



**EDUCACIÓN
EN LÍNEA**



Computación Paralela y Distribuida

DPC++

Integrantes:

- Anderson Lucero
- Diego Vaca
- Cristian Quilumbaquin

DPC++

¿Qué es Data Parallel C++?

- Data Parallel C++? = C++ y SYCL * estándar y extensiones.

Basado en C++ moderno

- Beneficios de productividad de C++ y construcciones familiares.

Arquitectura cruzada basada en estándares

- Incorpora el estándar SYCL para paralelismo de datos y programación heterogénea

DPC++

¿Qué hace Parallel C ++?

- El nuevo lenguaje, que forma parte de la iniciativa oneAPI, proporciona una alternativa abierta e intersectorial a los lenguajes propietarios de arquitectura única. Basado en el conocido C ++ e incorporando SYCL de The Khronos Group, DPC ++ permite a los desarrolladores migrar más fácilmente el código a una variedad de arquitecturas desde la base de código de una aplicación existente.
- Pero con esa capacidad vienen consideraciones únicas, como que los datos deben estar disponibles en el lado del dispositivo y la necesidad de puntos de sincronización entre los núcleos de cómputo que se ejecutan en un host y los dispositivos para garantizar resultados precisos y un comportamiento determinista.

DPC++

- SYCL

SYCL es una capa de abstracción multiplataforma libre de regalías que se basa en la portabilidad y la eficiencia de OpenCL que permite que el código para procesadores heterogéneos se escriba en un estilo de "fuente única" utilizando completamente estándar C ++ . SYCL permite el desarrollo de una sola fuente donde las funciones de plantilla de C ++ pueden contener tanto código de dispositivo como de host para construir algoritmos complejos que usan la aceleración OpenCL y luego reutilizarlos en todo su código fuente en diferentes tipos de datos.

DPC++

DPC++ Extends SYCL

Mejor Productividad

- Las cosas simples deben ser simples de expresar
- Reducir la verbosidad y la carga del programador

Mejor Performance

- Da a los programadores control sobre la ejecución del programa
- Habilitar funciones específicas de hardware

DPC++

DPC ++: colaboración abierta de rápido movimiento que se incorpora al estándar SYCL

- Implementación de código abierto LLVM .
- LLVM es una infraestructura para desarrollar compiladores, escrita a su vez en el lenguaje de programación C++, que está diseñada para optimizar el tiempo de compilación, el tiempo de enlazado, el tiempo de ejecución y el "tiempo ocioso" en cualquier lenguaje de programación que el usuario quiera definir.
- Las extensiones DPC ++ tienen como objetivo convertirse en extensiones principales de SYCL

DPC++

Un programa completo de DPC ++

- Única fuente
El código de host y los núcleos aceleradores heterogéneos se pueden mezclar en los mismos archivos fuente
- C ++ familiar
Las construcciones de la biblioteca agregan funcionalidades, como:

Construct	Propósito
queue	Orientación laboral
buffer	Gestión de datos
parallel_for	paralelismo

ed

Host
code

Accelerator
device code

Host
code

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

    queue{}.submit([&](handler& h) {
        auto out =
            A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

    auto result =
        A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```


DPC++

- Compilar un programa DPC ++

Se puede utilizar el compilador Intel DPC ++

- `dpcpp -fsycl-unnamed-lambda my_source.cpp -o executable`

DPC++

Los núcleos se pueden especificar de varias formas:

- Lambdas de C++
- Objetos de función
- Interoperabilidad

Los núcleos se envían a las colas:

- `parallel_for`
- `single_task`
- `parallel_for_work_group`

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

    queue{}.submit([&](handler& h) {
        auto out = A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

    auto result = A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```

Definición lambda de kernel



DPC++

Núcleo

- Código que se ejecuta en un acelerador, normalmente muchas veces.

Kernels claramente identificables en código

- Un pequeño número de clases puede definir un núcleo (por ejemplo, `parallel_for`)

El desarrollador especifica dónde los núcleos se ejecutarán

- Diversos niveles de control

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

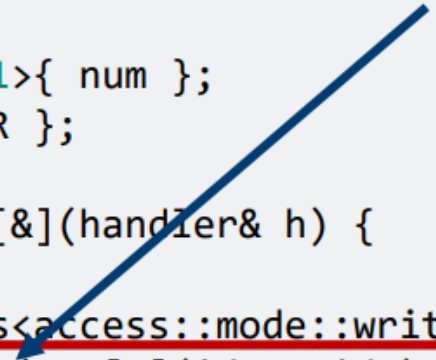
int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

    queue{}.submit([&](handler& h) {
        auto out =
            A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

    auto result =
        A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```

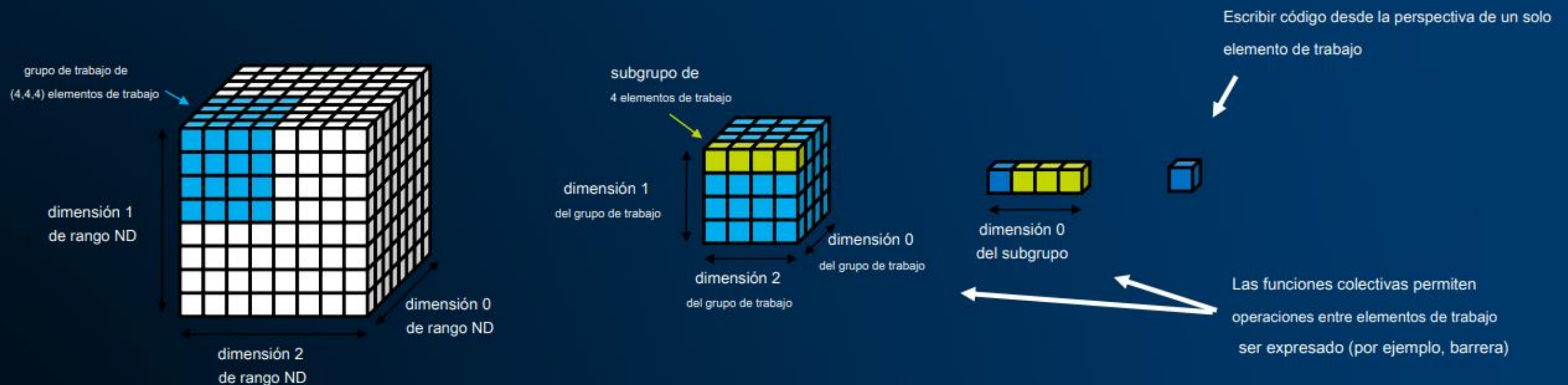
Define núcleo



DPC++

ExecutionModel en Kernels

- El paralelismo de datos se expresa mediante ND-Ranges
- Trabajo total = # grupos de trabajo x # elementos de trabajo por grupo de trabajo
- Modelo jerárquico de datos múltiples de programa único (SPMD) ascendente
- PMD es una técnica empleada para lograr paralelismo; considerado una subcategoría de MIMD. Las tareas son separadas y ejecutadas simultáneamente en múltiples procesadores con diferentes entradas para obtener los resultados con mayor rapidez

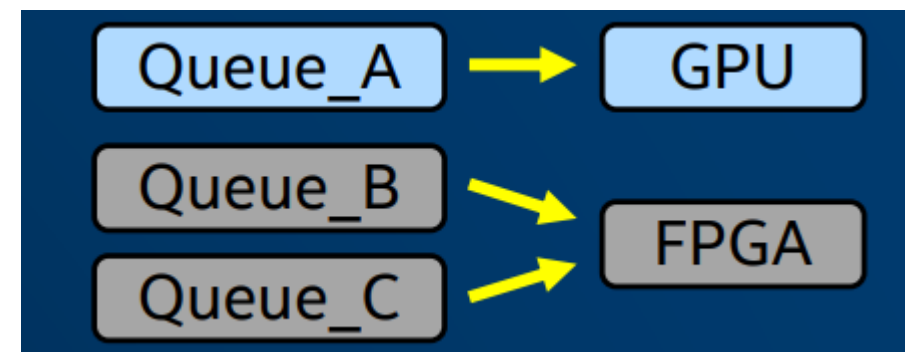


DPC++

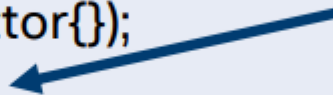
Elegir dónde se ejecutan los kernels del dispositivo

El trabajo se envía a las colas

- Cada cola está asociada con exactamente un dispositivo (por ejemplo, una GPU o FPGA específica)
- Usted puede:
 - ❖ Decidir con qué dispositivo está asociada una cola (si lo desea)
 - ❖ Tener tantas colas como desee para despachar trabajo en sistemas heterogéneos



DPC++

Cree una cola dirigida a cualquier dispositivo:	<code>queue();</code>
Cree una cola dirigida a clases de dispositivos preconfiguradas:	<code>queue(cpu_selector{});</code> <code>queue(gpu_selector{});</code> <code>queue(intel::fpga_selector{});</code> <code>queue(accelerator_selector{});</code> <code>queue(host_selector{});</code>  Always available
Cree una cola para un dispositivo específico (criterios personalizados):	<code>class custom_selector : public device_selector {</code> <code>int operator()(..... // Any logic you want!</code> <code>...</code> <code>queue(custom_selector{});</code>

DPC++

```

1  #include <CL/sycl.hpp>
2  #include <iostream>
3  #include <iomanip>
4
5  const int N = 6;
6  const int M = 2;
7
8  using namespace sycl;
9
10 int main() {
11     queue q;
12     buffer<int,2> buf(range<2>(N,N));
13
14     q.submit([&](handler &h){
15         auto bufacc = buf.get_access<access::mode::read_write>(h);
16         h.parallel_for(nd_range<2>(range<2>(N,N), range<2>(M,M)),
17             [=](nd_item<2> item){
18                 int i = item.get_global_id(0);
19                 int j = item.get_global_id(1);
20                 bufacc[i][j] = i + j;
21             });
22     });
23
24     auto bufacc1 = buf.get_access<access::mode::read>();
25     for(int i = 0; i < N; i++){
26         for(int j = 0; j < M; j++){
27             std::cout << std::setw(10) << bufacc1[i][j] << " ";
28             std::cout << "\n";
29         }
30     }
31     return 0;
32 }
```

With the following output:

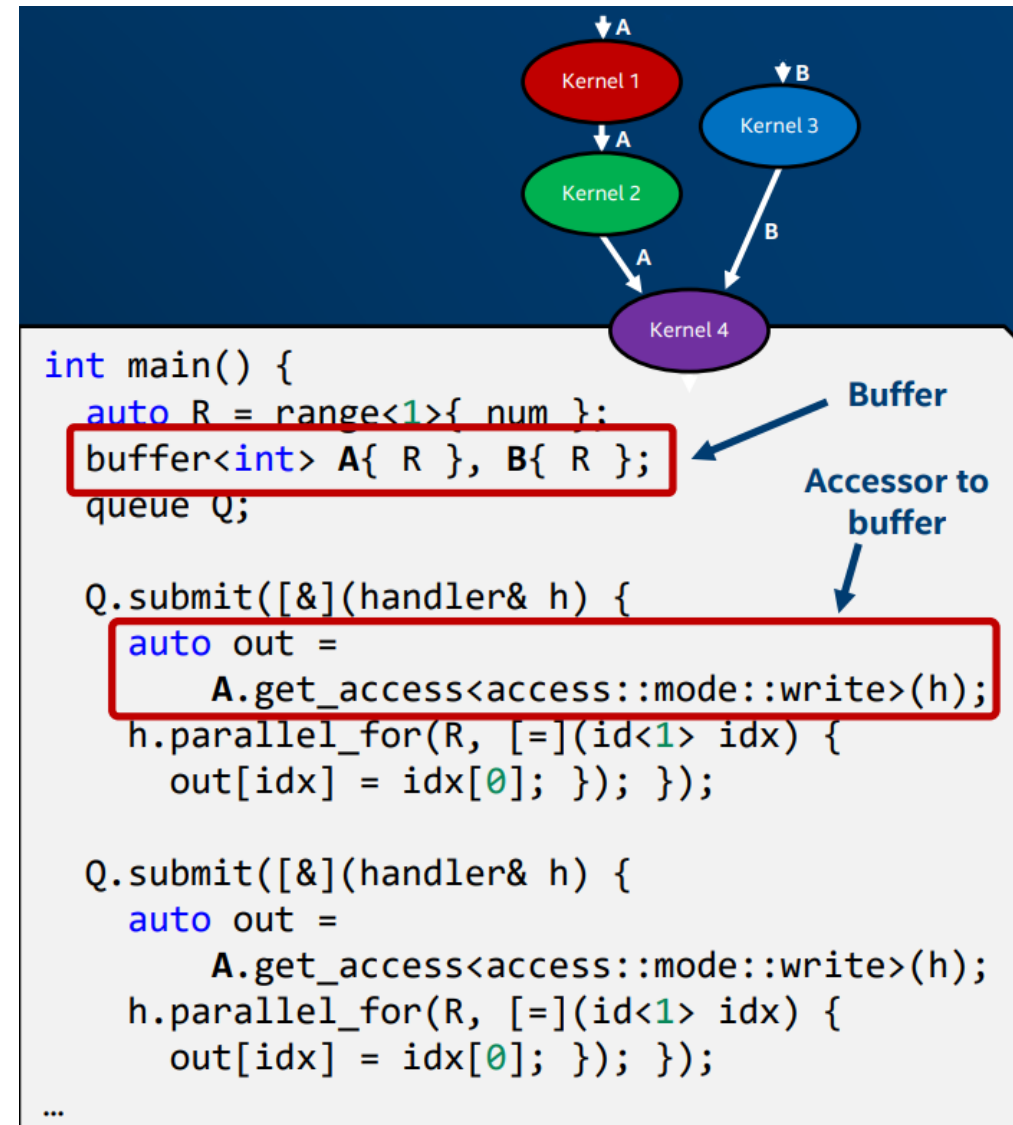
0	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9

DPC++

El modelo de búfer

Buffers: Encapsular datos en una aplicación SYCL, En ambos dispositivos y host

Accessors: Mecanismo para acceder a los datos del búfer, Crea dependencias de datos en el gráfico SYCL que ordenan las ejecuciones del kernel



DPC++

Ejecución asincrónica

Piense en una aplicación SYCL como dos partes:

1. Código de host
2. El gráfico de las ejecuciones del kernel

Estos se ejecutan de forma independiente, excepto en las operaciones de sincronización.

- El código de host envía el trabajo para construir el gráfico (y puede realizar el trabajo de cálculo por sí mismo)
- El gráfico de ejecuciones del kernel y movimientos de datos se ejecuta de forma asincrónica desde el código de host, administrado por el tiempo de ejecución de SYCL.

DPC++

Anfitrión

Código de host
ejecución



```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
    auto R = range<1>{ num };
    buffer<int> A{ R };

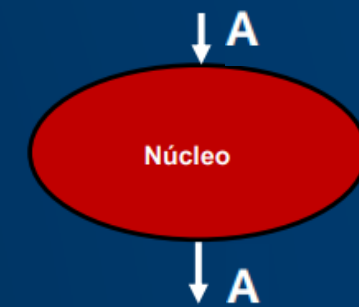
    queue{}.submit([&](handler& h) {
        auto out = A.get_access<access::mode::write>(h);
        h.parallel_for(R, [=](id<1> idx) {
            out[idx] = idx[0]; }); });

    auto result = A.get_access<access::mode::read>();
    for (int i=0; i<num; ++i)
        std::cout << result[i] << "\n";

    return 0;
}
```

Grafico

Graph se ejecuta
asincrónicamente
para albergar el programa



DPC++

- A medida que la tecnología informática evoluciona a un ritmo cada vez más acelerado, también lo hace la dependencia mundial del hardware informático que es lo suficientemente diverso como para manejar cargas de trabajo expansivas centradas en datos.
- Según Bill Savage, vicepresidente y gerente general de Compute Performance and Developer Products en Intel, es el enfoque preciso de oneAPI, una iniciativa liderada por Intel que simplifica la programación de diversas arquitecturas (CPU, GPU, FPGA, aceleradores de IA) para cumplir necesidades de carga de trabajo del cliente.



¡GRACIAS!

**TRAS
CENDE
MOS**

