?PU NEEDS CPUS

- CPUs excel at serial.
- Parallel programmers learn to hate slow serial processing, because it destroys scaling at an alarming rate thanks to Amdahl's Law.
- Any investment in speeding up an application, is easily destroyed if the serial part is compromised — even if the serial part is only 0.001% of the application.
- Even using full speed for 99.999% of compute with 20K PUs, a 1/3rd speed serial processing finds that Amdahl's Law tells us that we'll see no more than 68% of the performance that we could obtain with full speed serial processing.
- Amdahl's Law math: ((99999/30000)+1) / ((99999/30000)+3)

GOOD PLAN: ?PU NEEDS CPUS

- CPUs excel at serial.
- Parallel programmers learn to hate slow serial processing, because it destroys scaling at an alarming rate thanks to Amdahl's Law.
- Any investment in speeding up an application, is easily destroyed if the serial part is compromised — even if the serial part is only 0.001% of the application.
- Even using full speed for 99.999% of compute with 20K PUs, a 1/3rd speed serial processing finds that Amdahl's Law tells us that we'll see no more than 68% of the performance that we could obtain with full speed serial processing.
- Amdahl's Law math: ((99999/30000)+1) / ((99999/30000)+3)

ONEAPI FOR CROSS-ARCHITECTURE PERFORMANCE



Industry Data-Centric Applications

Millions

Industry Middleware & Frameworks

Thousands

oneAPI Product

Compatibility Tool

Direct Programming

Data Parallel C++

API-Based Programming

Libraries

Analysis & Debug Tools

oneAPI Language & Libraries

CPU

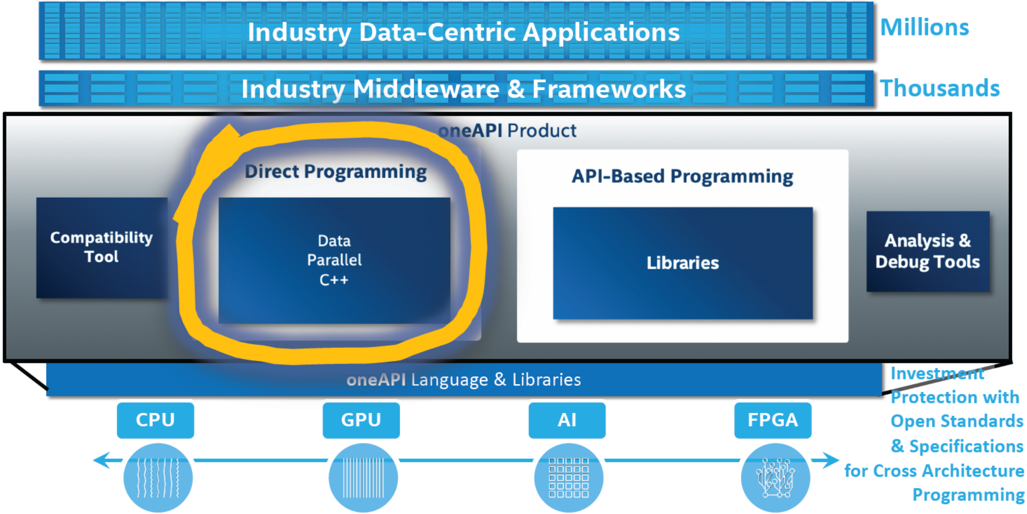
GPU

AI

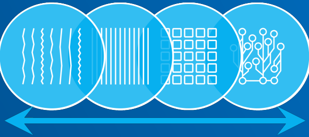
FPGA

Investment Protection with Open Standards & Specifications for Cross Architecture Programming

ONEAPI FOR CROSS-ARCHITECTURE PERFORMANCE


Industry Data-Centric Applications
Millions
Industry Middleware & Frameworks
Thousands
oneAPI Product
Direct Programming
Compatibility Tool
Data Parallel C++
API-Based Programming
Libraries
Analysis & Debug Tools
oneAPI Language & Libraries
CPU
GPU
AI
FPGA
Investment Protection with Open Standards & Specifications for Cross Architecture Programming

§3. HELLO DOUBLER - SIMPLE DPC++ CODING EXAMPLE



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

"HELLO DOUBLER" DPC++

```
#include <CL/sycl.hpp>
#include <iostream>
#include <array>
#include <cstdio>
#define SIZE 1024

int main() {
  std::array<int, SIZE> myArray;
  for (int i = 0; i<SIZE; ++i)
    myArray[i] = i;
```

```
  // cl::sycl:: adds clarity for teaching
  // but is not how you are likely to code...
  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
  {
    cl::sycl::queue myQ;    /* use defaults today */
    /* (queue parameters possible - future topic) */

    cl::sycl::range<1> mySize{SIZE};
    cl::sycl::buffer<int, 1> bufferA(myArray.data(), mySize);

    myQ.submit([&](cl::sycl::handler &myHandle) {
      auto deviceAccessorA =
        bufferA.get_access<cl::sycl::access::mode::read_write>(myHandle);
      myHandle.parallel_for<class uniqueID>(mySize,
        [=](cl::sycl::id<1> index)
          {
            deviceAccessorA[index] *= 2;
          }
      );
    });
  }
  printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
}
```

NAMESPACE CL::SYCL::

```
// cl::sycl:: adds clarity for teaching
// but is not how you are likely to code...
printf("Value at start: myArray[42] is %d.\n",myArray[42]);
{
  cl::sycl::queue myQ;    /* use defaults today */
  /* (queue parameters possible - future topic) */
  cl::sycl::range<1> mySize{SIZE};
  cl::sycl::buffer<int, 1> bufferA(myArray.data(), mySize);

  myQ.submit([&](cl::sycl::handler &myHandle) {
    auto deviceAccessorA =
      bufferA.get_access<cl::sycl::access::mode::read_write>(myHandle);
    myHandle.parallel_for<class uniqueID>(mySize,
      [=](cl::sycl::id<1> index)
        {
          deviceAccessorA[index] *= 2;
        }
    );
  });
}
printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- cl::sycl::

NAMESPACE CL::SYCL::

```
using namespace cl::sycl;

printf("Value at start: myArray[42] is %d.\n",myArray[42]);
{
  queue myQ;    /* use defaults today */
  /* (queue parameters possible - future topic) */
  range<1> mySize{SIZE};
  buffer<int, 1> bufferA(myArray.data(), mySize);

  myQ.submit([&](handler &myHandle) {
    auto deviceAccessorA =
      bufferA.get_access<access::mode::read_write>(myHandle);
    myHandle.parallel_for<class uniqueID>(mySize,
      [=](id<1> index)
        {
          deviceAccessorA[index] *= 2;
        }
    );
  });
}
printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

▷ that's better!

"HELLO DOUBLER" DPC++

```
using namespace cl::sycl;

printf("Value at start: myArray[42] is %d.\n",myArray[42]);
{
  queue myQ;    /* use defaults today */
  /* (queue parameters possible - future topic) */
  range<1> mySize{SIZE};
  buffer<int, 1> bufferA(myArray.data(), mySize);

  myQ.submit([&](handler &myHandle) {
    auto deviceAccessorA =
      bufferA.get_access<access::mode::read_write>(myHandle);
    myHandle.parallel_for<class uniqueID>(mySize,
      [=](id<1> index)
        {
          deviceAccessorA[index] *= 2;
        }
    );
  });
}
printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- Full power of C++
- DPC++ extends C++ with SYCL and more
- Syntax is pure C++, no new keywords
- Kernels are Key Data Parallel Programming Construct
- Cross-platform portability
- Optimizing compilers boost performance
- Full programmer control over performance

"HELLO DOUBLER" DPC++

```
using namespace cl::sycl;

printf("Value at start: myArray[42] is %d.\n",myArray[42]);
{
  queue myQ;    /* use defaults today */
  /* (queue parameters possible - future topic) */
  range<1> mySize{SIZE};
  buffer<int, 1> bufferA(myArray.data(), mySize);

  myQ.submit([&](handler &myHandle) {
    auto deviceAccessorA =
      bufferA.get_access<access::mode::read_write>(myHandle);
    myHandle.parallel_for<class uniqueID>(mySize,
      [=](id<1> index)
        {
          deviceAccessorA[index] *= 2;
        }
    );
  });
}
printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- Full power of C++
- DPC++ extends C++ with SYCL and more
- Syntax is pure C++, no new keywords
- Kernels are Key Data Parallel Programming Construct
- Cross-platform portability
- Optimizing compilers boost performance
- Full programmer control over performance

"HELLO DOUBLER" DPC++

```
$ make doubler2
dpcpp doubler2.cpp -o doubler2

$ ./doubler2
Value at start:  myArray[42] is 42.
Value at finish: myArray[42] is 84.
```

- Doubler, like other DPC++ kernels, can be mapped to all architectures.
- The suitability of each architecture is algorithm dependent.

42 DOUBLED IS 84

```
using namespace cl::sycl;

printf("Value at start: myArray[42] is %d.\n",myArray[42]);
{
  queue myQ;    /* use defaults today */
  /* (queue parameters possible - future topic) */
  range<1> mySize{SIZE};
  buffer<int, 1> bufferA(myArray.data(), mySize);

  myQ.submit([&](handler &myHandle) {
    auto deviceAccessorA =
      bufferA.get_access<access::mode::read_write>(myHandle);
    myHandle.parallel_for<class uniqueID>(mySize,
      [=](id<1> index)
        {
          deviceAccessorA[index] *= 2;
        }
    );
  });
}
printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- ▷ myArray[42] starts as 42
- ▷ afterwards it is 84

DPC++ PROVIDES THE MEANS!

- Doubler, like other DPC++ kernels, can be mapped to all architectures.
- The suitability of each architecture is algorithm dependent.
- Balancing performance, portability, and productivity during application development is a challenge we all face.
- DPC++ provides all of the tools required to maintain both generic portable code, and optimized target-specific code, using a single high-level programming language.

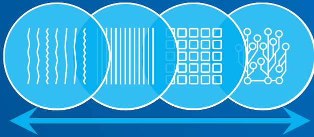
DPC++ PROVIDES THE MEANS!

- Doubler, like other DPC++ kernels, can be mapped to all architectures.
- The suitability of each architecture is algorithm dependent.
- Balancing performance, portability, and productivity during application development is a challenge we all face.
- DPC++ provides all of the tools required to maintain both generic portable code, and optimized target-specific code, using a single high-level programming language.

DPC++ PROVIDES THE MEANS!

- Doubler, like other DPC++ kernels, can be mapped to all architectures.
- The suitability of each architecture is algorithm dependent.
- Balancing performance, portability, and productivity during application development is a challenge we all face.
- DPC++ provides all of the tools required to maintain both generic portable code, and optimized target-specific code, using a single high-level programming language.

§4. WHAT IS SYCL?



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

SYCL

SYCL is an industry-wide standardization effort to define cross-platform data parallelism support for C++.

- pronounced `sickle'  `sick ell' /ˈsik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's contributes back new additions

SYCL

SYCL is an industry-wide standardization effort to define cross-platform data parallelism support for C++.

- pronounced `sickle'   `sick ell'   /'sik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's contributes back new additions

SYCL

SYCL is an industry-wide standardization effort to define cross-platform data parallelism support for C++.

- pronounced `sickle'   `sick ell'   /ˈsik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's contributes back new additions

SYCL

SYCL is an industry-wide standardization effort to define cross-platform data parallelism support for C++.

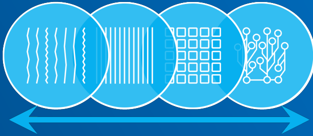
- pronounced `sickle'    `sick ell'   /ˈsik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's contributes back new additions

SYCL

SYCL is an industry-wide standardization effort to define cross-platform data parallelism support for C++.

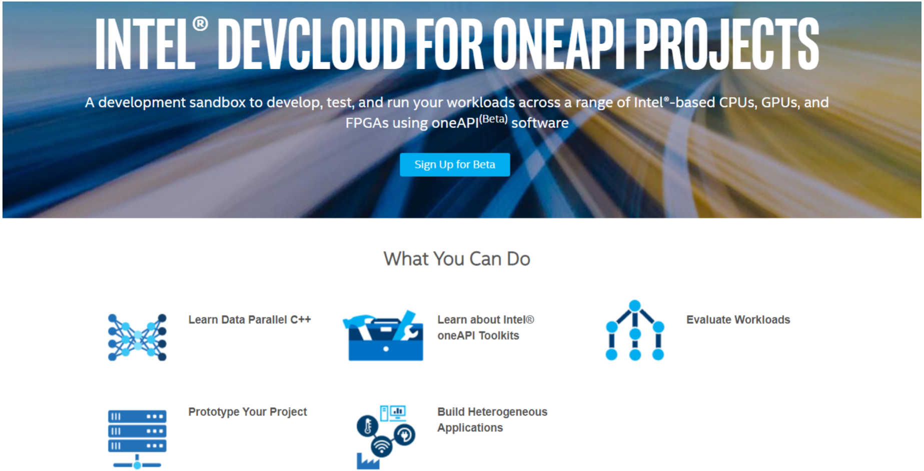
- pronounced `sickle'    `sick ell'   /ˈsik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's contributes back new additions

§5. DEVCLOUD - TRY ONEAPI EASILY



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. What is Data Parallel C++?

DEVCLOUD



INTEL® DEVCLOUD FOR ONEAPI PROJECTS

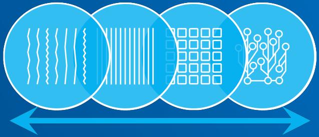
A development sandbox to develop, test, and run your workloads across a range of Intel®-based CPUs, GPUs, and FPGAs using oneAPI$^{(Beta)}$ software

What You Can Do

- Learn Data Parallel C++
- Learn about Intel® oneAPI Toolkits
- Evaluate Workloads
- Prototype Your Project
- Build Heterogeneous Applications

https://software.intel.com/en-us/devcloud/oneapi

§6. ONEAPI - WHY AND HOW



1 Programming in a Heterogeneous World

2 How oneAPI addresses our Heterogeneous World

3 Hello Doubler - simple DPC++ coding example

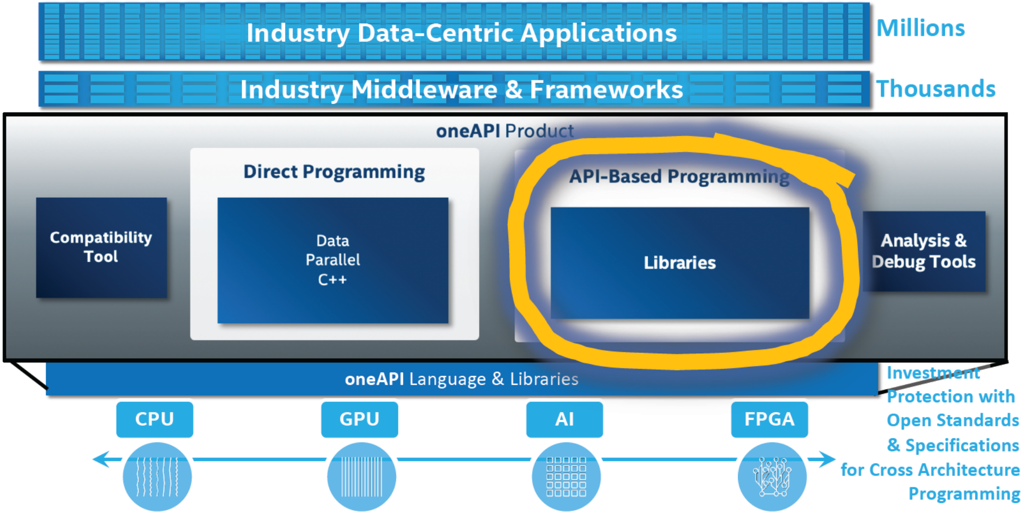
4 What is SYCL?

5 DevCloud - Try oneAPI easily

6 **oneAPI - Why and How**

7 What is Data Parallel C++?

ONEAPI FOR CROSS-ARCHITECTURE PERFORMANCE



Industry Data-Centric Applications

Millions

Industry Middleware & Frameworks

Thousands

oneAPI Product

Compatibility Tool

Direct Programming

Data Parallel C++

API-Based Programming

Libraries

Analysis & Debug Tools

oneAPI Language & Libraries

CPU

GPU

AI

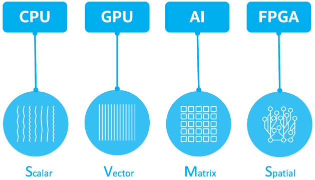
FPGA

Investment Protection with Open Standards & Specifications for Cross Architecture Programming

POWERFUL ONEAPI LIBRARIES

For Data-Centric Functions

- Key domain-specific functions to accelerate compute intensive workloads
- Custom-coded for uncompromised performance on SVMS (Scalar, Vector, Matrix, Spatial) architectures



POWERFUL ONEAPI TOOLS

Productive debugging and performance analysis across architectures

Intel® VTune™ Profiler

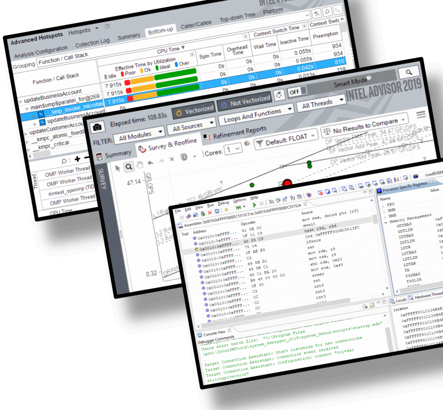
- ▷ Profiler to analyze CPU and accelerator performance of compute, threading, memory, storage, and more

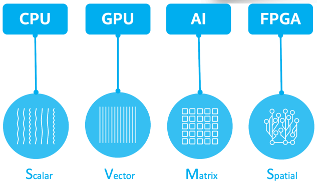
Intel® Advisor

- ▷ Design assistant to provide advice on threading, and vectorization

Intel®-enhanced gdb

- ▷ Application debugger for fast code debug on CPUs and accelerators

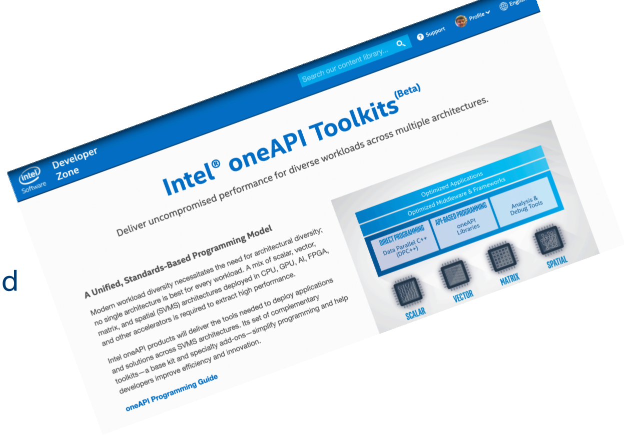




ONEAPI TOOLKITS

One core toolkit

- ▷ Additional toolkits targeting specific data-centric workloads
- ▷ Each includes oneAPI components and complementary oneAPI ecosystem components
- ▷ Ready-to-go containers and custom installer for easy startup



Intel® oneAPI Toolkits (Beta)

Deliver uncompromised performance for diverse workloads across multiple architectures.

A Unified, Standards-Based Programming Model

Modern workload diversity necessitates the need for architectural diversity; no single architecture is best for every workload. A mix of scalar, vector, matrix, and spatial (SVMS) architectures deployed in CPU, GPU, AI, FPGA, and other accelerators is required to extract high performance.

Intel oneAPI products will deliver the tools needed to deploy applications and solutions across SVMS architectures. Its set of complementary toolkits—a base kit and specialty add-ons—simplify programming and help developers improve efficiency and innovation.

oneAPI Programming Guide

https://software.intel.com/en-us/oneapi

(one stop website for all things oneAPI - software.intel.com/oneapi)

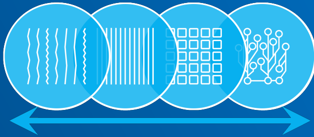
CROSS-ARCHITECTURE SYSTEMS TODAY, ONEAPI TODAY

The future of computing is here, and it is a diverse mix of scalar, vector, matrix, and spatial architectures deployed in CPU, GPU, AI, FPGA, and other accelerators.

- oneAPI unifies and simplifies programming of CPUs and accelerators, delivering developer productivity, and full native language performance
- oneAPI is based on industry standards and open specifications to encourage ecosystem collaboration and innovation

https://software.intel.com/en-us/oneapi

§7. WHAT IS DATA PARALLEL C++?



1. Programming in a Heterogeneous World
2. How oneAPI addresses our Heterogeneous World
3. Hello Doubler - simple DPC++ coding example
4. What is SYCL?
5. DevCloud - Try oneAPI easily
6. oneAPI - Why and How
7. **What is Data Parallel C++?**

WHAT IS DPC++?

DPC++ implements cross-platform data parallelism support (extends C++).

- adheres to the SYCL specification
- implements cross-platform abstraction layer for data parallelism
- open source implementation (github) with all features supported
- utilizes Clang and LLVM
- product implementation, support, and tools available from Intel
- DPC++ book in progress - first four chapters available (free)

TERMS THAT WILL BE THROWN AROUND

▷ Single Source	programmers use
▷ Fat Binaries	implementations use
▷ Directed Programming	programmers use

TERMS THAT WILL BE THROWN AROUND

▹ Single Source	programmers use
▹ Fat Binaries	implementations use
▹ Directed Programming	programmers use

TERMS THAT WILL BE THROWN AROUND

▹ Single Source	programmers use
▹ Fat Binaries	implementations use
▹ Directed Programming	programmers use

PROGRAMMING IN DPC++

DPC++ implements cross-platform data parallelism support (extends C++).

- ▹ Write `kernels'
- ▹ Control when/where/how they might be accelerated

DPC++

The same programming language can support all SVMS architectures.

Data Parallel C++
provides the features and abstraction necessary to deliver uncompromised performance on SVMS architectures.

ONEAPI TRAINING SERIES

- Module 1: Getting Started with oneAPI
- Module 2: Introduction to DPC++
- Module 3: Fundamentals of DPC++, part 1 of 2
- Module 4: Fundamentals of DPC++, part 2 of 2
- Modules 5+: Deeper dives into specific DPC++ features, oneAPI libraries and tools

https://oneapi.com

https://software.intel.com/en-us/oneapi

https://tinyurl.com/book-dpcpp

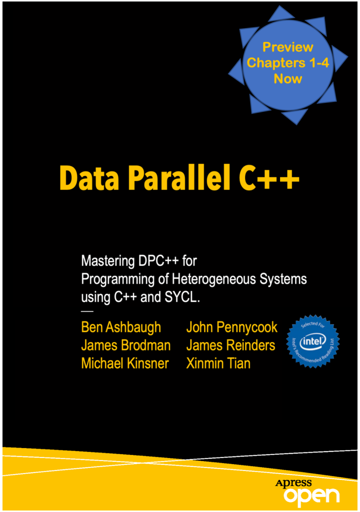
http://tinyurl.com/oneapimodule?1

RESOURCES

- Book (Chapters 1-4 Preview)
- oneAPI Toolkit(s)
- Training, Support, Forums, Example Code

All available
Free

https://software.intel.com/en-us/oneapi



Preview Chapters 1-4 Now

Data Parallel C++

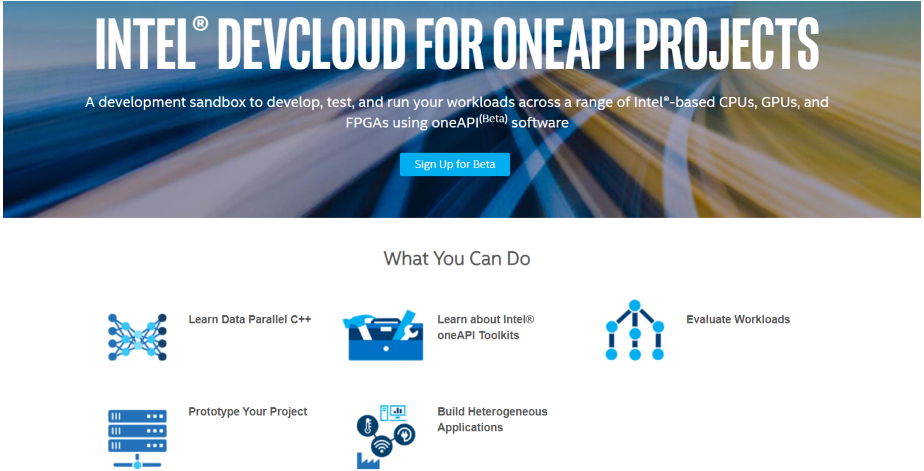
Mastering DPC++ for
Programming of Heterogeneous Systems
using C++ and SYCL.

Ben Ashbaugh
James Brodman
Michael Kinsner
John Pennycook
James Reinders
Xinmin Tian

Apress open

https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?1

DEVCLOUD



INTEL® DEVCLOUD FOR ONEAPI PROJECTS

A development sandbox to develop, test, and run your workloads across a range of Intel®-based CPUs, GPUs, and FPGAs using oneAPI$^{(\text{Beta})}$ software

What You Can Do

- Learn Data Parallel C++
- Learn about Intel® oneAPI Toolkits
- Evaluate Workloads
- Prototype Your Project
- Build Heterogeneous Applications

https://software.intel.com/en-us/devcloud/oneapi

ONEAPI TRAINING SERIES

- Module 1: Getting Started with oneAPI
- Module 2: Introduction to DPC++
- Module 3: Fundamentals of DPC++, part 1 of 2
- Module 4: Fundamentals of DPC++, part 2 of 2
- Modules 5+: Deeper dives into specific DPC++ features, oneAPI libraries and tools

https://oneapi.com

https://software.intel.com/en-us/oneapi

https://tinyurl.com/book-dpcpp

http://tinyurl.com/oneapimodule?1