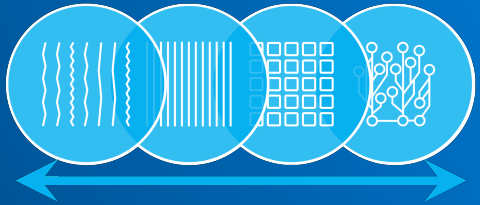
ONEAPI

SINGLE PROGRAMMING MODEL TO DELIVER CROSS-ARCHITECTURE PERFORMANCE



MODULE 2
INTRODUCTION TO DPC++

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?2

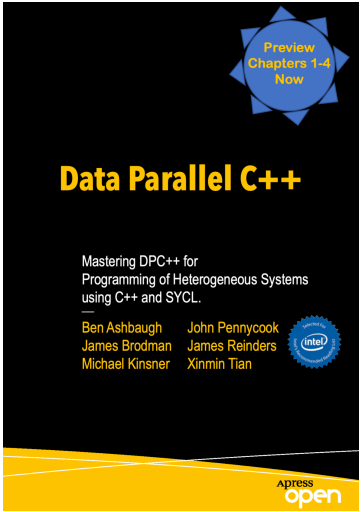
1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

RESOURCES

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

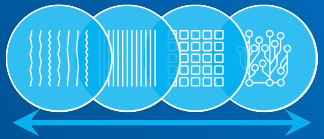
All available
Free

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



https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?2

§1. DPC++, SYCL, C++, AND A HETEROGENEOUS UNIVERSE



1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

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)

WHAT IS SYCL?

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

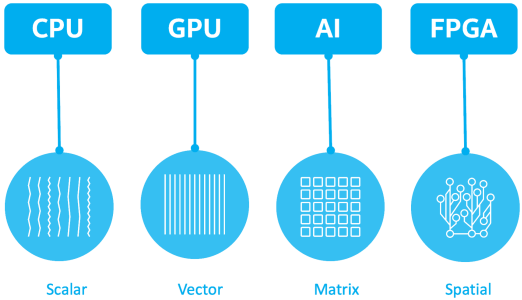
- pronounced `sickle'  `sick ell' /ˈsik(ə)l/
- cross-platform abstraction layer for data parallelism
- single source programming
- extends modern C++
- defined by a Khronos standards group
- Intel is a participant in the standards group, as are many more
- Most of DPC++ is already part of SYCL
- Intel's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

SYCL AND C++

- SYCL is based on purely modern C++.
- SYCL should feel familiar to C++11 programmers.
- SYCL will run ahead of C++ standardization for support of heterogeneous platforms and parallelism.
- ISO C++ of the future, may look a lot like SYCL.

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.



The same programming language can support all SVMS architectures.

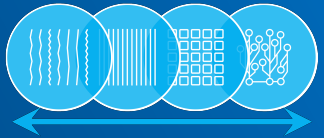
Data Parallel C++
provides the features and abstraction necessary to deliver uncompromised performance on SVMS architectures.

PROGRAMMING IN DPC++

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

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

§2. ANATOMY OF A DPC++ PROGRAM

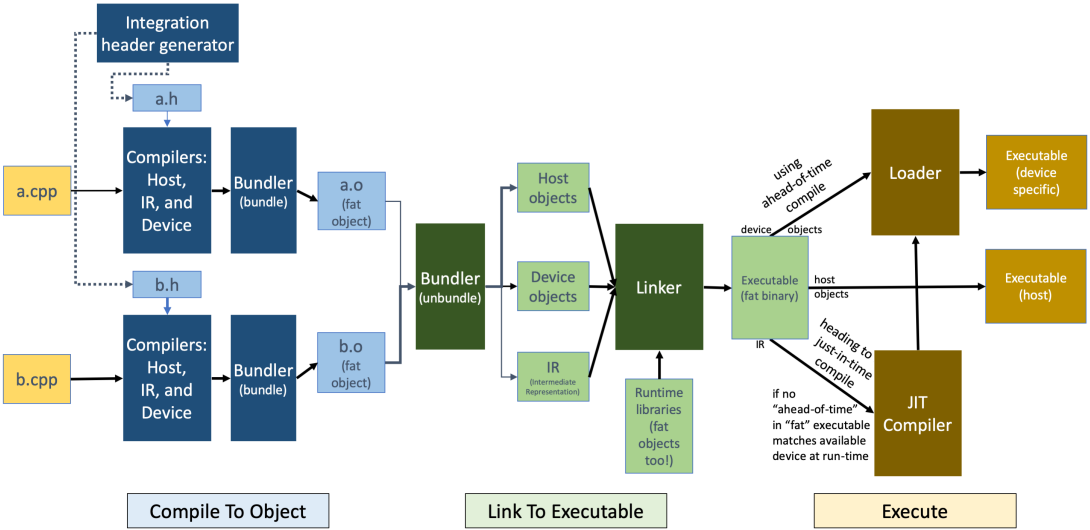


1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

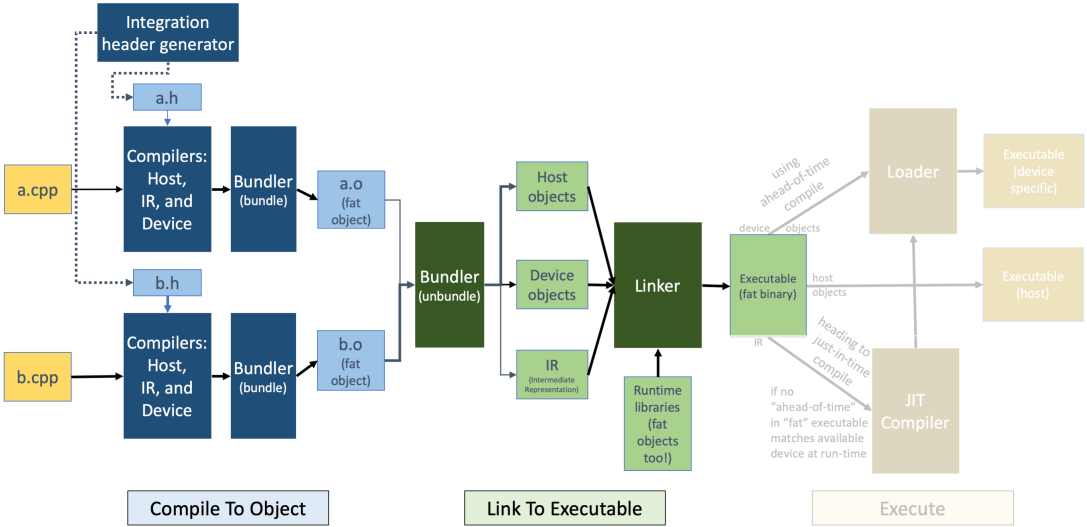
SOFTWARE PROGRAMMING MODEL

- Platform model: what to program (host and devices)
- Execution model: how to control (command queues)
- Kernel model: how to program (kernels)
- Memory model: how to feed with data (memory model)

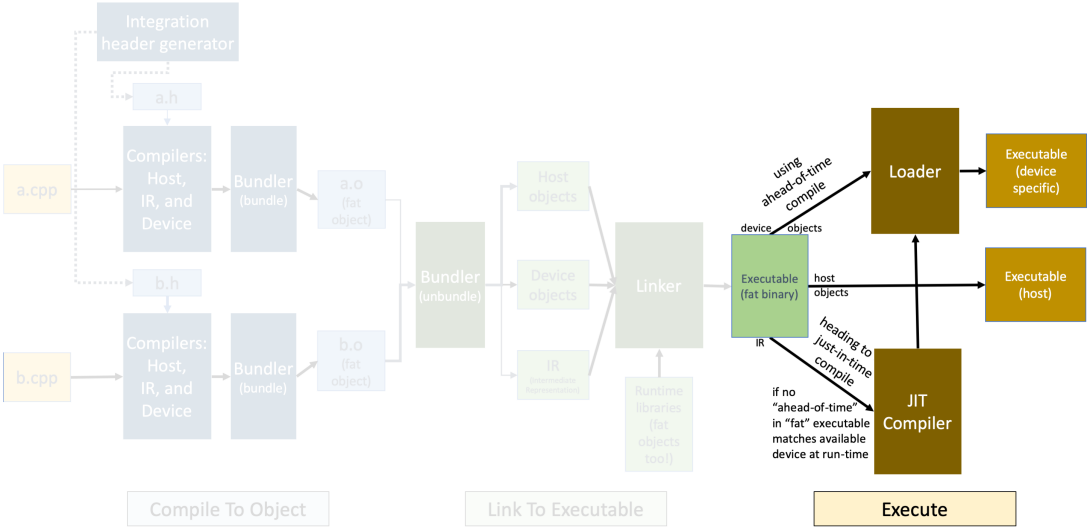
COMPILATION AND EXECUTION



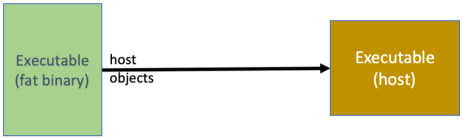
COMPILATION



EXECUTION

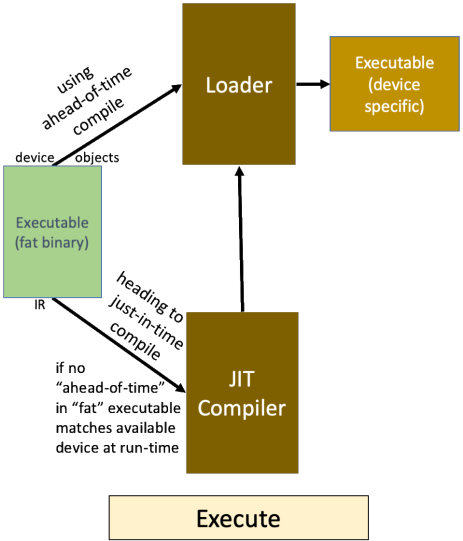

Integration header generator
a.h
a.cpp
Compilers: Host, IR, and Device
Bundler (bundle)
a.o (fat object)
b.h
b.cpp
Compilers: Host, IR, and Device
Bundler (bundle)
b.o (fat object)
Bundler (unbundle)
Host objects
Device objects
IR (Intermediate Representation)
Linker
Runtime libraries (fat objects tool)
Executable (fat binary)
device objects
IR
using ahead-of-time compile
heading to just-in-time compile
if no "ahead-of-time" in "fat" executable matches available device at run-time
Loader
JIT Compiler
Executable (device specific)
host objects
Executable (host)
Compile To Object
Link To Executable
Execute

EXECUTION - HOST

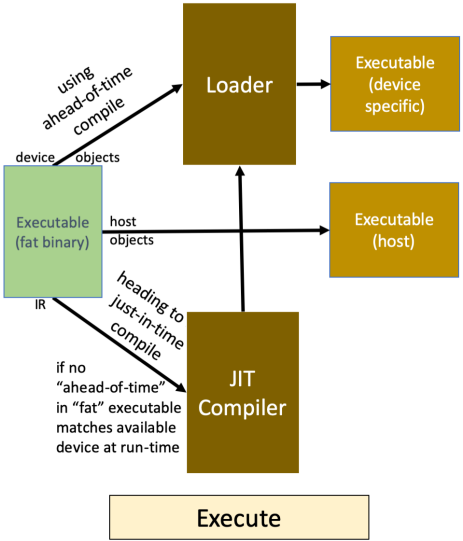


Execute

EXECUTION - DEVICE(S)



EXECUTION


using ahead-of-time compile
Loader
Executable (device specific)
device objects
Executable (fat binary)
host objects
Executable (host)
IR
heading to just-in-time compile
if no "ahead-of-time" in "fat" executable matches available device at run-time
JIT Compiler
Execute

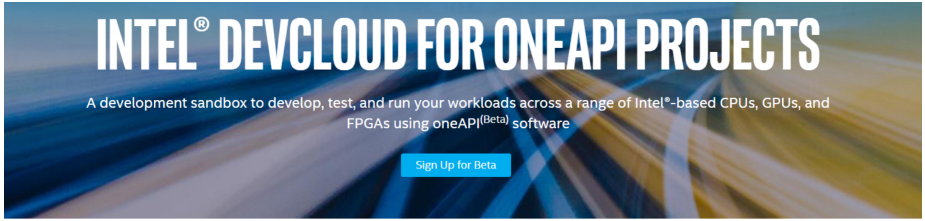
PLATFORM TERMINOLOGY

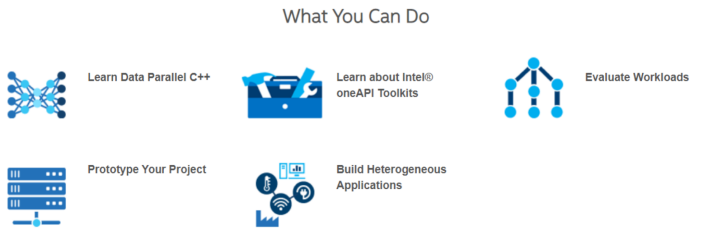
- Platform = Host + Devices
- Host makes API calls (queue work, define memory)
- Devices run Kernels
- *host device* always available

PLATFORM INFORMATION

There are three important sets of capabilities to consider for a SYCL device:
the platform version,
the version of a device, and
the extensions supported.

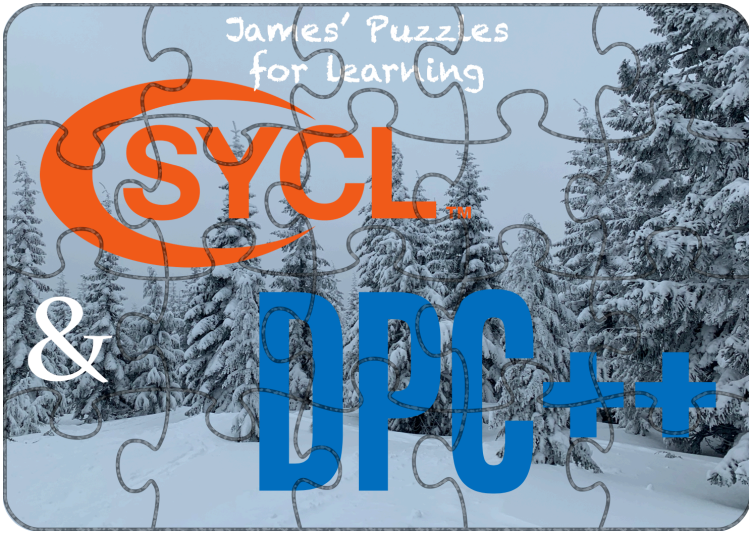
DEVCLOUD





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

PUZZLES



code for this module: http://tinyurl.com/oneapimodule?2

JUMP ON DEVCLOUD

```
$ ssh devcloud
...login to the devcloud...

$ wget tinyurl.com/oneapimodule?2 -O 2.tz
$ tar xvfz 2.tz
...fetch and unpack code I'll be playing with for module 2...

$ pbsnodes -l free
...list of free nodes...

$ pbsnodes s001-145
...information about node s001-145...

$ pbsnodes | more
...lots more detail...
$ pbsnodes | grep properties
...useful properties list...
$ pbsnodes | grep fpga
...useful fpga oriented list...
```

HELLO QSUB

```
$ mkdir mytst
$ cd mytst

$ cat - > myhello.sh
echo "HELLO, WORLD!"
^D

$ qsub myhello.sh

$ qstat
Job ID Name       ...
------ ----------
3463   myhello.sh ...

$ qstat
```

HELLO QSUB

```
$ ls -l
total 8
-rw-r--r-- 1 u27938 u27938  21 Oct 14 22:58 myhello.sh
-rw------- 1 u27938 u27938   0 Oct 14 22:58 myhello.sh.e3463
-rw------- 1 u27938 u27938 603 Oct 14 22:58 myhello.sh.o3463

$ cat myhello.sh.o3463

################################################################################
#      Date:           Mon Oct 14 22:58:58 PDT 2019
#    Job ID:           3463.v-qsvr-nda.aidevcloud
#      User:           u27938
# Resources:           neednodes=1:ppn=2,nodes=1:ppn=2,walltime=06:00:00
################################################################################

HELLO, WORLD!

################################################################################
# End of output for job 3463.v-qsvr-nda.aidevcloud
# Date: Mon Oct 14 22:58:59 PDT 2019
```

HELLO QSUB

```
...use a particular node...
$ qsub -lnodes=s001-n155:ppn=2

...use a node based on a property...
$ qsub -lnodes=1:ppn=2:fpga_compile
$ qsub -lnodes=1:ppn=2:gpu
$ qsub -lnodes=1:ppn=2:skl
$ qsub -lnodes=1:ppn=2:cfl
```

Command docs: https://tinyurl.com/pbsnodes

CURIOUS2.CPP - SOURCE CODE

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

using namespace cl::sycl;

int main() {
  unsigned number = 0;
  auto MyPlatforms =
            platform::get_platforms();

  /* Loop through the platforms
     SYCL can find
     there is always ONE */
  for (auto &OnePlatform : MyPlatforms) {
    std::cout
     << ++number << " found..."
     << std::endl
     << "Platform: "
     << OnePlatform.get_info<info::platform::name>()
     << std::endl;

    /* Loop through the devices SYCL can find
       there is always ONE */
    auto MyDevices = OnePlatform.get_devices();
    for (auto &OneDevice : MyDevices ) {
      std::cout
       << " Device: "
       << OneDevice.get_info<info::device::name>()
       << std::endl;
    }
    std::cout << std::endl;
  }
}
```

CURIOUS2.CPP - DOWNLOAD

```
...if you did not already fetch and unpack code for module 2...
$ wget tinyurl.com/oneapimodule?2 -O 2.tz
$ tar xvfz 2.tz

$ cd module02

$ cat mycode.sh
#!/bin/sh
if [ -f /opt/intel/inteloneapi/setvars.sh ]; then
   source /opt/intel/inteloneapi/setvars.sh 2>/dev/null 1>/dev/null
fi
cd module02
make clean
make curious2
./curious2
```

CURIOUS2.CPP - RUN

```
$ qsub mycode.sh
3592...

$ qstat 3592
Job ID   Name
-------- ---------
3592...  mycode.sh

$ qstat 3592
qstat: Unknown Job Id Error 3592...

$ ls -l *3592*
-rw------- 1 u27938 u27938    0 Oct 15 13:40 mycode.sh.e3592
-rw------- 1 u27938 u27938 1072 Oct 15 13:40 mycode.sh.o3592
```

CURIOUS2.CPP - RESULTS 1 OF 2

```
$ cat mycode.sh.o3592
#################################################################################
#      Date:           Tue Oct 15 13:40:17 PDT 2019
#    Job ID:           3592.v-qsvr-nda.aidevcloud
#      User:           u27938
# Resources:           neednodes=1:ppn=2,nodes=1:ppn=2,walltime=06:00:00
#################################################################################

rm -f curious curious2 verycurious doubler doubler2 doubler3 *~ *.o a.out
dpcpp     curious2.cpp   -o curious2
1 found...
Platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
 Device: Intel(R) FPGA Emulation Device

2 found...
Platform: Intel(R) FPGA SDK for OpenCL(TM)
 Device: pac_a10 : Intel PAC Platform (pac_ee00000)

3 found...
Platform: Intel(R) OpenCL
```

CURIOUS2.CPP - RESULTS 2 OF 2

```
3 found...
Platform: Intel(R) OpenCL
 Device: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz

4 found...
Platform: SYCL host platform
 Device: SYCL host device


########################################################################
# End of output for job 3592.v-qsvr-nda.aidevcloud
# Date: Tue Oct 15 13:40:26 PDT 2019
```

VERYCURIOUS.CPP

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

int main() {
  unsigned number = 0;
  auto MyPlatforms = cl::sycl::platform::get_platforms();

  /* Loop through the platforms SYCL can find
     there is always ONE */
  for (auto &OnePlatform : MyPlatforms) {
    std::cout << ++number << " found..."
              << std::endl
              << "Platform: "
              << OnePlatform.get_info<cl::sycl::info::platform::name>()
              << std::endl;

    /* Loop through the devices SYCL can find
       there is always ONE */
    auto MyDevices = OnePlatform.get_devices();
    for (auto &OneDevice : MyDevices ) {
      std::cout << " Device: "
```

VERYCURIOUS - PLATFORM 1 OF 4

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

$ ./verycurious
1 found...
Platform:
     cl::sycl::info::platform::profile is 'EMBEDDED_PROFILE'
     cl::sycl::info::platform::version is 'OpenCL 1.0 Intel(R) FPGA SDK for OpenCL(TM), Version 19.2'
     cl::sycl::info::platform::name is 'Intel(R) FPGA Emulation Platform for OpenCL(TM)'
     cl::sycl::info::platform::vendor is 'Intel(R) Corporation'
     cl::sycl::info::platform::extensions is :
       1) cl_khr_icd
       2) cl_khr_byte_addressable_store
       3) cl_intel_fpga_host_pipe
       4) cles_khr_int64
       5) cl_khr_il_program
 Device: Intel(R) FPGA Emulation Device
     is_host() = No
     is_cpu() = No
     is_gpu() = No
```

VERYCURIOUS - PLATFORM 2 OF 4

```
2 found...
Platform:
     cl::sycl::info::platform::profile is 'FULL_PROFILE'
     cl::sycl::info::platform::version is 'OpenCL 2.1 '
     cl::sycl::info::platform::name is 'Intel(R) OpenCL HD Graphics'
     cl::sycl::info::platform::vendor is 'Intel(R) Corporation'
     cl::sycl::info::platform::extensions is :
       1) cl_khr_3d_image_writes
       2) cl_khr_byte_addressable_store
       3) cl_khr_fp16
       4) cl_khr_depth_images
       5) cl_khr_global_int32_base_atomics
     ...
       37) cl_intel_advanced_motion_estimation
       38) cl_intel_va_api_media_sharing
 Device: Intel(R) Gen9 HD Graphics NEO
     is_host() = No
     is_cpu() = No
     is_gpu() = Yes
     is_accelerator() = No
```

VERYCURIOUS - PLATFORM 3 OF 4

```
3 found...
Platform:
     cl::sycl::info::platform::profile is 'FULL_PROFILE'
     cl::sycl::info::platform::version is 'OpenCL 2.1 LINUX'
     cl::sycl::info::platform::name is 'Intel(R) OpenCL'
     cl::sycl::info::platform::vendor is 'Intel(R) Corporation'
     cl::sycl::info::platform::extensions is :
       1) cl_khr_icd
       2) cl_khr_global_int32_base_atomics
       3) cl_khr_global_int32_extended_atomics
       4) cl_khr_local_int32_base_atomics
       5) cl_khr_local_int32_extended_atomics
     ...
       16) cl_khr_fp64
       17) cl_khr_image2d_from_buffer
 Device: Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz
     is_host() = No
     is_cpu() = Yes
     is_gpu() = No
     is_accelerator() = No
```

VERYCURIOUS - PLATFORM 4 OF 4

```
4 found...
Platform:
     cl::sycl::info::platform::profile is 'FULL PROFILE'
     cl::sycl::info::platform::version is '1.2'
     cl::sycl::info::platform::name is 'SYCL host platform'
     cl::sycl::info::platform::vendor is ''
     cl::sycl::info::platform::extensions is :
     NO extensions
 Device: SYCL host device
     is_host() = Yes
     is_cpu() = No
     is_gpu() = No
     is_accelerator() = No
     cl::sycl::info::device::vendor_id is '32902'
     cl::sycl::info::device::max_compute_units is '12'
     cl::sycl::info::device::max_work_item_dimensions is '3'
     ...
     cl::sycl::info::device::name is 'SYCL host device'
     cl::sycl::info::device::vendor is ''
     cl::sycl::info::device::driver_version is '1.2'
```

COMMAND GROUPS AND QUEUES

- Host code creates Queue(s) to command devices
- Multiple Queues allowed per Device
- but - a Queue cannot command multiple devices

- Command Groups are submitted into Queues

COMMAND GROUPS AND QUEUES

- Host code creates Queue(s) to command devices
- Multiple Queues allowed per Device
- but - a Queue cannot command multiple devices

- Command Groups are submitted into Queues

RESTRICTIONS ON KERNEL CODE

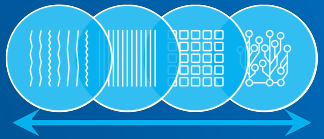
Supported include:

- lambdas
- operator overloading
- templates
- classes
- static polymorphism
- share data with host via accessors
- read-only values of host variables subject via lambda captures

Not supported:

- dynamic polymorphism
- dynamic memory allocations
- static variables
- function pointers
- pointer structure members
- runtime type information
- exception handling

§3. REVISIT DOUBLER - DPC++



1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

"HELLO DOUBLER" DPC++

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

using namespace cl::sycl;

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

```
  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]);
}
```

SCOPES

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;    /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18 }
19 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- *command scope lines 8-17*
- kernel scope lines 11-16
- application scope all else (default))

SCOPES

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;    /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18 }
19 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- *command scope lines 8-17*
- kernel scope lines 11-16
- application scope all else (default))

SCOPES

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;    /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18 }
19 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- command scope lines 8-17
- *kernel scope lines 11-16*
- application scope all else (default))

SCOPES

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;     /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18 }
19 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- command scope lines 8-17
- kernel scope lines 11-16
- *application scope all else (default))*

WHY THE { } BLOCK?

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;    /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18 }
19 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- Putting all the SYCL work inside the { } block ensures all SYCL work has concluded.
- Destruction of bufferA is a barrier (causes a wait)

USE ACCESSORS

```
1  printf("Value at start: myArray[42] is %d.\n",myArray[42]);
2  {
3    queue myQ;    /* use defaults today */
4    /* (queue parameters possible - future topic) */
5    range<1> mySize{SIZE};
6    buffer<int, 1> bufferA(myArray.data(), mySize);
7
8    myQ.submit([&](handler &myHandle) {
9      auto deviceAccessorA =
10       bufferA.get_access<access::mode::read_write>(myHandle);
11     myHandle.parallel_for<class uniqueID>(mySize,
12       [=](id<1> index)
13         {
14           deviceAccessorA[index] *= 2;
15         }
16     );
17   });
18   printf("Premature peek: myArray[42] is %d.\n",myArray[42]);
19 }
20 printf("Value at finish: myArray[42] is %d.\n",myArray[42]);
```

- When I ran it - it printed 42, 42, and 84.
- The *Premature* line is not defined.
- Should use an accessor!

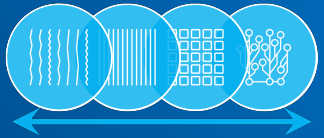
- Doubler, like other DPC++ kernels, can be mapped to all architectures.
- The suitability of each architecture is algorithm dependent.

PROGRAMMING IN DPC++

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

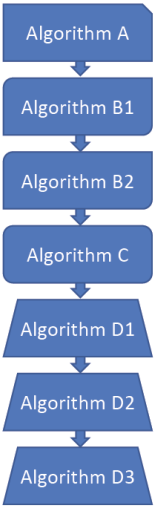
- Write `kernels'
- Control when/where/how they might be accelerated

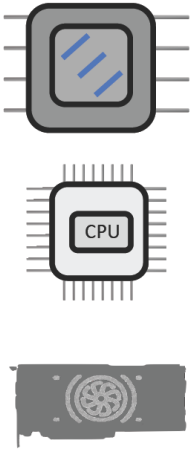
§4. DEVICE SELECTION IN DPC++



1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

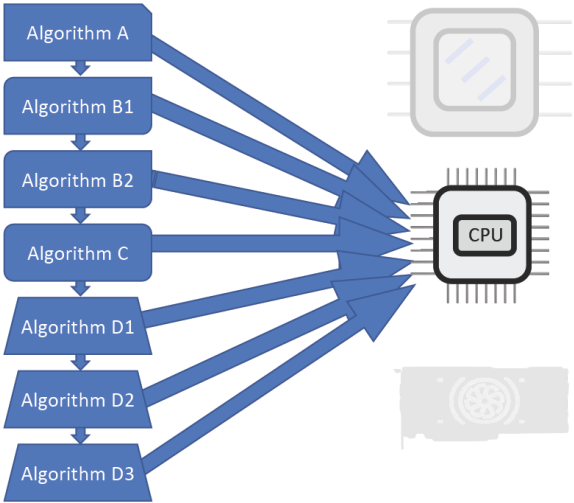
DPC++ DEVICE SELECTION





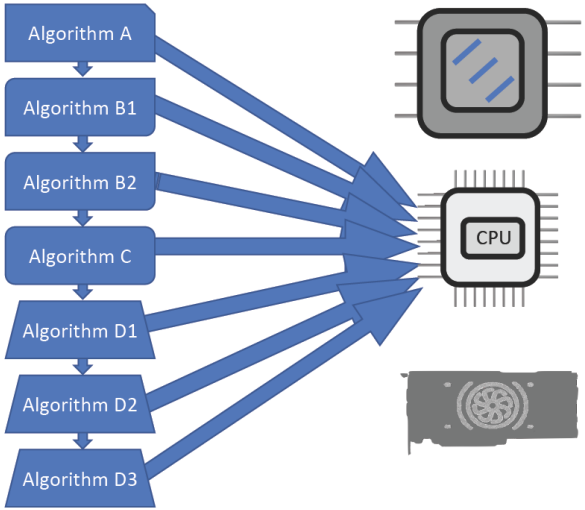
- Algorithms
- Devices

DPC++ DEVICE SELECTION



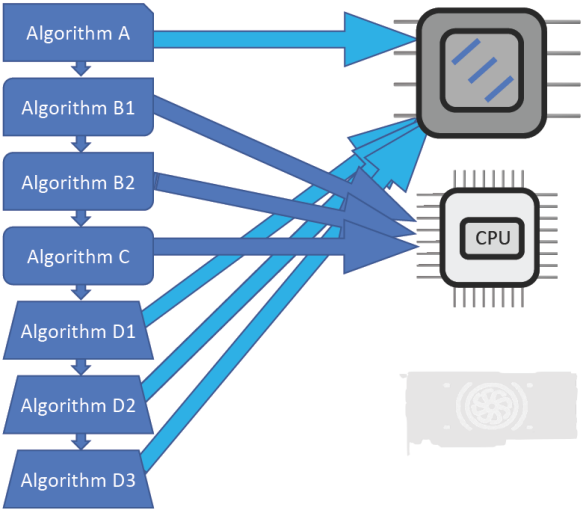
- In a system without accelerators, we use the CPU.
- The grayed out devices are not in this system!

DPC++ DEVICE SELECTION



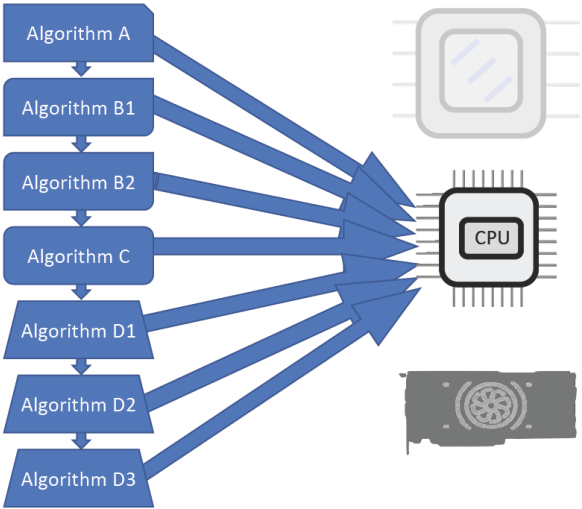
- Even with accelertors present, all algorithms can go to the CPU.
- In module 2 - we'll discuss the three ways this can happen: Default, Host, or CPU.

DPC++ DEVICE SELECTION



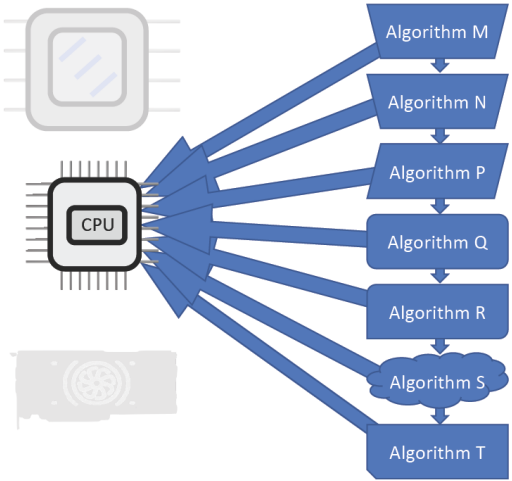
- We can control devices to use.
- Some algorithms use the top device.

DPC++ DEVICE SELECTION



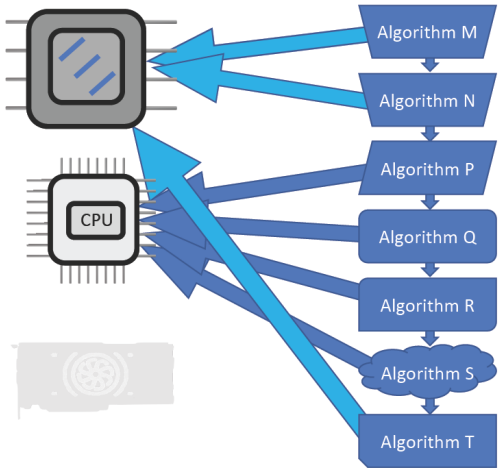
- In a system with an accelerator, but not the one we wanted.
- Let's use the CPU.

DPC++ DEVICE SELECTION



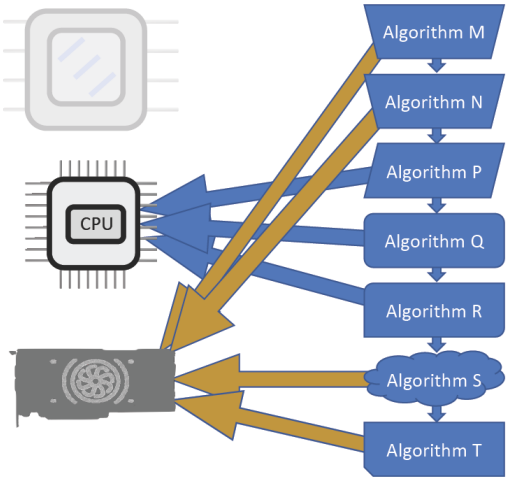
- Consider a different program.
- This one has algorithms we have figured out how to map to multiple devices.

DPC++ DEVICE SELECTION



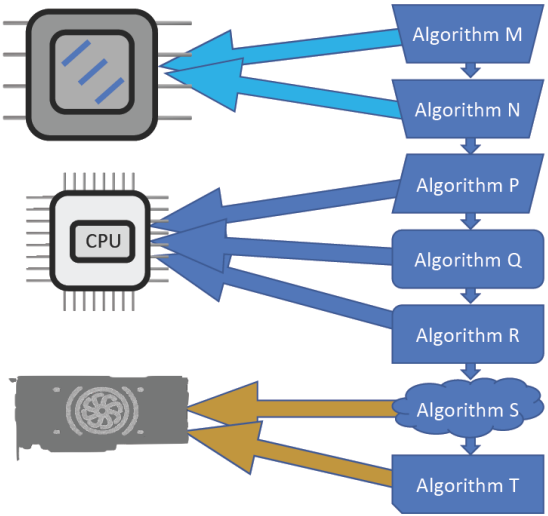
- The top device can be used.
- Algorithms M, N, and T have code tailored for it.

DPC++ DEVICE SELECTION



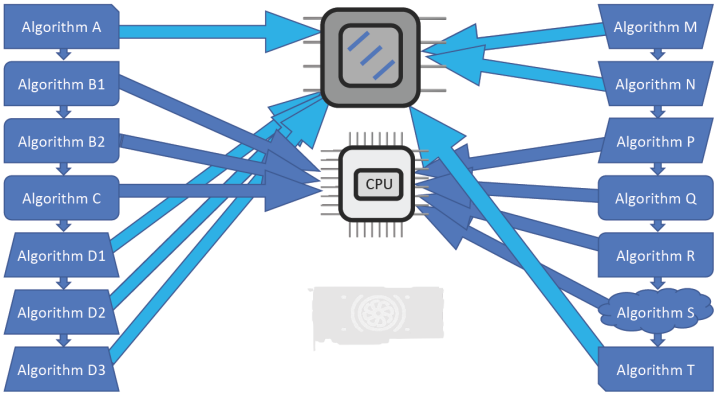
- The bottom device can be used as well.
- Same algorithms (M,N,T) plus Algorithm S.

DPC++ DEVICE SELECTION



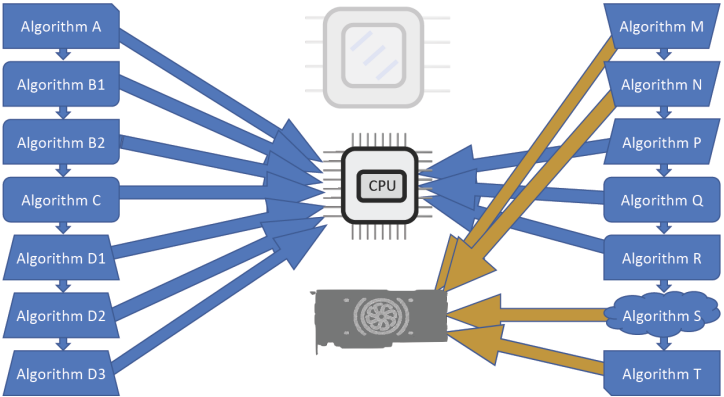
- A single system might have both accelerators available.
- We can choose the mapping we want - based on best match, application balance, data movement, etc.

DPC++ DEVICE SELECTION



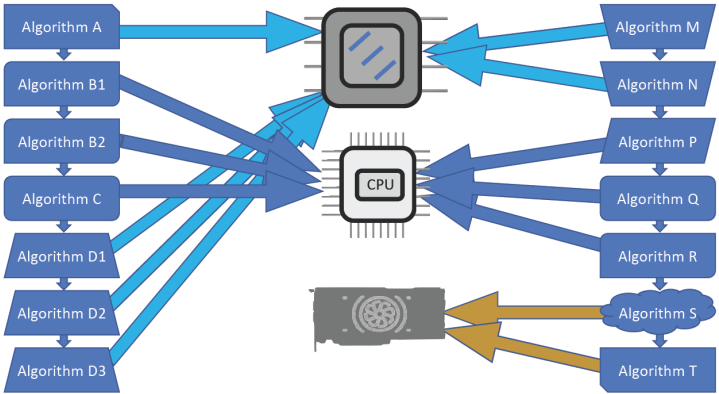
- In a system with the top device.
- Both programs can be accelerated.

DPC++ DEVICE SELECTION



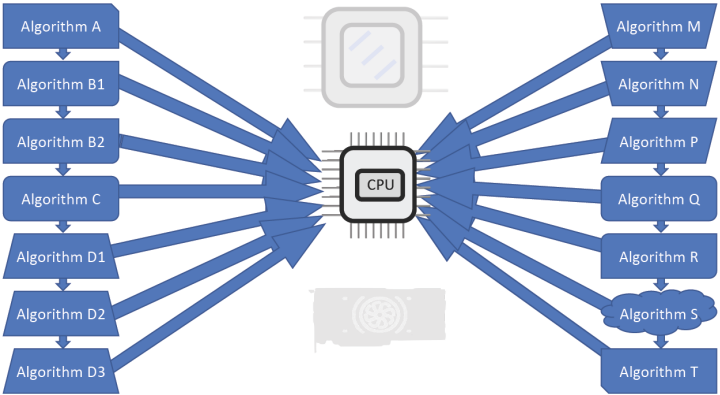
- In a system with the bottom device.
- Only our second example had uses for it.

DPC++ DEVICE SELECTION



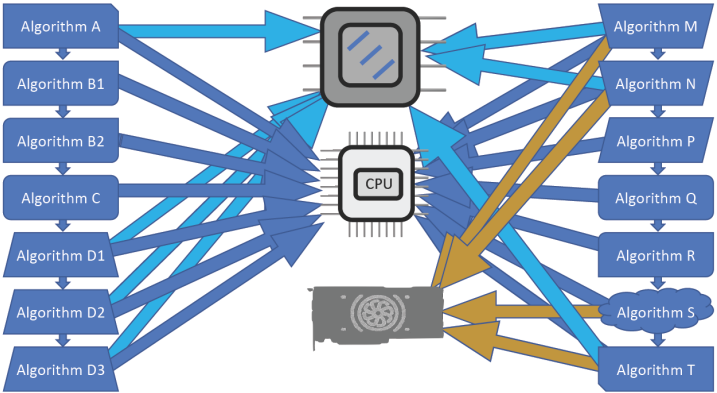
- In system with BOTH devices.
- Both programs have options.

DPC++ DEVICE SELECTION



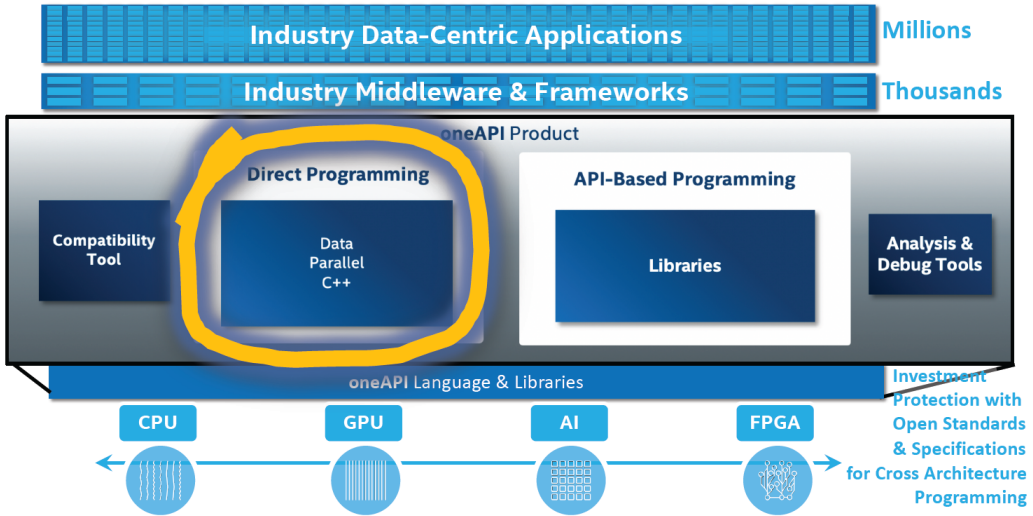
- In a system with NO accelerators.
- Both programs use the CPU only.

DPC++ DEVICE SELECTION

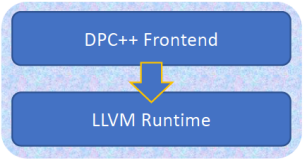


- Reconsidering a system with both accelerators.
- Choices for the second program can include both. We have full control in our program!

ONEAPI FOR CROSS-ARCHITECTURE PERFORMANCE



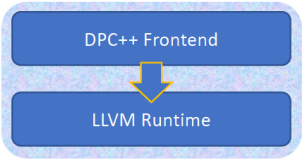
DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets…
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

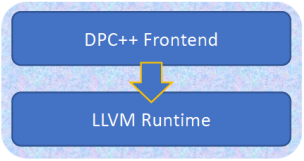
DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets...
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

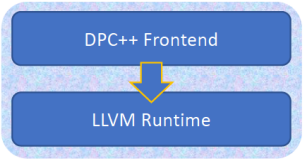
DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets…
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

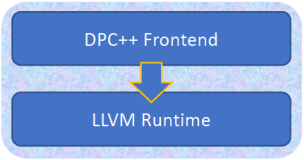
DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets…
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

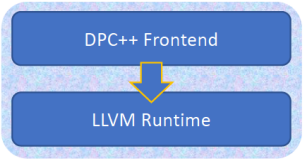
DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets…
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

DATA PARALLEL C++ (DPC++)



Standards-Based Cross-Architecture Language

- Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators
  - Allows code reuse across hardware targets…
  - while permitting custom tuning for a specific accelerator
  - Open, cross-industry alternative to single architecture proprietary language
- Based on C++
  - Delivers C++ productivity benefits, using common and familiar C and C++ constructs
  - Incorporates SYCL (from the Khronos Group) to support data parallelism and heterogeneous programming
- Language enhancements being driven through community project
  - Extensions to simplify data parallel programming
  - Open and cooperative development for continued evolution
- Builds upon many years of experience in architecture and compilers

WHY NOT AN EXISTING LANGUAGE?

- Portable languages are either serial (C++, Fortran, Java, ...) or high-level (MATLAB, Python, ...).
- Data Parallel languages are either proprietary (CUDA) or low-level (OpenCL).

- Intel has developed DPC++ by building on C++, embracing open standard SYCL, and filling in real-world gaps with solutions today.
- The result is open, and available today to enable data parallel programming across SVMS architectures.

WHY NOT AN EXISTING LANGUAGE?

- Portable languages are either serial (C++, Fortran, Java, …) or high-level (MATLAB, Python, …).
- Data Parallel languages are either proprietary (CUDA) or low-level (OpenCL).

- Intel has developed DPC++ by building on C++, embracing open standard SYCL, and filling in real-world gaps with solutions today.
- The result is open, and available today to enable data parallel programming across SVMS architectures.

WHY NOT AN EXISTING LANGUAGE?

- Portable languages are either serial (C++, Fortran, Java, …) or high-level (MATLAB, Python, …).
- Data Parallel languages are either proprietary (CUDA) or low-level (OpenCL).

- Intel has developed DPC++ by building on C++, embracing open standard SYCL, and filling in real-world gaps with solutions today.
- The result is open, and available today to enable data parallel programming across SVMS architectures.

The language is:

C++

+

SYCL

+

Additional Features

The language is:

C++

+

SYCL

+

Additional Features

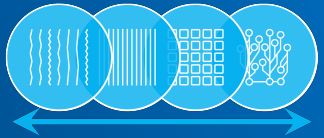
such as...

ndrange subgroups, USM, ordered queue

The implementation is:

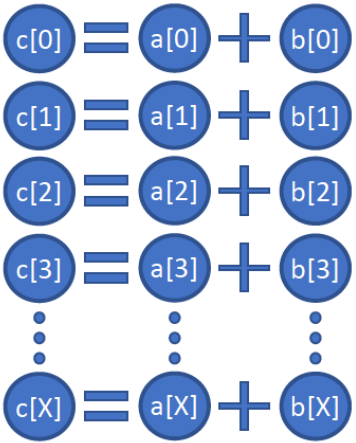
Clang
+
LLVM
+
Runtime

§5. WHAT IS DATA PARALLEL COMPUTING?

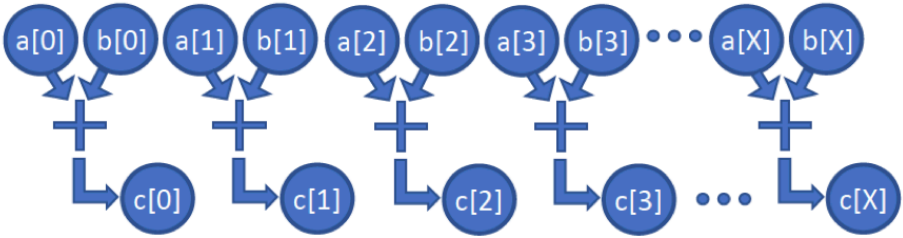


WHAT IS DATA PARALLEL COMPUTING?

Vector Add: c[0..X] = a[0..X] + b[0..X]



WHAT IS DATA PARALLEL COMPUTING?



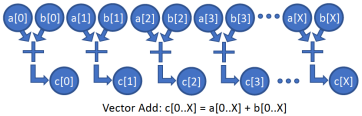
Vector Add: c[0..X] = a[0..X] + b[0..X]

Independent Computations

(preferably, a lot of them!)

HARDWARE FOR DATA PARALLEL COMPUTING

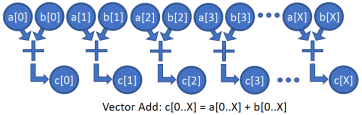
- Pipelining
- SIMD (vector)
- Multithreading
- Multicore
- Combinations of the above



Independent Computations
(preferably, a lot of them!)

HARDWARE FOR DATA PARALLEL COMPUTING

- Pipelining
- SIMD (vector)
- Multithreading
- Multicore
- Combinations of the above

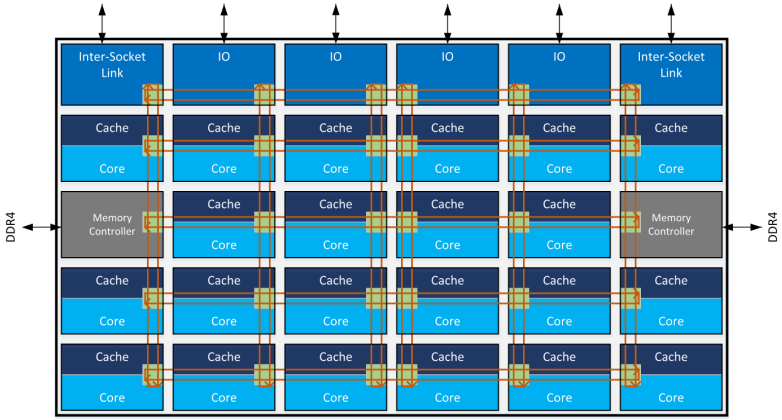


Independent Computations
(preferably, a lot of them!)

Compiler and runtime map the Independent Computations
onto various data parallel hardware(s).

INTEL® XEON® SCALABLE PROCESSOR

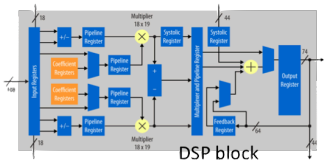
- Pipelining (1993
- SIMD (1996)
- Multithreading (2002)
- Multicore (2005)
- Combinations of the above



- Pipelining
- SIMD (vector)
- Multithreading
- Multicore
- Combinations of the above

FPGAS

- Compiler sythesizes custom SIMD pipeline, and replicates
- Effectively Pipelining + SIMD + Multi*



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.

DPC++

- DPC++ can be completely built from open source (all features)
- github project, community collaboration
- Full documentation of all Intel® extensions
- C++, and SYCL, of course are open and well defined as well
- Goal of DPC++ open source project is to
  merge with top trunk of LLVM and Clang
- Intel has product version of DPC++ as well (freely available)

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?2

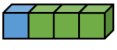
DPC++ - READY TO DIVE IN

- Modules 3 and 4 dive into DPC++
- We have covered the ``why" and ``what" for oneAPI and DPC++

Work-item

DPC++ - READY TO DIVE IN

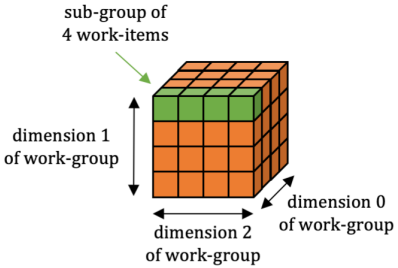
- Modules 3 and 4 dive into DPC++
- We have covered the ``why" and ``what" for oneAPI and DPC++



Sub-group

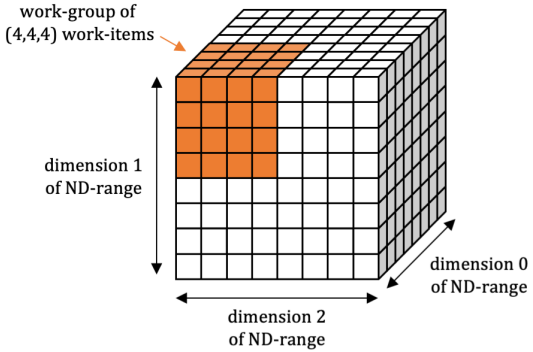
DPC++ - READY TO DIVE IN

- Modules 3 and 4 dive into DPC++
- We have covered the ``why" and ``what" for oneAPI and DPC++



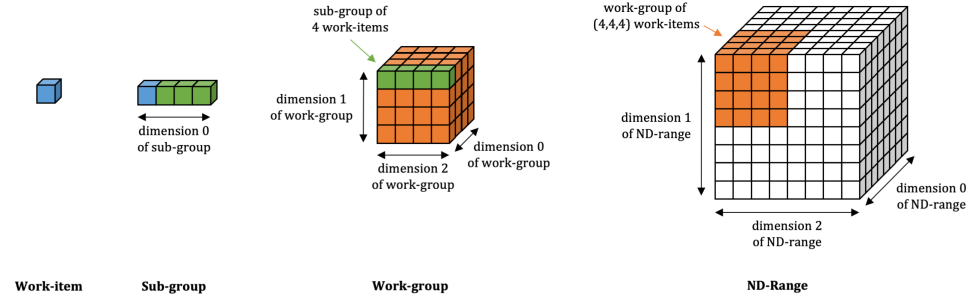
DPC++ - READY TO DIVE IN

- Modules 3 and 4 dive into DPC++
- We have covered the ``why" and ``what" for oneAPI and DPC++



DPC++ - READY TO DIVE IN

- Modules 3 and 4 dive into DPC++
- We have covered the ``why" and ``what" for oneAPI and DPC++

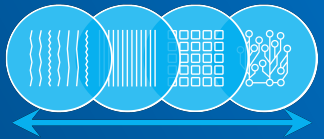


ONEAPI TRAINING SERIES

- 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

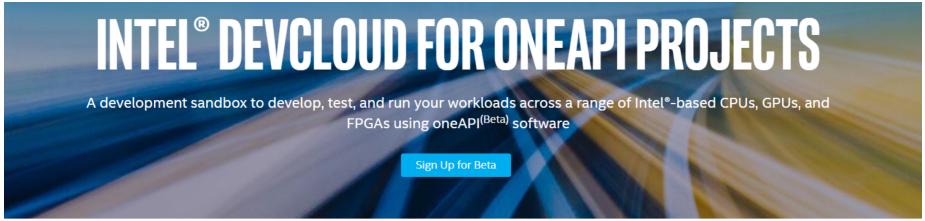
https://oneapi.com
https://software.intel.com/en-us/oneapi
https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?2

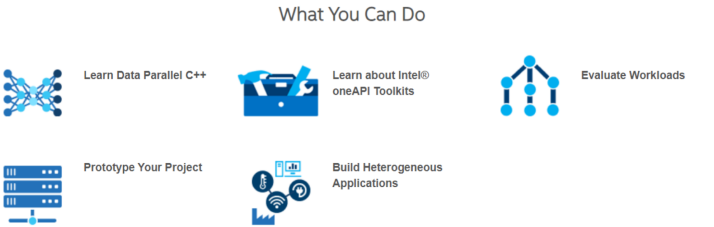
§6. DEVCLOUD - TRY ONEAPI EASILY



1. DPC++, SYCL, C++, and a Heterogeneous Universe
2. Anatomy of a DPC++ program
3. Revisit Doubler - DPC++
4. Device Selection in DPC++
5. What is Data Parallel Computing?
6. DevCloud - Try oneAPI easily

DEVCLOUD





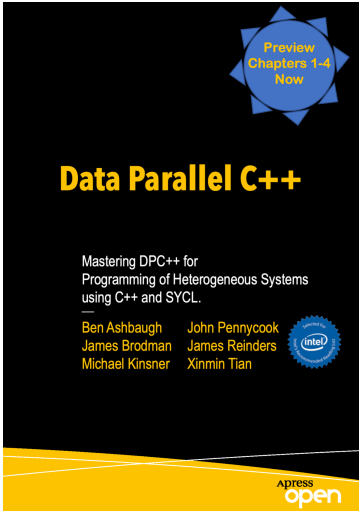
https://software.intel.com/en-us/devcloud/oneapi

RESOURCES

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

All available
Free

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



https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?2

NEXT FOR YOU

- Sign up for DevCloud
- Try the oneAPI toolkit
- Watch module 1 if you skipped it
- Join us for Modules 3+, to continue diving into DPC++

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