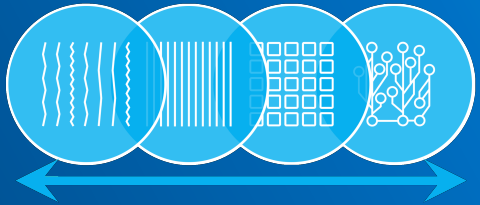
ONEAPI

SINGLE PROGRAMMING MODEL TO DELIVER CROSS-ARCHITECTURE PERFORMANCE



MODULE 3
DPC++ FUNDAMENTALS, PART 1 OF 2

COLFAX RESEARCH

intel

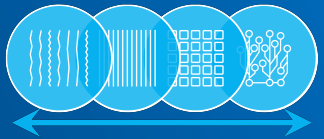
ONEAPI TRAINING SERIES

- 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?3

1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

§1. DPC++ PROGRAMS



1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

PRIOR TRAINING MODULES

Training Module 2 covers What and Why of DPC++
Training Module 1 covers What and Why of oneAPI

WHAT IS DPC++?

Data Parallel C++ is

- Open source project built on
  - C++
  - with SYCL for data parallelism
  - with a few extensions to smooth things out

Note: Extensions are not Intel specific - but Intel implemented, with an eventualy hope of influencing SYCL. This includes *subgroups* (covered in training module 3) and *USM* (covered in module 4).

PARTS OF DPC++ PROGRAM, 1 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {









}
```

- Include the SYCL header file.
  - defines the runtime API
  - To use Intel's FPGA selector, also #include <CL/sycl/intel/fpga_extensions.hpp>
- Most of us will also add a *using* for namespace cl::sycl to make our coding more readable.

PARTS OF DPC++ PROGRAM, 2 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...


  queue myQueue{...};









}
```

- All work requests are done via a queue.
- A queue uniquely attaches to a single device (e.g., GPU, FPGA, AI, CPU, Host).

PARTS OF DPC++ PROGRAM, 3 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...


  queue myQueue{...};

  ...

  myQueue.submit([&](handler &cgh) {

    // accessors (for connecting to memory via buffers)
    // kernel defined here (with lambda -
    // by value captures only)

  });


}
```

- Queue accepts work requests as submissions.
- Highlighted lines are the *command group scope*.
- Submissions finish asynchronously.
- Only one kernel (work described in a lambda) per submit!

PARTS OF DPC++ PROGRAM, 4 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...

  {
    // define buffers!!!
    queue myQueue{...};

    ...

    myQueue.submit([&](handler &cgh) {

      // accessors (for connecting to memory
      // via buffers) choose one of four ways
      // to express parallelism; only one per
      // submit; use by-value lambda capture

    });
  } // destroy buffers - synchronizes us!

}
```

```
cgh.single_task(
  [=]() {
    // kernel function is executed EXACTLY once on a SINGLE work-item
});

cgh.parallel_for(
  range<3>(1024,1024,1024), // using 3D in this example
  [=](id<3> myID) {
    // kernel function is executed on an n-dimensional range (NDrange)
});

cgh.parallel_for(
  nd_range<3>({1024,1024,1024},{16,16,16}), // using 3D in this example
  [=](nd_item<3> myID) {
    // kernel function is executed on an n-dimensional range (NDrange)
});

cgh.parallel_for_work_group(
  range<2>(1024,1024),      // using 2D in this example
  [=](group<2> myGroup) {
    // kernel function is executed once per work-group
});

grp.parallel_for_work_item(
  range<1>(1024),           // using 1D in this example
  [=](h_item<1> myItem) {
    // kernel function is executed once per work-item
});
```

PARTS OF DPC++ PROGRAM, 5 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...

  {
    // define buffers!!!
    queue myQueue{...};

    ...

    myQueue.submit([&](handler &cgh) {

      // accessors (for connecting to memory via buffers)
      // kernel defined here (with lambda -
      // by value captures only)

    });
  } // destroy buffers - synchronizes us!

}
```

- Control the scope of buffers, to control synchronization with host programs.
- This is a convention - follow it!

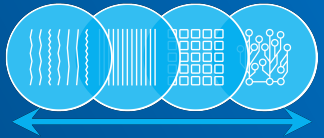
PARTS OF DPC++ PROGRAM, 6 OF 6

We should *always* wrap SYCL code in a try-catch-block!

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...

  try {
    // define buffers!!!
    queue myQueue{...};

    ...

    myQueue.submit([&](handler &cgh) {

      // accessors (for connecting to memory via buffers)
      // kernel defined here (with lambda -
      // by value captures only)

    });

    myQueue.wait_and_throw();
  } catch (...) { /* error handling */ }
    // destruction of buffers synchronizes us too

}
```

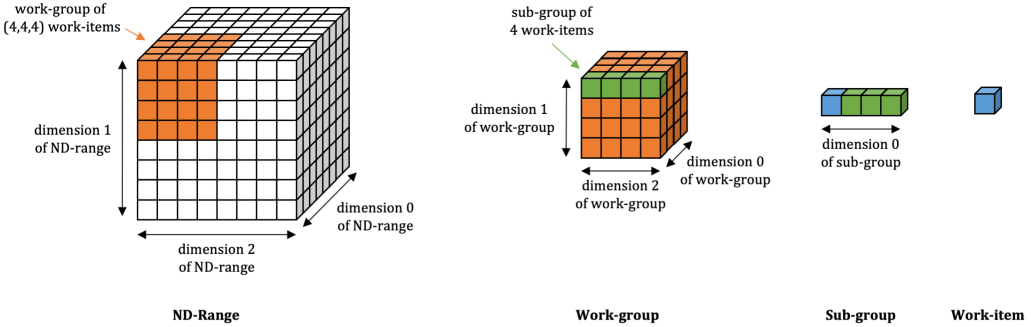
- Two categories of errors exist:
  - runtime error due to scheduling errors that may happen during execution
  - device error due to the execution errors on a SYCL device
- Some are thrown asynchronously at the use of a SYCL API (caught by try/catch in code to left).
- Some are thrown synchronously, they are stored by the runtime and passed to and async_handler (myAsyncHandler in code to left).
- We will come back to error handling in a *future* training module, so I will completely ignore error handling for all examples today. ☺

§2. EXECUTION MODEL



1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

LANGUAGE OF THE HIERARCHY ABSTRACTION (FULL 3D)



DPC++ vocabulary follows and extends vocabulary of CUDA, OpenCL, SYCL.

EXECUTION MODEL, 1 OF 7

Kernel functions are executed as work-items.

- Like a thread, yet very different from a C++ thread
- A work-item cannot synchronize with another work-item (achieve by kernel must end, submit another kernel invocation). Wait - sub-groups and work-groups offer synchronizations.
  - *could* be a OS thread
  - but it *could* be done on a GPU element
  - *or* it could be processed in an FPGA
  - *or* it could be processed in a DSP
  - *or* it could be processed in an AI accelerator

Work-item

Such flexibility for target, brings some restrictions and responsibilities.

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

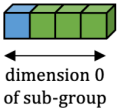
Not supported:

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

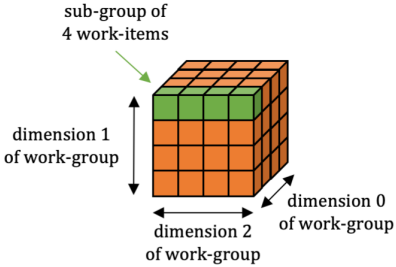
EXECUTION MODEL, 2 OF 7

We collect work-items into (work) sub-groups, and work-groups.

- There are sub-group and group barriers, which force all work-items in a particular sub-group or group to reach a certain point. There are also group and sub-group memory fences to manage memory consistency.
- SYCL (currently) only provides work-groups.
- work-items are single items (1D)
- work-groups can be 3D (or 2D)
- work-subgroup (DPC++ specific) give us a 2D option (useful when working in 3D spaces)
- When optimizing for performance, the choice of sizes for work-groups, and for sub-groups, have a connection to the capabilities of the device(s) being targeted. This is an advanced topic we will discuss in a future training module, and not important for making functional DPC++ code.



Sub-group

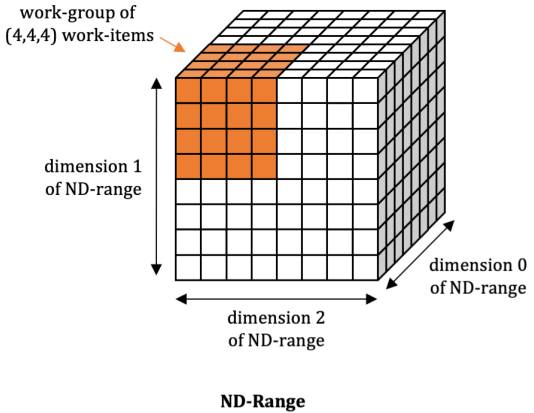


Work-group

EXECUTION MODEL, 3 OF 7

Kernel functions are invoked in an ND-Range.

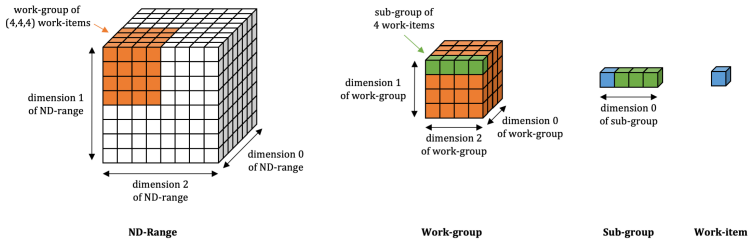
- An ND-Range consists of work-groups, which consist of sub-groups, which consist of work-items.
- Work-groups in a range, all have the same size (number of sub-groups, total number of work-items).
- Work sub-groups in a range, all have the same size (number of work-items).



EXECUTION MODEL, 4 OF 7

While an ND Range can be 3D, it can also be 2D, or 1D.

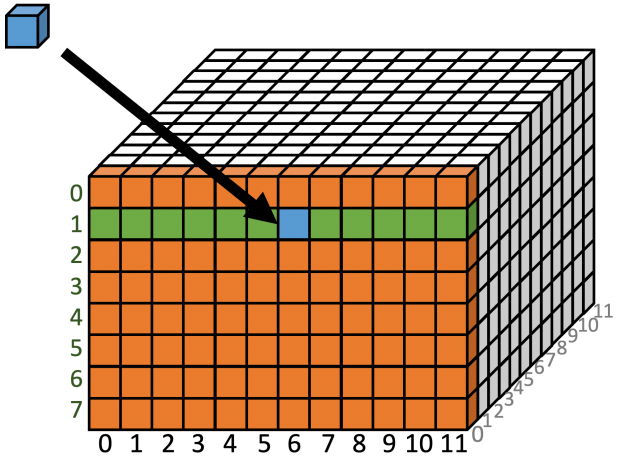
- Dimension support is for programmer convenience, the mapping to a contiguous linear block of a memory always sits underneath in a predictable realiable fashion - allowing us to reason about locality (for optimization).
- There is no built-in support for more than three dimensions.



EXECUTION MODEL, 5 OF 7

An ND-Range has a global and local components.

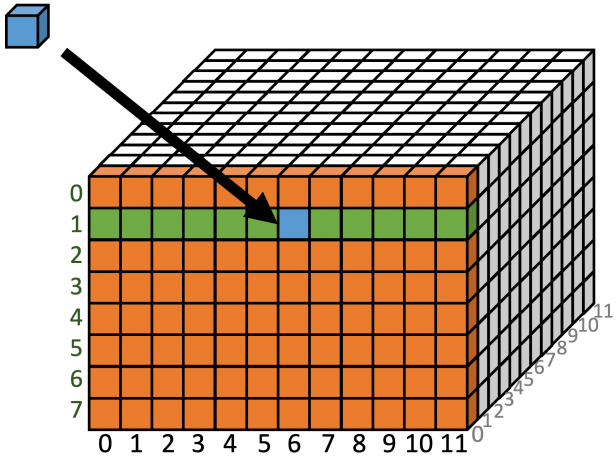
- global-range describes the total number of work-items in each dimension
- local-range describes number of work-items in a work-group in a dimension



EXECUTION MODEL, 6 OF 7

Each invocation of a kernel is based on a particular work-item.

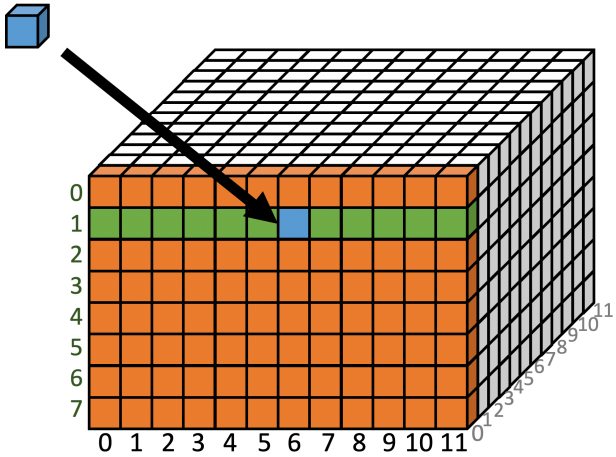
- This is in the 'execution space' which can be different that we think about the data. More on that in an example later in this training module.



EXECUTION MODEL, 7 OF 7

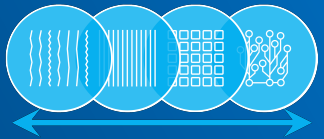
Information on the work-item is available to the kernel via queries.

```
global range          {12, 8, 12}
global id             {6, 1, 0}
global linear id      588
group range           {12, 8, 1}
group                 {6, 1, 0}
group linear id       49
local range           {1, 1, 12}
local id              {0, 0, 0}
local linear id       0

subgroup group range          1
subgroup id                   {0}
subgroup local range          {12}
subgroup local id             {0}
subgroup uniform group range  1
subgroup max local range      {12}
```



Output from module03/explore12 sample program.

§3. WHERE AND HOW TO GET AND USE DPC++, ETC.



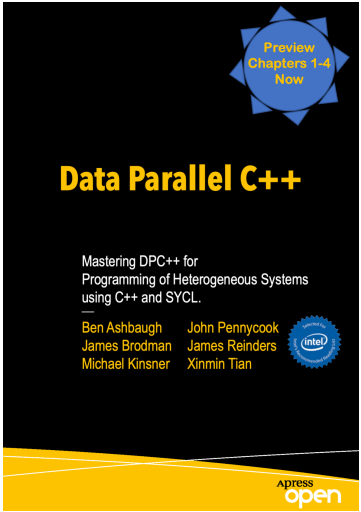
1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

RESOURCES

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

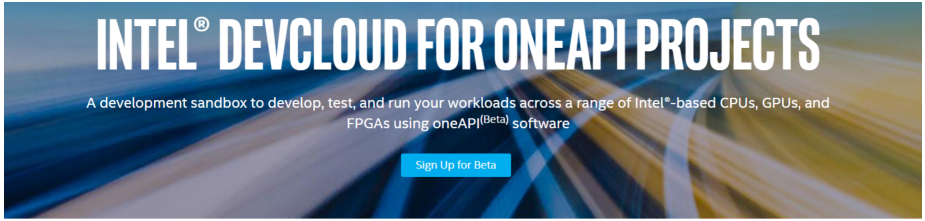
All available
Free

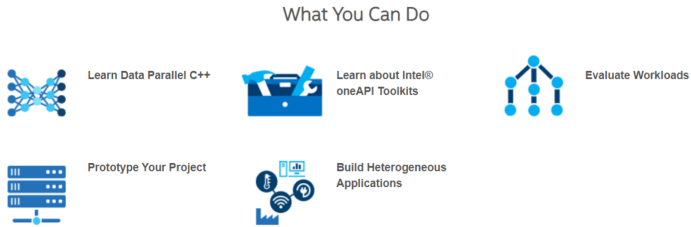
https://software.intel.com/en-us/oneapi



https://tinyurl.com/book-dpcpp
http://tinyurl.com/oneapimodule?3

DEVCLOUD





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

EASIEST - USE THE PREBUILT DPC++ WITH COMPLETE ONEAPI TOOLKITS

- DevCloud
- Download Toolkits

You'll want oneAPI toolkits, even if you build your own DPC++ compiler.

BUILD FROM OPEN SOURCE, EASY LINUX OR WINDOWS

https://tinyurl.com/openSYCL

Branch: sycl ▾ llvm / sycl / doc / GetStartedWithSYCLCompiler.md

Prerequisites

- `git` - https://git-scm.com/downloads
- `cmake` version 3.2 or later - http://www.cmake.org/download/
- `python` - https://www.python.org/downloads/release/python-2716/
- C++ compiler
  - Linux: `GCC` version 5.1.0 or later (including libstdc++) - https://gcc.gnu.org/install/
  - Windows: `Visual Studio` version 15.7 preview 4 or later - https://visualstudio.microsoft.com/downloads/

BUILD FROM OPEN SOURCE, LINUX (FOR EXAMPLE)

```
export SYCL_HOME=/export/home/sycl_workspace
mkdir $SYCL_HOME
```

```
cd $SYCL_HOME
git clone https://github.com/intel/llvm -b sycl
mkdir $SYCL_HOME/build
cd $SYCL_HOME/build

cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD="X86" \
-DLLVM_EXTERNAL_PROJECTS="llvm-spirv;sycl" \
-DLLVM_ENABLE_PROJECTS="clang;llvm-spirv;sycl" \
-DLLVM_EXTERNAL_SYCL_SOURCE_DIR=$SYCL_HOME/llvm/sycl \
-DLLVM_EXTERNAL_LLVM_SPIRV_SOURCE_DIR=$SYCL_HOME/llvm/llvm-spirv \
$SYCL_HOME/llvm/llvm

make -j`nproc` sycl-toolchain
```

```
export PATH=$SYCL_HOME/build/bin:$PATH
export LD_LIBRARY_PATH=$SYCL_HOME/build/lib:$LD_LIBRARY_PATH
```

```
clang++ -fsycl foo.cpp
```

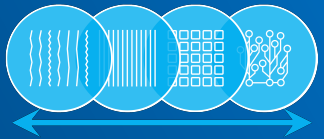
https://tinyurl.com/openSYCL

PROGRAMMING IN DPC++

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

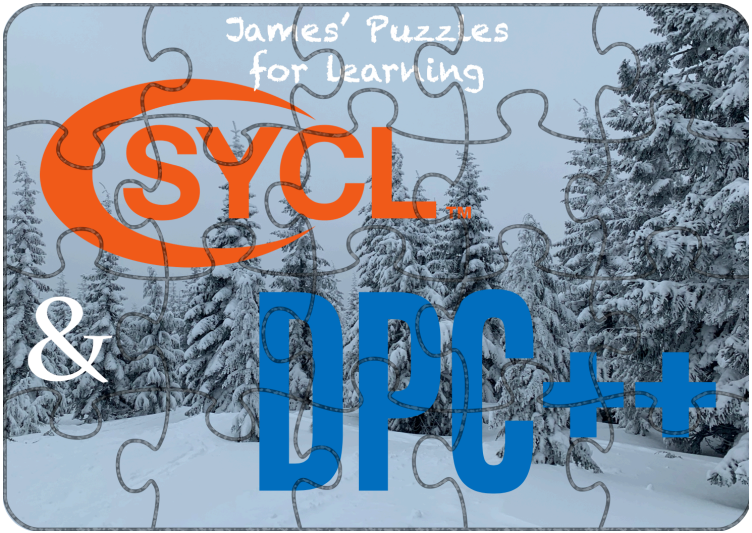
- Write `kernels'
- Control when/where/how they might be accelerated

§4. ID, ITEM, ND_ITEM



1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

PUZZLES



code for this module: http://tinyurl.com/oneapimodule?3

JUMP ON DEVCLOUD

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

$ wget tinyurl.com/oneapimodule?3 -O 3.tz
... note the -O option does use a capital letter O...
$ tar xvfz 3.tz
...fetch and unpack code I'll be playing with for module 3...

$ pbsnodes -l free
...see if many are free...
...if only a handful are free... please BE POLITE and use batch...
...training module 2 (please WATCH) explains all this...

$ qsub -I -lnodes=1:ppn=2
... I am using this because lots of nodes are free and available...
```

CODE: SETUP X[ ][ ][ ]=7, Y[ ][ ][ ]=8, Z[ ][ ][ ]=9

```
1  // Set up SYCL device and queue.
2  queue q = queue();
3
4  // ultimately we will be 4 x 4 x 4 in 3D
5  const uint32_t D = 4;
6
7  std::vector<int> x(D*D*D);
8  std::vector<int> y(D*D*D);
9  std::vector<int> z(D*D*D);
10 std::fill(x.begin(), x.end(), 7);
11 std::fill(y.begin(), y.end(), 8);
12 std::fill(z.begin(), z.end(), 9);
13
14 {
15   // buffers for device access to x[], y[], and z[]
16   buffer<int,1> x_buf(x.data(), range<1>(D*D*D));
17   buffer<int,1> y_buf(y.data(), range<1>(D*D*D));
18   buffer<int,1> z_buf(z.data(), range<1>(D*D*D));
19
20   q.submit([&](handler& cgh) {
21
22         // accessors are way for device to touch shared data
23         auto xx = x_buf.get_access<access::mode::read_write>(cgh);
24         auto yy = y_buf.get_access<access::mode::read_write>(cgh);
25         auto zz = z_buf.get_access<access::mode::read_write>(cgh);
```

- explore1.cpp
- more observations

CODE: EXPLORE1.CPP

```
1  cgh.parallel_for<class foo>(range<1>{D*D*D}, [=](id<1> item) {
2    xx[ 3 ] = 3;
3    yy[ 4 ] = 4;
4    zz[ 5 ] = 5;
5  });
6
7
8     0         1         2         3         4         5         6
9     .123456789.123456789.123456789.123456789.123456789.123456789.123
10 x: 777377777777777777777777777777777777777777777777777777777777777777
11 y: 888848888888888888888888888888888888888888888888888888888888888888
12 z: 999995999999999999999999999999999999999999999999999999999999999999
```

- explore1.cpp
- sample output (output should not vary)

CODE: EXPLORE2.CPP

```
1  cgh.parallel_for<class foo>(range<1>{D*D*D}, [=](id<1> item) {
2    xx[ item[0] - 1 ] = 3;
3    yy[ item[0]     ] = item[0] % 10;
4    zz[ item[0] + 1 ] = 5;
5  });
6
7
8    0         1         2         3         4         5         6
9     .123456789.123456789.123456789.123456789.123456789.123456789.123
10 x: 33333333333333333333333333333333333333333333333333333333333333337
11 y: 01234567890123456789012345678901234567890123456789012345678901230
12 z: 95555555555555555555555555555555555555555555555555555555555555555
13
14 double free or corruption (out)
15 Aborted
```

- explore2.cpp
- sample output (output may vary, if any (due to error))
- NOTE: simply compiling with -g helps get errors explained (therefore, Makefile has -g)

CODE: EXPLORE3.CPP

```
1  cgh.parallel_for<class foo1D>(range<1>{D*D*D}, [=](id<1> item) {
2    if (item[0] > 0)
3      xx[ item[0] - 1 ] = 3;
4    yy[ item[0]     ] = flag ? -item[0] : item[0] % 10;
5    if (item[0] < (D*D*D+1))
6      zz[ item[0] + 1 ] = 5;
7  });
8
9  $ ./explore3 (no arguments)
10
11 0         1         2         3         4         5         6
12    .123456789.123456789.123456789.123456789.123456789.123456789.123
13 x: 33333333333333333333333333333333333333333333333333333333333333337
14 y: 01234567890123456789012345678901234567890123456789012345678901230
15 z: 95555555555555555555555555555555555555555555555555555555555555555
16
17 $ ./explore3 X
18
19    0         1         2         3         4         5         6
20    .123456789.123456789.123456789.123456789.123456789.123456789.123
21 x: 33333333333333333333333333333333333333333333333333333333333333337
22 y: 0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-4
23 z: 95555555555555555555555555555555555555555555555555555555555555555
```

- explore3.cpp
- sample outputs (output should not vary)

CODE: EXPLORE4.CPP

```
1  if (flag & 1) {
2    cgh.parallel_for<class foo1D>(range<1>{D*D*D}, [=](id<1> item) {
3      xx[ item[0] ] = 1;
4    });
5  }
6  if (flag & 2) {
7    cgh.parallel_for<class foo2D>(range<2>{D,D*D}, [=](id<2> item) {
8      yy[ item[0] + D*item[1] ] = 2;
9    });
10 }
11 if (flag & 4) {
12   cgh.parallel_for<class foo3D>(range<3>{D,D,D}, [=](id<3> item) {
13     zz[ item[0] + D*(item[1]+item[2]*D) ] = 3;
14   });
15 }
16
17 $ ./explore4 7
18
19    0         1         2         3         4         5         6
20    .123456789.123456789.123456789.123456789.123456789.123456789.123
21 x: 7777777777777777777777777777777777777777777777777777777777777777777
22 y: 8888888888888888888888888888888888888888888888888888888888888888888
23 z: 3333333333333333333333333333333333333333333333333333333333333333333
```

- explore4.cpp
- why this output?

OUTPUT: EXPLORE4.CPP

```
$ ./explore4 1

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 11111111111111111111111111111111111111111111111111111111111111111
y: 88888888888888888888888888888888888888888888888888888888888888888
z: 99999999999999999999999999999999999999999999999999999999999999999

$ ./explore4 2

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 22222222222222222222222222222222222222222222222222222222222222222
z: 99999999999999999999999999999999999999999999999999999999999999999

$ ./explore4 3

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 22222222222222222222222222222222222222222222222222222222222222222
z: 99999999999999999999999999999999999999999999999999999999999999999
```

```
$ ./explore4 4

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 88888888888888888888888888888888888888888888888888888888888888888
z: 33333333333333333333333333333333333333333333333333333333333333333

$ ./explore4 7

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 88888888888888888888888888888888888888888888888888888888888888888
z: 33333333333333333333333333333333333333333333333333333333333333333
```

- explore4.cpp
- sample outputs (output may vary based on implementation)

LOOK AGAIN: EXPLORE4.CPP

```
1  if (flag & 1) {
2    cgh.parallel_for<class foo1D>(range<1>{D*D*D}, [=](id<1> item) {
3      xx[ item[0] ] = 1;
4    });
5  }
6  if (flag & 2) {
7    cgh.parallel_for<class foo2D>(range<2>{D,D*D}, [=](id<2> item) {
8      yy[ item[0] + D*item[1] ] = 2;
9    });
10 }
11 if (flag & 4) {
12   cgh.parallel_for<class foo3D>(range<3>{D,D,D}, [=](id<3> item) {
13     zz[ item[0] + D*(item[1]+item[2]*D) ] = 3;
14   });
15 }
16
17 $ ./explore4 7
18
19    0         1         2         3         4         5         6
20    .123456789.123456789.123456789.123456789.123456789.123456789.123
21 x: 7777777777777777777777777777777777777777777777777777777777777777777
22 y: 8888888888888888888888888888888888888888888888888888888888888888888
23 z: 3333333333333333333333333333333333333333333333333333333333333333333
```

- explore4.cpp
- figure it out?

REVIEW: PARTS OF DPC++ PROGRAM, 3 OF 6

```
#include <CL/sycl.hpp>
using namespace cl::sycl;
int main(int argc, char *argv[]) {

  ...


  queue myQueue{...};

  ...

  myQueue.submit([&](handler &cgh) {

    // accessors (for connecting to memory via buffers)
    // kernel defined here (with lambda -
    // by value captures only)

  });


}
```

- Queue accepts work requests as submissions.
- Highlighted lines are the *command group scope*.
- Submissions finish asynchronously.
- Only one kernel (work described in a lambda) per submit!

UNDERSTANDING: EXPLORE4.CPP

```
$ ./explore4 1

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 11111111111111111111111111111111111111111111111111111111111111111
y: 88888888888888888888888888888888888888888888888888888888888888888
z: 99999999999999999999999999999999999999999999999999999999999999999

$ ./explore4 2

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 22222222222222222222222222222222222222222222222222222222222222222
z: 99999999999999999999999999999999999999999999999999999999999999999

$ ./explore4 4

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 77777777777777777777777777777777777777777777777777777777777777777
y: 88888888888888888888888888888888888888888888888888888888888888888
z: 33333333333333333333333333333333333333333333333333333333333333333
```

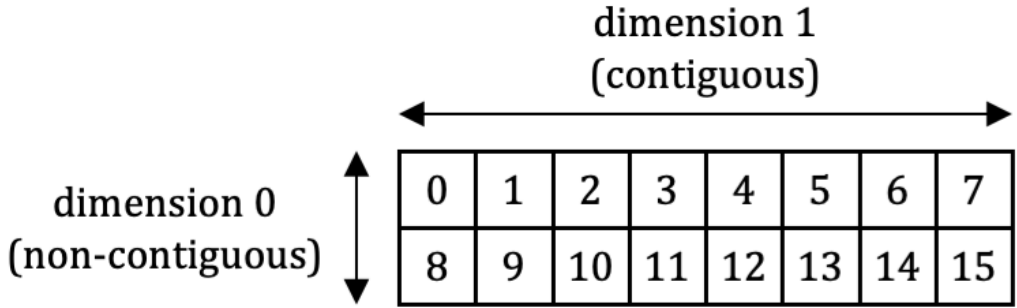
- explore4.cpp
- if we choose only one kernel, everything works
- if we choose more, that's illegal - and not defined at all

ANOTHER VIEW: SAME PROBLEM

```
// Case 1 (Wrong)
for (size_t j=0; j < TRIAL; j++) {
  cgh.parallel_for<class VectorAdd>(num_items, [=](id<1> wiID) {
    sum_accessor[wiID] += addend_1_accessor[wiID] + addend_2_accessor[wiID];
  });
}

// Case 2 (Correct)
cgh.parallel_for<class VectorAdd>(num_items, [=](id<1> wiID) {
  for (size_t j=0; j < TRIAL; j++) {
    sum_accessor[wiID] += addend_1_accessor[wiID] + addend_2_accessor[wiID];
  }
});
```

The first code was a failed attempt to loop
a bunch of times for checking performance.
Oooops. ☺

ROW MAJOR



Dimensions are numbered 0, 1, 2. x[0] y[0][1] z[0][1][2]

CODE: EXPLORE5.CPP

```
  q.submit([&](handler& cgh) {
    ...
        cgh.parallel_for<class foo1D>(range<1>{D*D*D}, [=](id<1> item) {
            xx[ item[0] ] = 1;
          });
    });
  printit

  q.submit([&](handler& cgh) {
    ...
        cgh.parallel_for<class foo2D>(range<2>{D,D*D}, [=](id<2> item) {
            xx[ item[0] + D*item[1] ] = 2;
          });
    });
  printit

  q.submit([&](handler& cgh) {
    ...
        cgh.parallel_for<class foo3D>(range<3>{D,D,D}, [=](id<3> item) {
            xx[ item[0] + D*(item[1]+item[2]*D) ] = 3;
          });
    });
}
printit
```

- explore5.cpp
- Will this work?

OUTPUT FROM EXPLORE5.CPP

```
Note: actual output is a little more verbose, condensed here for viewing!

$ ./explore5
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 3333333333333333333333333333333333333333333333333333333333333333
$ ./explore5
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333
$ ./explore5
x: 7777777777777777777777777777777777777777777777777777777777777777
x: 7777777777777777777777777777777777777777777777777777777777777777
x: 3333333333333333333333333333333333333333333333333333333333333333

Above runs on default device (GPU when I ran these) were inconsistent.

$ ./explore5 h
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333
$ ./explore5 h
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333

When run on the host device ('h' option above)...
output were always correct (I tried many many times!).
```

- explore5.cpp
- sample outputs (output should not vary)

CODE: EXPLORE6.CPP

```
1  {
2    // buffers...
3    q.submit([&](handler& cgh) {
4      ...
5      });
6  }
7  printit
8  {
9    // buffers...
10   q.submit([&](handler& cgh) {
11     ...
12     });
13 }
14 printit
15 {
16   // buffers...
17   q.submit([&](handler& cgh) {
18     ...
19     });
20 }
21 printit
```

- explore6.cpp
- Forced synchronization is one possible fix.

OUTPUT: EXPLORE6.CPP

```
$ ./explore6
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333

$ ./explore6
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333

$ ./explore6
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333

$ ./explore6
x: 1111111111111111111111111111111111111111111111111111111111111111
x: 2222222222222222222222222222222222222222222222222222222222222222
x: 3333333333333333333333333333333333333333333333333333333333333333
```

- explore6.cpp
- output is deterministic
- Yeah!

CODE: EXPLORE7.CPP

```
// std::cout will not work, and is not permitted, in device code.
// Fortunately, SYCL provides a stream class which is useful for
// sending messages to standard output from device code.
// This can be very useful for debugging!
// SYCL 1.2.1r5 section 4.12 "Stream class"
q.submit([&](handler& cgh) {
  cl::sycl::stream kernelout(512*D*D*D*1024, 512, cgh);
  ...
  cgh.parallel_for<class foo1D>(range<1>{D*D*D},
    [=](id<1> item) {
      xx[ item[0] ] = 1;
      kernelout << "Hello:" << item[0] << cl::sycl::endl;
    });
});
```

- explore7.cpp
- printing from a kernel is useful
- stream is implemented to hold lines together
- lines are delivered asynchronously, which will interleave (non-deterministic ordering)
- explicit use of cl::sycl:: avoids confusion with std::

OUTPUT: EXPLORE7.CPP

```
Hello:0
Hello:1
Hello:2
Hello:3
Hello:4
Hello:5
Hello:6
Hello:7
Hello:8
Hello:9
Hello:10
Hello:11
Hello:12
Hello:13
Hello:14
Hello:15
Hello:16
Hello:17
Hello:18
Hello:19
Hello:20
Hello:21
Hello:22
Hello:23
Hello:24
Hello:25
```

```
Hello:26
Hello:27
Hello:28
Hello:29
Hello:30
Hello:31
Hello:32
Hello:33
Hello:34
Hello:35
Hello:36
Hello:37
Hello:38
Hello:39
Hello:40
Hello:41
Hello:42
Hello:43
Hello:44
Hello:45
Hello:46
Hello:47
Hello:48
Hello:49
Hello:50
Hello:51
```

```
Hello:52
Hello:53
Hello:54
Hello:55
Hello:56
Hello:57
Hello:58
Hello:59
Hello:60
Hello:61
Hello:62
Hello:63

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 11111111111111111111111111111111111111111111111111111111111111111
```

- explore7.cpp
- sample output (output may vary in ordering)

CODE: EXPLORE8.CPP

```
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo1D>(range<1>{D*D*D},
  [=](id<1> item) { xx[ item[0] ] = 1;
    kernelout << "1D(" ...; }); });
printit
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo2D>(range<2>{D,D*D},
  [=](id<2> item) { xx[ item[0] + D*item[1] ] = 2;
    kernelout << "2D(" ...; }); });
printit
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo3D>(range<3>{D,D,D},
  [=](id<3> item) {
    xx[ item[0] + D*(item[1]+item[2]*D) ] = 3;
    kernelout << "3D(" ...; }); });
printit
```

- explore8.cpp
- more observations

OUTPUT (1 OF 3): EXPLORE8.CPP

```
1D(0)
1D(1)
1D(2)
1D(3)
1D(4)
1D(5)
1D(6)
1D(7)
1D(8)
1D(9)
1D(10)
1D(11)
1D(12)
1D(13)
1D(14)
1D(15)
1D(16)
1D(17)
1D(18)
1D(19)
1D(20)
1D(21)
1D(22)
1D(23)
1D(24)
1D(25)
```

```
1D(26)
1D(27)
1D(28)
1D(29)
1D(30)
1D(31)
1D(32)
1D(33)
1D(34)
1D(35)
1D(36)
1D(37)
1D(38)
1D(39)
1D(40)
1D(41)
1D(42)
1D(43)
1D(44)
1D(45)
1D(46)
1D(47)
1D(48)
1D(49)
1D(50)
1D(51)
```

```
1D(52)
1D(53)
1D(54)
1D(55)
1D(56)
1D(57)
1D(58)
1D(59)
1D(60)
1D(61)
1D(62)
1D(63)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 11111111111111111111111111111111111111111111111111111111111111111
```

- explore8.cpp
- sample output (output may vary - ordering and assignments)

OUTPUT (2 OF 3): EXPLORE8.CPP

```
2D(0,0)
2D(0,1)
2D(0,2)
2D(0,3)
2D(1,0)
2D(1,1)
2D(1,2)
2D(1,3)
2D(0,4)
2D(0,5)
2D(0,6)
2D(0,7)
2D(1,4)
2D(1,5)
2D(1,6)
2D(1,7)
2D(2,0)
2D(2,1)
2D(2,2)
2D(2,3)
2D(3,0)
2D(3,1)
2D(3,2)
2D(3,3)
2D(2,4)
2D(2,5)
```

```
2D(2,6)
2D(2,7)
2D(3,4)
2D(3,5)
2D(3,6)
2D(3,7)
2D(0,8)
2D(0,9)
2D(0,10)
2D(0,11)
2D(1,8)
2D(1,9)
2D(1,10)
2D(1,11)
2D(0,12)
2D(0,13)
2D(2,8)
2D(0,14)
2D(0,15)
2D(2,9)
2D(1,12)
2D(1,13)
2D(2,10)
2D(2,11)
2D(1,14)
2D(1,15)
```

```
2D(2,12)
2D(3,8)
2D(3,10)
2D(3,9)
2D(3,11)
2D(2,13)
2D(2,14)
2D(2,15)
2D(3,14)
2D(3,15)
2D(3,12)
2D(3,13)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 2222222222222222222222222222222222222222222222222222222222222222
```

- explore8.cpp
- sample output (output may vary - ordering and assignments)

OUTPUT (3 OF 3): EXPLORE8.CPP

```
3D(0,0,0)
3D(0,0,1)
3D(0,0,2)
3D(0,0,3)
3D(0,1,0)
3D(0,1,1)
3D(0,1,2)
3D(0,1,3)
3D(1,0,0)
3D(1,0,1)
3D(1,0,2)
3D(1,0,3)
3D(1,1,0)
3D(1,1,1)
3D(1,1,2)
3D(1,1,3)
3D(0,2,0)
3D(0,2,1)
3D(0,2,2)
3D(0,2,3)
3D(0,3,0)
3D(0,3,1)
3D(0,3,2)
3D(0,3,3)
3D(1,2,0)
3D(1,2,1)
```

```
3D(1,2,2)
3D(1,2,3)
3D(1,3,0)
3D(1,3,1)
3D(1,3,2)
3D(1,3,3)
3D(2,0,0)
3D(2,0,1)
3D(2,0,2)
3D(2,0,3)
3D(2,1,0)
3D(2,1,1)
3D(2,1,2)
3D(2,1,3)
3D(3,0,0)
3D(3,0,1)
3D(3,0,2)
3D(3,0,3)
3D(3,1,0)
3D(3,1,1)
3D(3,1,2)
3D(3,1,3)
3D(2,2,0)
3D(2,2,1)
3D(2,2,2)
3D(2,2,3)
```

```
3D(2,3,0)
3D(2,3,1)
3D(2,3,2)
3D(2,3,3)
3D(3,2,0)
3D(3,2,1)
3D(3,2,2)
3D(3,2,3)
3D(3,3,0)
3D(3,3,1)
3D(3,3,2)
3D(3,3,3)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 333333333333333333333333333333333333333333333333333333333333333
```

- explore8.cpp
- sample output (output may vary - ordering and assignments)

CODE: EXPLORE9.CPP

```
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo3Did>(range<3>{D,D,D},
    [=](id<3> item) {
...
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo3Ditem>(range<3>{D,D,D},
    [=](item<3> item) {
...
q.submit([&](handler& cgh) {
  ...
  cgh.parallel_for<class foo3Dnd_item>(nd_range<3>({D,D,D},{2,2,2}),
    [=](nd_item<3> item) {
...
```

- explore9.cpp
- id, item, nd_item

OUTPUT: EXPLORE9.CPP - ID

```
3D(0,0,0)
3D(0,0,1)
3D(0,0,2)
3D(0,0,3)
3D(0,1,0)
3D(0,1,1)
3D(0,1,2)
3D(0,1,3)
3D(1,0,0)
3D(1,0,1)
3D(1,0,2)
3D(1,0,3)
3D(1,1,0)
3D(1,1,1)
3D(1,1,2)
3D(1,1,3)
3D(0,2,0)
3D(0,2,1)
3D(0,2,2)
3D(0,2,3)
3D(0,3,0)
3D(0,3,1)
3D(0,3,2)
3D(0,3,3)
3D(1,2,2)
3D(1,2,3)
```

```
3D(1,2,0)
3D(1,3,0)
3D(1,2,1)
3D(1,3,1)
3D(2,2,0)
3D(1,3,2)
3D(1,3,3)
3D(2,0,0)
3D(2,2,1)
3D(2,2,2)
3D(2,2,3)
3D(2,0,1)
3D(2,0,2)
3D(2,0,3)
3D(2,1,0)
3D(2,1,1)
3D(2,1,2)
3D(2,1,3)
3D(2,3,0)
3D(2,3,1)
3D(2,3,2)
3D(2,3,3)
3D(3,1,0)
3D(3,1,1)
3D(3,1,2)
3D(3,1,3)
```

```
3D(3,2,0)
3D(3,2,1)
3D(3,2,2)
3D(3,2,3)
3D(3,3,0)
3D(3,3,1)
3D(3,3,2)
3D(3,3,3)
3D(3,0,2)
3D(3,0,3)
3D(3,0,0)
3D(3,0,1)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 333333333333333333333333333333333333333333333333333333333333333
```

- explore9.cpp
- id<3>
- sample output (output may vary - ordering and assignments)

OUTPUT: EXPLORE9.CPP - ITEM

```
3D(0,0,0)
3D(0,0,1)
3D(0,0,2)
3D(0,0,3)
3D(0,1,0)
3D(0,1,1)
3D(0,1,2)
3D(0,1,3)
3D(2,0,0)
3D(1,0,0)
3D(2,0,1)
3D(1,0,1)
3D(1,0,2)
3D(1,0,3)
3D(1,1,0)
3D(1,1,1)
3D(1,1,2)
3D(1,1,3)
3D(2,0,2)
3D(2,0,3)
3D(0,2,0)
3D(0,2,1)
3D(2,2,0)
3D(0,2,2)
3D(0,2,3)
3D(2,3,0)
```

```
3D(2,2,1)
3D(2,3,1)
3D(0,3,0)
3D(0,3,1)
3D(2,2,2)
3D(0,3,2)
3D(2,3,2)
3D(0,3,3)
3D(2,3,3)
3D(2,1,0)
3D(2,2,3)
3D(2,1,1)
3D(1,2,0)
3D(1,2,1)
3D(3,2,0)
3D(1,2,2)
3D(3,2,1)
3D(1,2,3)
3D(1,3,0)
3D(1,3,1)
3D(2,1,2)
3D(1,3,2)
3D(2,1,3)
3D(1,3,3)
3D(3,0,0)
3D(3,0,1)
```

```
3D(3,0,2)
3D(3,2,2)
3D(3,2,3)
3D(3,3,0)
3D(3,3,1)
3D(3,0,3)
3D(3,3,2)
3D(3,3,3)
3D(3,1,0)
3D(3,1,1)
3D(3,1,2)
3D(3,1,3)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 444444444444444444444444444444444444444444444444444444444444444
```

- explore9.cpp
- item<3>
- sample output (output may vary - ordering and assignments)

OUTPUT: EXPLORE9.CPP - ND_ITEM

```
3D(0,0,0)
3D(1,0,0)
3D(0,1,0)
3D(1,1,0)
3D(0,0,1)
3D(1,0,1)
3D(0,1,1)
3D(1,1,1)
3D(2,0,0)
3D(3,0,0)
3D(2,1,0)
3D(3,1,0)
3D(2,0,1)
3D(3,0,1)
3D(2,1,1)
3D(3,1,1)
3D(0,2,0)
3D(1,2,0)
3D(0,3,0)
3D(1,3,0)
3D(0,2,1)
3D(1,2,1)
3D(0,3,1)
3D(1,3,1)
3D(2,2,0)
3D(3,2,0)
```

```
3D(2,3,0)
3D(3,3,0)
3D(2,2,1)
3D(3,2,1)
3D(2,3,1)
3D(3,3,1)
3D(0,0,2)
3D(1,0,2)
3D(0,1,2)
3D(1,1,2)
3D(0,0,3)
3D(1,0,3)
3D(0,1,3)
3D(1,1,3)
3D(2,0,2)
3D(3,0,2)
3D(2,1,2)
3D(3,1,2)
3D(2,0,3)
3D(3,0,3)
3D(2,1,3)
3D(3,1,3)
3D(0,2,2)
3D(1,2,2)
3D(0,3,2)
3D(1,3,2)
```

```
3D(0,2,3)
3D(1,2,3)
3D(0,3,3)
3D(1,3,3)
3D(2,2,2)
3D(3,2,2)
3D(2,3,2)
3D(3,3,2)
3D(2,2,3)
3D(3,2,3)
3D(2,3,3)
3D(3,3,3)

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 55555555555555555555555555555555555555555555555555555555555555555
```

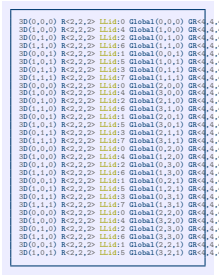
- explore9.cpp
- nd_item<3>
- sample output (output may vary - ordering and assignments)

OUTPUT: CODE10.CPP

```
cgh.parallel_for<class foo3Dnd_item>(nd_range<3>({D,D,D},{2,2,2}), [=](nd_item<3> item) {
  xx[ item.get_global_id()[0] +
    D*(item.get_global_id()[1]+item.get_global_id()[2]*D) ] = 5;
  kernelout << "3D(" << item.get_local_id()[0] << ","
  << item.get_local_id()[1] << "," << item.get_local_id()[2] << ") R<"
  << item.get_local_range()[0] << "," << item.get_local_range()[1] << ","
  << item.get_local_range()[2] << "> LLid:"
  << item.get_local_linear_id() << " Global("
  << item.get_global_id()[0] << "," << item.get_global_id()[1] << ","
  << item.get_global_id()[2] << ") GR<"
  << item.get_global_range()[0] << "," << item.get_global_range()[1] << ","
  << item.get_global_range()[2] << "> GLid:"
  << item.get_global_linear_id() << " group:("
  << item.get_group()[0] << "," << item.get_group()[1] << ","
  << item.get_group()[2] << ") R<"
  << item.get_group_range()[0] << "," << item.get_group_range()[1] << ","
  << item.get_group_range()[2] << "> Grpid:"
  << item.get_group_linear_id() << cl::sycl::endl;
  });
printit
...
cgh.parallel_for<class foo2Dnd_item>(nd_range<2>({D,D*D},{2,2}), [=](nd_item<2> item) {
  xx[ item.get_global_id()[0] + D*item.get_global_id()[1] ] = 8;
...
printit
...
cgh.parallel_for<class foo1Dnd_item>(nd_range<1>({D*D*D},{2}), [=](nd_item<1> item) {
  xx[ item.get_global_id()[0] ] = 2;
```

- explore10.cpp
- 3D, 2D, or 1D... it's the same data underneath

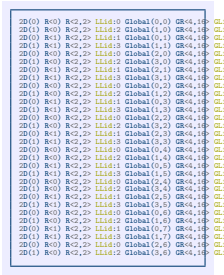
OUTPUT(3D): EXPLORE10.CPP



```
  ...
3D(0,1,0) R<2,2,2> LLid:2 Global(0,3,2) GR<4,4,4> GLid:14 group:(0,1,1) R<2,2,2> Grpid:3
3D(1,1,0) R<2,2,2> LLid:6 Global(1,3,2) GR<4,4,4> GLid:30 group:(0,1,1) R<2,2,2> Grpid:3
3D(0,0,1) R<2,2,2> LLid:1 Global(0,2,3) GR<4,4,4> GLid:11 group:(0,1,1) R<2,2,2> Grpid:3
3D(1,0,1) R<2,2,2> LLid:5 Global(1,2,3) GR<4,4,4> GLid:27 group:(0,1,1) R<2,2,2> Grpid:3
3D(0,1,1) R<2,2,2> LLid:3 Global(0,3,3) GR<4,4,4> GLid:15 group:(0... 
3D(1,1,1) R<2,2,2> LLid:7 Global(1,3,3) GR<4,4,4> GLid:31 group:(0... 
3D(0,0,0) R<2,2,2> LLid:0 Global(2,2,2) GR<4,4,4> GLid:42 group:(1... 
3D(1,0,0) R<2,2,2> LLid:4 Global(3,2,2) GR<4,4,4> GLid:58 group:(1... 
3D(0,1,0) R<2,2,2> LLid:2 Global(2,3,2) GR<4,4,4> GLid:46 group:(1... 
3D(1,1,0) R<2,2,2> LLid:6 Global(3,3,2) GR<4,4,4> GLid:62 group:(1... 
3D(0,0,1) R<2,2,2> LLid:1 Global(2,2,3) GR<4,4,4> GLid:43 group:(1... 
3D(1,0,1) R<2,2,2> LLid:5 Global(3,2,3) GR<4,4,4> GLid:59 group:(1... 
3D(0,1,1) R<2,2,2> LLid:3 Global(2,3,3) GR<4,4,4> GLid:47 group:(1... 
3D(1,1,1) R<2,2,2> LLid:7 Global(3,3,3) GR<4,4,4> GLid:63 group:(1... 

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 55555555555555555555555555555555555555555555555555555555555555555
```

- explore10.cpp
- sample output (output may vary - ordering and assignments)

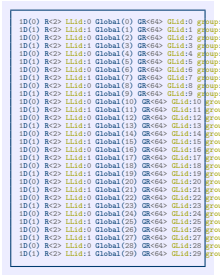
OUTPUT(2D): EXPLORE10.CPP



```
...
2D(0) R<0) R<2,2> LLid:0 Global(2,12) GR<4,16> GLid:44 group:(1,6) R<2,8> Grpid:14
2D(1) R<0) R<2,2> LLid:2 Global(3,12) GR<4,16> GLid:60 group:(1,6) R<2,8> Grpid:14
2D(0) R<1) R<2,2> LLid:1 Global(2,13) GR<4,16> GLid:45 group:(1,6) R<2,8> Grpid:14
2D(1) R<1) R<2,2> LLid:3 Global(3,13) GR<4,16> GLid:61 group:(1,6)...
2D(0) R<0) R<2,2> LLid:0 Global(0,14) GR<4,16> GLid:14 group:(0,7)...
2D(1) R<0) R<2,2> LLid:2 Global(1,14) GR<4,16> GLid:30 group:(0,7)...
2D(0) R<1) R<2,2> LLid:1 Global(0,15) GR<4,16> GLid:15 group:(0,7)...
2D(1) R<1) R<2,2> LLid:3 Global(1,15) GR<4,16> GLid:31 group:(0,7)...
2D(0) R<0) R<2,2> LLid:0 Global(2,14) GR<4,16> GLid:46 group:(1,7)...
2D(1) R<0) R<2,2> LLid:2 Global(3,14) GR<4,16> GLid:62 group:(1,7)...
2D(0) R<1) R<2,2> LLid:1 Global(2,15) GR<4,16> GLid:47 group:(1,7)...
2D(1) R<1) R<2,2> LLid:3 Global(3,15) GR<4,16> GLid:63 group:(1,7)...

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 888888888888888888888888888888888888888888888888888888888888888
```

- explore10.cpp
- sample output (output may vary - ordering and assignments)

OUTPUT(1D): EXPLORE10.CPP



```
  ...
1D(0) R<2> LLid:0 Global(54) GR<64> GLid:54 group:(27) R<32> Grpid:27
1D(1) R<2> LLid:1 Global(55) GR<64> GLid:55 group:(27) R<32> Grpid:27
1D(0) R<2> LLid:0 Global(56) GR<64> GLid:56 group:(28) R<32> Grpid:28
1D(1) R<2> LLid:1 Global(57) GR<64> GLid:57 group:(28) R<32> Grpid:28
1D(0) R<2> LLid:0 Global(58) GR<64> GLid:58 group:(29) R<32> Grpid:29
1D(1) R<2> LLid:1 Global(59) GR<64> GLid:59 group:(29) R<32> Grpid:29
1D(0) R<2> LLid:0 Global(60) GR<64> GLid:60 group:(30) R<32> Grpid:30
1D(1) R<2> LLid:1 Global(61) GR<64> GLid:61 group:(30) R<32> Grpid:30
1D(0) R<2> LLid:0 Global(62) GR<64> GLid:62 group:(31) R<32> Grpid:31
1D(1) R<2> LLid:1 Global(63) GR<64> GLid:63 group:(31) R<32> Grpid:31

   0         1         2         3         4         5         6
   .123456789.123456789.123456789.123456789.123456789.123456789.123
x: 22222222222222222222222222222222222222222222222222222222222222222
```

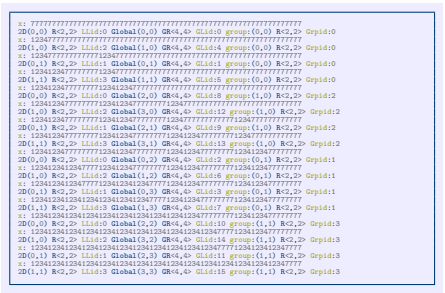
- explore10.cpp
- sample output (output may vary - ordering and assignments)

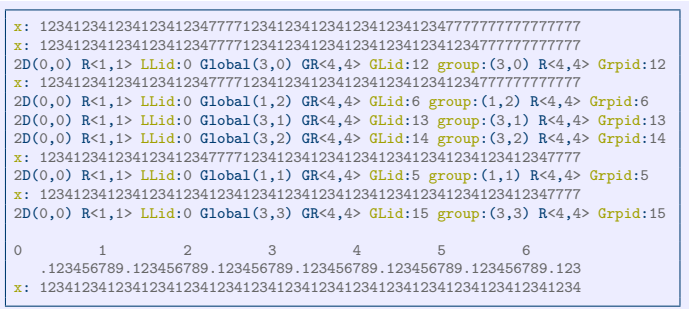
CODE: EXPLORE11.CPP

```
1  q.submit([&](handler& cgh) {
2  ...
3    // 2 dimensional execution shape used to
4    // give out a row at a time from a
5    // three dimensional data shape
6    cgh.parallel_for<class foo2Dnd_item>(nd_range<2>({D,D},{2,2}),
7    [=](nd_item<2> item) {
8      kernelout << "x: ";
9      for (int i = 0; i < 64; ++i) kernelout << xx[i];
10     kernelout << cl::sycl::endl;
11     xx[ item.get_global_linear_id()*D+0 ] = 1;
12     xx[ item.get_global_linear_id()*D+1 ] = 2;
13     xx[ item.get_global_linear_id()*D+2 ] = 3;
14     xx[ item.get_global_linear_id()*D+3 ] = 4;
15     kernelout << "2D(" << ... << cl::sycl::endl;
16   });
17 });
18 printit
```

- explore11.cpp
- the execution shape need not match the data shape

OUTPUT: EXPLORE11.CPP

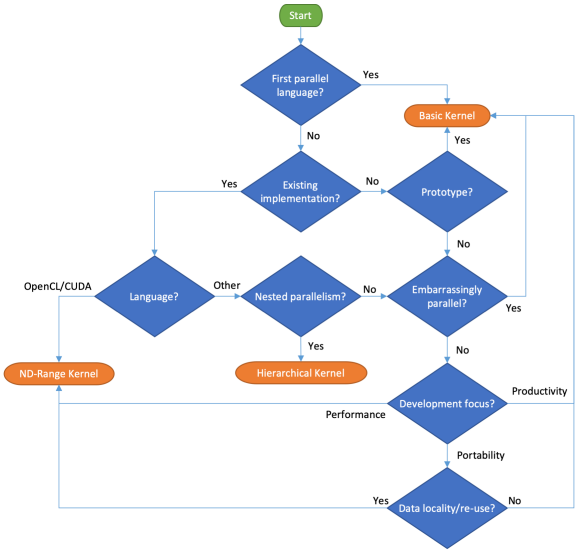




- explore11.cpp
- in this example, we purposefully handle a row at a time ourselves

- there are multiple ways to do this
- the KEY really is to *think* about the mappings we want

ND RANGE - SELECT FOR PERFORMANCE OR FAMILIARITY

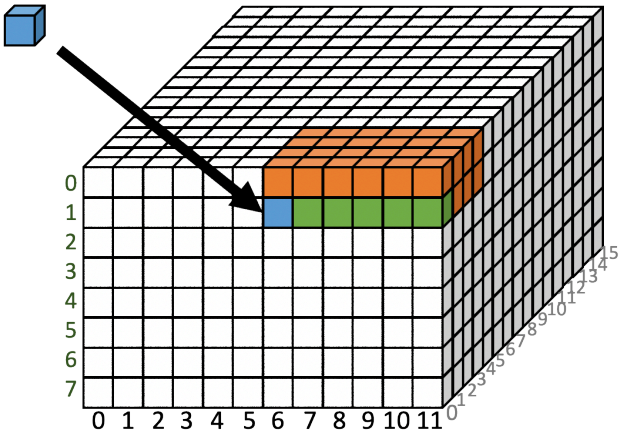


CODE: EXPLORE12.CPP

```
cgh.parallel_for<class foo3Dnd_item>(nd_range<3>{D1,D2,D3}, [=](nd_item<3> item) {
  if ((item.get_global_id()[0] == 6) &&
      (item.get_global_id()[1] == 1) &&
      (item.get_global_id()[2] == 0)) {
    intel::sub_group sg =  item.get_sub_group();
    kernelout << "global range        "
    << item.get_global_range() << cl::sycl::endl;
    kernelout << "global id          "
    << item.get_global_id() <<  cl::sycl::endl;
    kernelout << "global linear id   "
    << item.get_global_linear_id() << cl::sycl::endl;
    kernelout << "group range        "
    << item.get_group_range() << cl::sycl::endl;
    kernelout << "group              "
    << item.get_group().get_id() << cl::sycl::endl;
    kernelout << "group linear id    "
    << item.get_group_linear_id() << cl::sycl::endl;
    kernelout << "local range        "
    << item.get_local_range() << cl::sycl::endl;
    kernelout << "local id           "
    << item.get_local_id() << cl::sycl::endl;
    kernelout << "local linear id    "
    << item.get_local_linear_id() << cl::sycl::endl << cl::sycl::endl;
    kernelout << "subgroup group range         "
    << sg.get_group_range() << cl::sycl::endl;
    kernelout << "subgroup group id            "
    << sg.get_group_id() << cl::sycl::endl;
    kernelout << "subgroup local range         "
    << sg.get_local_range() << cl::sycl::endl;
```

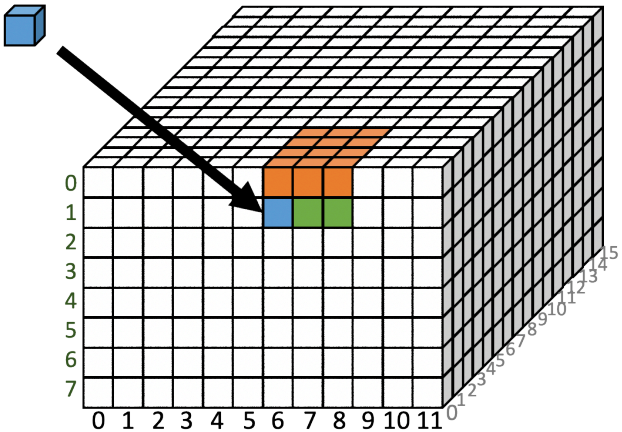
- explore12.cpp
- dump information on {6,1,0}

ND_RANGE - EXPLORE12 1



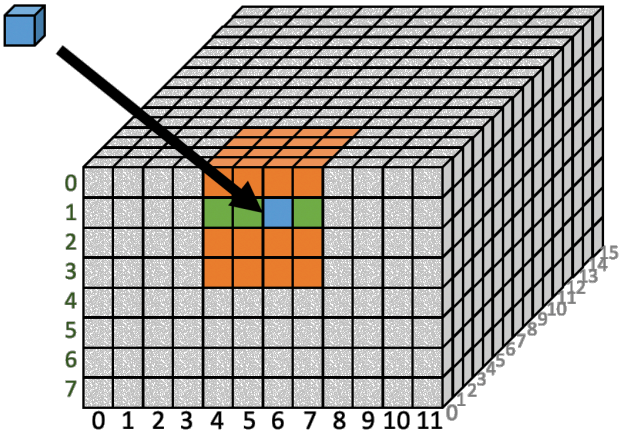

```
$ ./explore12 1
nd_range<3>({12,8,16}{6,2,4})
global range          {12, 8, 16}
global id             {6, 1, 0}
global linear id      784
group range           {2, 4, 4}
group                 {1, 0, 0}
group linear id       16
local range           {6, 2, 4}
local id              {0, 1, 0}
local linear id       4

subgroup group range           3
subgroup group id              {0}
subgroup local range           {16}
subgroup local id              {6}
subgroup uniform group range 3
subgroup max local range       {16}
```

ND_RANGE - EXPLORE12 2




```
$ ./explore12 2
nd_range<3>({12,8,16}{3,2,4})
global range          {12, 8, 16}
global id             {6, 1, 0}
global linear id      784
group range           {4, 4, 4}
group                 {2, 0, 0}
group linear id       32
local range           {3, 2, 4}
local id              {0, 1, 0}
local linear id       4

subgroup group range           2
subgroup group id              {0}
subgroup local range           {16}
subgroup local id              {3}
subgroup uniform group range   2
subgroup max local range       {16}
```

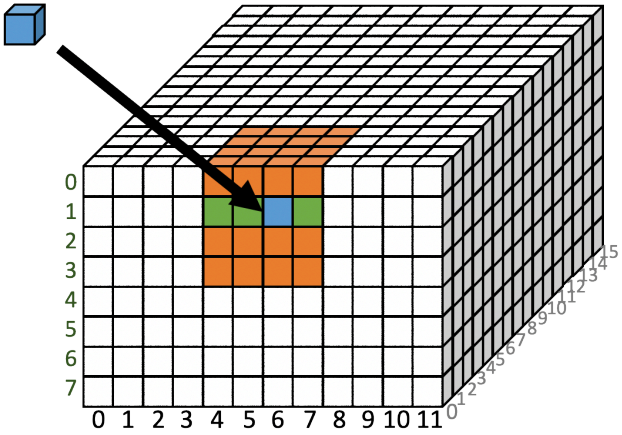
ND_RANGE - EXPLORE12 3




```
$ ./explore12 3
nd_range<3>({12,8,16}{4,4,4})
global range          {12, 8, 16}
global id             {6, 1, 0}
global linear id      784
group range           {3, 2, 4}
group                 {1, 0, 0}
group linear id       8
local range           {4, 4, 4}
local id              {2, 1, 0}
local linear id       36

subgroup group range           4
subgroup group id              {0}
subgroup local range           {16}
subgroup local id              {6}
subgroup uniform group range   4
subgroup max local range       {16}
```

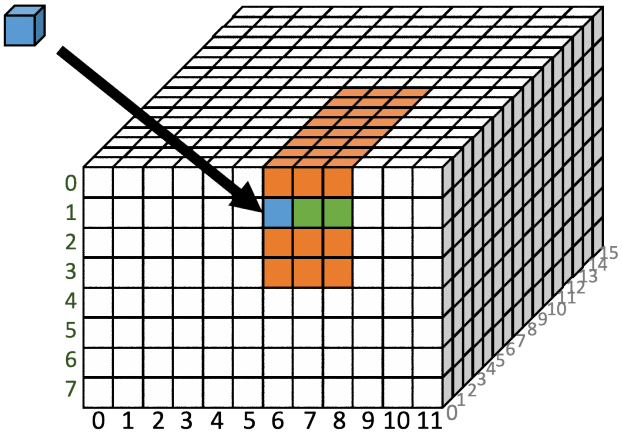
ND_RANGE - EXPLORE12 3

The **group range** multiplied by the **local range** gives the **global range**.



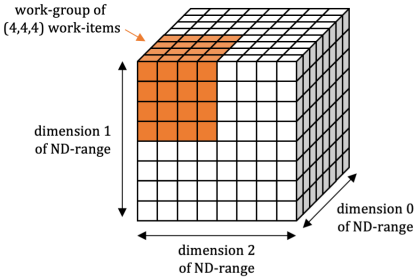

```
$ ./explore12 3
nd_range<3>({12,8,16}{4,4,4})
global range          {12, 8, 16}
global id             {6, 1, 0}
global linear id      784
group range           {3, 2, 4}
group                 {1, 0, 0}
group linear id       8
local range           {4, 4, 4}
local id              {2, 1, 0}
local linear id       36

subgroup group range            4
subgroup group id               {0}
subgroup local range            {16}
subgroup local id               {6}
subgroup uniform group range    4
subgroup max local range        {16}
```

ND_RANGE - EXPLORE12 4

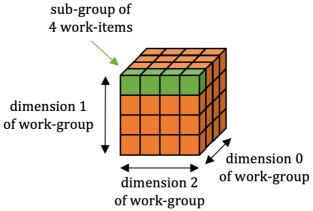



```
$ ./explore12 4
nd_range<3>({12,8,16}{3,4,8})
global range        {12, 8, 16}
global id           {6, 1, 0}
global linear id    784
group range         {4, 2, 2}
group               {2, 0, 0}
group linear id     8
local range         {3, 4, 8}
local id            {0, 1, 0}
local linear id     8

subgroup group range          6
subgroup group id             {0}
subgroup local range          {16}
subgroup local id             {3}
subgroup uniform group range  6
subgroup max local range      {16}
```

ND_RANGE - EXECUTION, DATA, THINKING



ND-Range

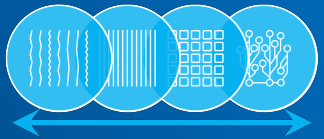


Work-group

Sub-group

Work-item

§5. LAB EXERCISE: VADD ON VARIOUS DEVICES



1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

LAB EXERCISE: VADD ON VARIOUS DEVICES

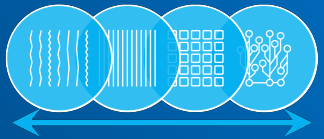
- Follow the directions in the Lab-VADD subdirectory of the module03 directory.
- The instructions will help you use a Jupyter Notebook interface to DevCloud to learn from the VADD example.

PROGRAMMING IN DPC++

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

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

§6. HOST/ACCELERATOR MODEL



1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

WHAT RUNS ON DEVICES?

Kernel functions are executed as work-items, on devices.

- Like a thread, yet very different from a C++ thread
- A work-item cannot synchronize with another work-item (achieve by kernel must end, submit another kernel invocation). Wait - sub-groups and work-groups offer synchronizations.
  - *could* be a OS thread
  - but it *could* be done on a GPU element
  - *or* it could be processed in an FPGA
  - *or* it could be processed in a DSP
  - *or* it could be processed in an AI accelerator

Work-item

Such flexibility for target, brings some restrictions and responsibilities.

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:

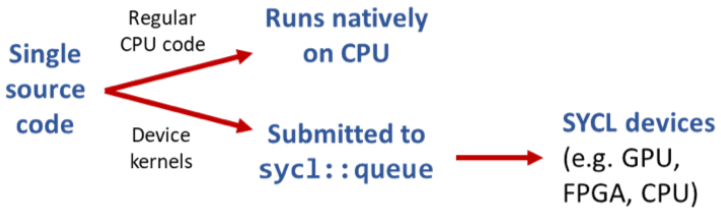
- dynamic polymorphism
- dynamic memory allocations
- static variables
- function pointers
- pointer structure members
- runtime type information
- exception handling

SINGLE SOURCE

```
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]);
```

- All code runs on the host, except *kernel scope* code
- *kernel scope lines 11-16*

SINGLE SOURCE



FIVE METHODS TO STEER WORK TO A DEVICE

- Method#1: Just run on any device
- Method#2: host device fordev & dbg
- Method#3: use accelerator (e.g., GPU)
- Method#4: multiple devices
- Method#5: very specific (custom)

Debugging using Method #2 is an attractive option before moving on to Methods 3+.

METHOD#1: JUST RUN ON ANY DEVICE

```
1  queue myQueue;
2
3  // equivalent to:
4  queue myQueue( default_selector );
5
6  // either one lets the implementation choose
7  // that will be the host if there is no
8  //   accelerator available when the program runs
9  // if there is an accelerator, the implementation will pick
10 //   there is no standard on what it will pick
```

Method#1

- Running device code but not caring which device runs the code.
- Potentially a first step in application development, because it requires the least code.

METHOD#2: HOST DEVICE FORDEV & DBG

```
1   queue myQueue( host_selector );
```

Method#2

- Explicitly run device code on the host device.
- Typical used for debugging and is guaranteed to be always available on any system.
- explore5.cpp showed how this can mask memory move issues, a pro and a con

METHOD#3: USE ACCELERATOR (E.G., GPU)

```
1   queue myQueue( default_selector );
2   queue myQueue( host_selector );
3   queue myQueue( cpu_selector );
4   queue myQueue( gpu_selector );
5   queue myQueue( accelerator_selector );
6
7   // DPC++ only...
8   #include "CL/sycl/intel/fpga_extensions.hpp"
9   queue myQueue( intel::fpga_selector );
10
11  // When a queue is created, if no device matching the
12  // requested type exists, then the selector throws
13  // a sycl::runtime_error exception.
14  //
15  // Yes, I said we'd want try/catch in our real programs.
16  // We'll get to that in a future training module!
```

Method#3
Explicitly dispatching device code to:

- default (Method #1)
- host (Method #2)
- CPU (a device that identifies itself as a CPU)
- GPU (a device that identifies itself as a GPU)
- accelerator (a device that identifies itself as an accelerator, includes FPGAs)
- DPC++ has an extension to specifically request an FPGA

VERYCURIOUS.CPP - FROM MODULE 2

```
// available with: wget tinyurl.com/oneapimodule?2 -O 2.tz
  #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: "
                << OneDevice.get_info<cl::sycl::info::device::name>()
                << std::endl;
    }
    std::cout << std::endl;
  }
}
```

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'
```

METHOD#4: MULTIPLE DEVICES

```
1  queue myGpuQueue( gpu_selector{} );
2  queue myFpgaQueue( intel::fpga_selector{} );
```

Method#4

- Dispatching device code to a heterogeneous set of devices, such as a GPU *and* an FPGA.
- Each device needs its own queue.
- A single queue can never dispatch to multiple devices.

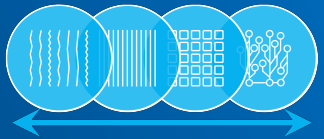
METHOD#5: VERY SPECIFIC (CUSTOM)

```
1   // Defining operator in a class derived from
2   // sycl::device_selector is all that is required
3   // to define any complexity of device selection logic.
4   //
5   // Here is an example from Chapter 2 in the DPC++ book...
6   // demonstrating creating arria_selector to select
7   // an Intel Arria FPGA.
8  class arria_selector : public device_selector {
9    public:
10     virtual int operator()(const device &dev) const {
11       if (
12           dev.get_info<info::device::name>().find("Arria")
13                 != std::string::npos &&
14           dev.get_info<info::device::vendor>().find("Intel")
15                 != std::string::npos) {
16         return 1;
17       } else {
18         return -1;
19       }
20     }
21 };
```

Method#5

- Selecting specific devices from a more general class of devices, such as a specific type of FPGA from a collection of available FPGA devices.

§7. LAB EXERCISE: STENCIL

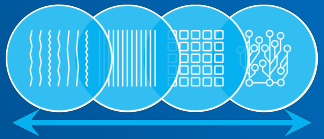


1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

LAB EXERCISE: STENCIL

- Follow the directions in the Lab-Stencil subdirectory of the module03 directory.
- We have a small stencil example from Intel's oneAPI example codes, as a real-world application from which to learn.
- The instructions will help you use a Jupyter Notebook interface to DevCloud to learn from the stencil example.

§8. MODULE 3 DRAWS TO A CLOSE

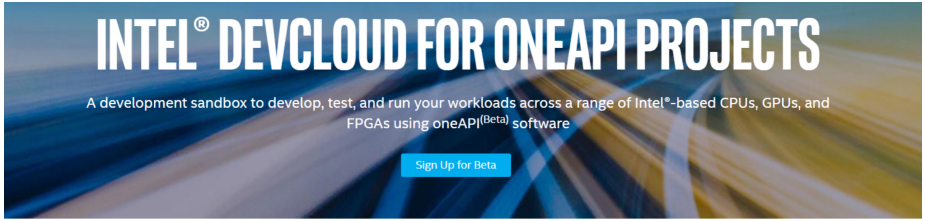


1. DPC++ Programs
2. Execution Model
3. Where and how to get and use DPC++, etc.
4. id, item, nd_item
5. Lab exercise: VADD on Various Devices
6. Host/Accelerator Model
7. Lab exercise: Stencil
8. Module 3 draws to a close

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?3



What You Can Do

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

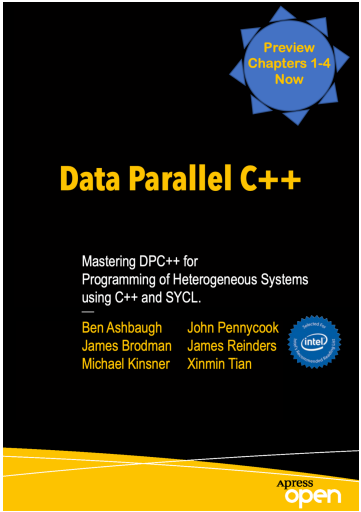
https://software.intel.com/en-us/devcloud/oneapi

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?3

NEXT FOR YOU

- Do the LAB exercises - cool learning
- Module 4 will discuss:
  - Hierarchical Parallelism
  - Data Management (buffers, USM, synchronization, DAGs)
  - Launching Kernels with dependencies (DAGs, queues, etc.)
- Watch prior modules if you skipped them!