
OneAPI de Intel:

**Integración y Aplicación en Arquitecturas de Computadoras
Modernas**



**Universidad Champagnat
Arquitecturas de las Computadoras II**

Natalia Anahí Amaya

Introducción

Intel oneAPI es una iniciativa de Intel que proporciona un conjunto de herramientas unificadas y abiertas diseñadas para desarrollar aplicaciones que puedan aprovechar el poder de múltiples arquitecturas de hardware, como CPUs, GPUs, FPGAs y otros aceleradores. El objetivo principal de oneAPI es simplificar el desarrollo de software para múltiples tipos de procesadores sin necesidad de escribir código específico para cada tipo de arquitectura.

La visión de oneAPI

OneAPI proporciona un conjunto completo de bibliotecas, repositorios de código abierto, extensiones de lenguaje C++ basadas en SYCL* e implementaciones de referencia optimizadas para acelerar los siguientes objetivos:

- Definir una plataforma de software multiarquitectura, abierta, unificada y común de múltiples proveedores.
- Garantizar la portabilidad del código funcional y la portabilidad del rendimiento entre proveedores de hardware y tecnologías de aceleradores, ya sea que se basen en CPU, GPU, NPU, ASIC, FPGA u otros aceleradores personalizados.
- Mantener y nutrir un conjunto integral de API de biblioteca para cubrir las necesidades del dominio de programación en todos los sectores y casos de uso de aplicaciones:
 - industrial, transporte, aviación, ciencias de la salud, investigación científica, gestión de emergencias, finanzas, análisis de datos, comunicaciones y más.
 - Inteligencia artificial (IA), aprendizaje automático (ML), mantenimiento predictivo, simulación, análisis de datos, procesamiento de imágenes, matemáticas numéricas y solucionadores de ecuaciones, y más.
- Proporcionar una comunidad de desarrolladores y un foro abierto para impulsar una funcionalidad API unificada e interfaces que satisfagan las

necesidades de un modelo de desarrollo de software multiarquitectura unificado en esta era de computación de descarga acelerada y amplia adopción de computación de descarga para cargas de trabajo de IA en muchos dominios.

- Fomente la colaboración en proyectos oneAPI e implementaciones oneAPI compatibles en todo el ecosistema de desarrolladores.



Características clave de Intel oneAPI:

Arquitectura abierta: No se limita al hardware de Intel. Aunque está optimizado para los procesadores de Intel, se puede utilizar con diferentes arquitecturas.

Lenguaje de programación DPC++ (Data Parallel C++): Extiende C++ para permitir la programación paralela y de alto rendimiento, facilitando el desarrollo de software para arquitecturas heterogéneas.

Conjunto de bibliotecas: Incluye bibliotecas optimizadas para computación científica, inteligencia artificial, análisis de datos y más, para aprovechar al máximo el hardware.

Compatibilidad con varios entornos: oneAPI incluye herramientas para la creación de software en entornos tanto locales como en la nube, brindando flexibilidad en su implementación.

El modelo de programación oneAPI simplifica la programación de CPU y aceleradores mediante las características modernas de C++ para expresar el paralelismo mediante SYCL*. SYCL permite la reutilización de código para el host (como una CPU) y los aceleradores (como una GPU) mediante un único lenguaje fuente, con dependencias de ejecución y memoria claramente comunicadas. El mapeo dentro del código SYCL se puede utilizar para realizar la transición de la aplicación para que se ejecute en el hardware, o conjunto de hardware, que mejor acelere la carga de trabajo. Hay un host disponible para simplificar el desarrollo y la depuración del código del dispositivo, incluso en plataformas que no tienen un acelerador disponible.

oneAPI también admite la programación en CPU y aceleradores utilizando la función de descarga OpenMP* con código C/C++ o Fortran existente.

El objetivo de Intel oneAPI es reducir la complejidad de la programación heterogénea, ofreciendo una solución que permita el desarrollo eficiente en múltiples plataformas de hardware sin la necesidad de recodificar o ajustar aplicaciones para cada tipo de procesador.

Modelo de programación oneAPI

En la computación heterogénea, el procesador host aprovecha los dispositivos aceleradores para ejecutar el código de manera más eficiente.

El modelo de programación oneAPI admite dos métodos portátiles importantes de computación heterogénea: Data Parallel C++ con SYCL* y OpenMP* para C, C++ y Fortran.

Paralelismo de datos en C++ usando SYCL*

El soporte abierto, multiproveedor y multiarquitectura para la programación paralela de datos productiva en C++ se logra a través de C++ estándar con soporte para SYCL.

SYCL es una capa de abstracción multiplataforma que permite escribir código para procesadores heterogéneos utilizando el estándar ISO C++ con el código

de host y kernel para una aplicación contenida en el mismo archivo fuente. El proyecto de código abierto DPC++ está agregando compatibilidad con SYCL al compilador LLVM C++. El compilador Intel® oneAPI DPC++/C++ está disponible como parte del kit de herramientas básico Intel oneAPI.

Modelo de programación de descarga de C/C++ o Fortran con OpenMP*

El compilador Intel® oneAPI DPC++/C++ y el compilador Intel® Fortran permiten a los desarrolladores de software utilizar directivas OpenMP* para descargar trabajo a los aceleradores Intel y mejorar el rendimiento de las aplicaciones.

OpenMP ha sido un lenguaje de programación estándar durante más de 20 años e Intel implementa la versión 5 del estándar OpenMP. El compilador Intel oneAPI DPC++/C++ con soporte para descarga de OpenMP está disponible como parte del kit de herramientas base Intel oneAPI y del kit de herramientas Intel® HPC. El compilador Intel® Fortran Classic y el compilador Intel® Fortran con soporte para descarga de OpenMP están disponibles como parte del kit de herramientas Intel® HPC.

Usos actuales de oneAPI

OneAPI se está utilizando en diversas industrias y casos de uso donde se requiere computación de alto rendimiento y heterogénea. Gracias a su portabilidad y optimización para múltiples arquitecturas (CPUs, GPUs, FPGAs, etc.), su aplicación se extiende a campos como inteligencia artificial, análisis de datos, simulaciones científicas y más. A continuación, se presentan los usos actuales más relevantes:

1. Inteligencia Artificial y Aprendizaje Automático

- **Entrenamiento de modelos:** Aprovecha la aceleración en GPUs y CPUs para entrenar redes neuronales profundas (DL) con bibliotecas como oneDNN y oneMKL.

- **Inferencia optimizada:** Reduce la latencia en aplicaciones como visión por computadora y procesamiento de lenguaje natural (NLP).
- **Análisis de datos masivos:** Herramientas como oneDAL (Data Analytics Library) facilitan el desarrollo de sistemas de clasificación, regresión y clustering.
- **Ejemplo:** Empresas como Bosch utilizan OneAPI para acelerar el entrenamiento e inferencia de modelos en plataformas heterogéneas.

2. Simulación Científica

- **Dinámica de fluidos computacional (CFD):** Simula el flujo de líquidos y gases con precisión en aplicaciones como diseño aerodinámico y predicción del clima.
- **Simulaciones moleculares:** Acelera cálculos en química cuántica y diseño de fármacos.
- **Astronomía y astrofísica:** Procesamiento masivo de datos para análisis de señales y simulaciones cósmicas.
- **Ejemplo:** Instituciones como NASA emplean OneAPI para simulaciones espaciales en GPUs de alto rendimiento.

3. Computación de Alto Rendimiento (HPC)

- **Optimización de software para supercomputadoras:** Permite escalar aplicaciones para sistemas heterogéneos como los supercomputadores basados en CPUs y GPUs.
- **Resolución de ecuaciones complejas:** Utiliza SYCL para implementar algoritmos en paralelo, mejorando el tiempo de cálculo.
- **Simulaciones industriales:** Modelos para ingeniería estructural, análisis térmico y diseño de materiales.
- **Ejemplo:** Argonne National Laboratory utiliza OneAPI para simulaciones moleculares a gran escala.

4. Finanzas y Negocios

- **Análisis de riesgos financieros:** Implementa algoritmos complejos en paralelo para evaluar escenarios y riesgos en tiempo real.
- **Predicción de mercados:** Acelera el análisis de grandes volúmenes de datos históricos y su modelado.
- **Optimización de portafolios:** Resuelve problemas matemáticos y financieros complejos con oneMKL.
- **Ejemplo:** Instituciones financieras usan OneAPI para optimizar estrategias de trading y análisis predictivo.

5. Procesamiento Multimedia

- **Codificación y decodificación de video:** Utiliza oneVPL (Video Processing Library) para transcodificación eficiente en plataformas heterogéneas.
- **Edición y renderización:** Acelera tareas intensivas en aplicaciones de edición de video, efectos visuales y producción de contenido.
- **Gaming en la nube:** Optimización de gráficos y simulaciones físicas en GPUs.
- **Ejemplo:** Netflix ha explorado herramientas basadas en OneAPI para transcodificación eficiente de contenido multimedia.

6. Robótica y Automóviles Autónomos

- **Procesamiento de sensores:** Acelera la interpretación de datos provenientes de LiDAR, cámaras y radares.
- **Planeación de rutas:** Resuelve problemas de optimización en sistemas autónomos.
- **Simulación en tiempo real:** Permite simular entornos complejos para pruebas de vehículos autónomos.
- **Ejemplo:** BMW utiliza OneAPI para simulaciones y procesamiento en tiempo real en plataformas heterogéneas.

7. Diseño y Optimización de Hardware

- **Compiladores y arquitecturas personalizadas:** Acelera la simulación y verificación de nuevos diseños de chips.
- **Optimización de arquitecturas:** Prototipa algoritmos que maximizan el rendimiento en FPGAs y ASICs.
- **Modelado y simulación electrónica:** Utiliza cálculos avanzados para optimizar diseños electrónicos.

8. Educación e Investigación Académica

- **Cursos de computación paralela:** Se utiliza en programas educativos para enseñar conceptos de programación heterogénea.
- **Investigación científica:** Acelera experimentos que requieren grandes volúmenes de datos y cálculos, como biología computacional y ciencias de la tierra.
- **Prototipado rápido:** Facilita la implementación de algoritmos en plataformas híbridas para probar ideas.

Conclusión

OneAPI está transformando diversos sectores al permitir que desarrolladores y científicos utilicen múltiples dispositivos de forma eficiente y sin depender de plataformas propietarias. Su flexibilidad lo convierte en una herramienta fundamental para resolver problemas complejos y optimizar aplicaciones en áreas críticas.

Practica

<https://github.com/namaya-nana/proyectoOneApi>

Proyecto OneAPI: Multiplicador de Matrices

Este proyecto implementa y compara el cálculo de la multiplicación de matrices en dos enfoques: **secuencial** y **paralelo utilizando SYCL** de OneAPI. Es un caso práctico ideal para demostrar las capacidades de paralelización en hardware como CPUs, GPUs y FPGAs.

La multiplicación de matrices es un problema computacional clave en varios campos, incluyendo gráficos por computadora, simulaciones científicas, inteligencia artificial y más. El algoritmo en este proyecto toma dos matrices cuadradas **A** y **B** de tamaño **N×N** y calcula una matriz resultado **C** donde cada elemento **C[i][j]** es la suma de productos de las filas de **A** con las columnas de **B**:

$$C[i][j] = \sum_{k=0}^{N-1} A[i][k] \cdot B[k][j]$$

Implicancias del algoritmo

1. Secuencial:

- Realiza los cálculos uno por uno, iterando sobre todos los índices de la matriz.
- Es fácil de entender pero no eficiente para tamaños grandes debido a la limitación del hardware secuencial.

2. Paralelo (SYCL):

- Utiliza programación paralela para distribuir el trabajo entre múltiples hilos en un dispositivo compatible con SYCL (GPU o CPU).
- Reduce significativamente el tiempo de ejecución, especialmente para matrices grandes.
- Aprovecha la arquitectura moderna, optimizando el uso de recursos computacionales.

Usos del algoritmo

1. Inteligencia Artificial (IA):

- Operaciones básicas en redes neuronales, como el cálculo de pesos y convoluciones, involucran multiplicaciones de matrices.

2. Simulaciones Científicas:

- Resolución de ecuaciones lineales, modelado de sistemas físicos y cálculos numéricos complejos.

3. Gráficos por Computadora:

- Transformaciones geométricas como rotaciones y escalado se calculan con multiplicaciones de matrices.

4. Criptografía:

- Algoritmos criptográficos avanzados dependen de operaciones matriciales para generar claves seguras.

5. Procesamiento de Señales:

- Transformaciones y filtros que requieren cálculos intensivos.

Ejecución del proyecto

Flujo del programa

1. Inicialización:

- Dos matrices A y B son inicializadas con valores predefinidos.
- La matriz C almacena los resultados.

2. Multiplicación Secuencial:

- Calcula la matriz C de manera iterativa en la CPU.
- Mide el tiempo de ejecución.

3. Multiplicación Paralela con SYCL:

- Distribuye los cálculos en un dispositivo acelerador.
- Utiliza la abstracción `parallel_for` de SYCL para dividir las iteraciones.
- Sincroniza las operaciones con la CPU.

4. Comparación de Tiempos:

- Evalúa la mejora en rendimiento entre el cálculo secuencial y paralelo.

Resultados esperados

- **Tiempo de ejecución secuencial** será significativamente mayor para matrices grandes.
- **Tiempo de ejecución paralelo** será más eficiente debido al uso de la paralelización.

Visualización de resultados

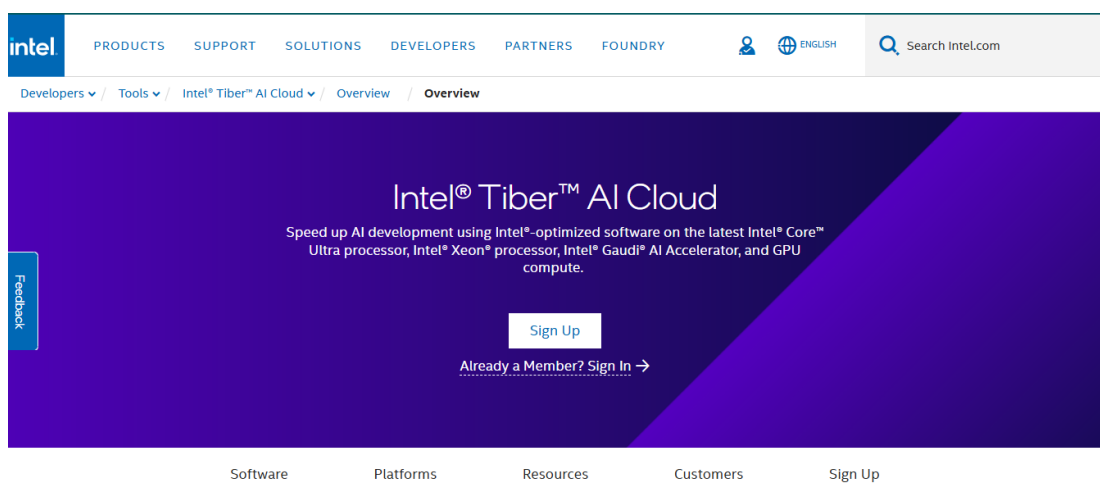
- La consola imprimirá:
 - El dispositivo utilizado (CPU o GPU).
 - Los tiempos de ejecución para las implementaciones secuencial y paralela.
 - Los elementos de la matriz resultante para verificar la exactitud del cálculo.

Guia de Pasos

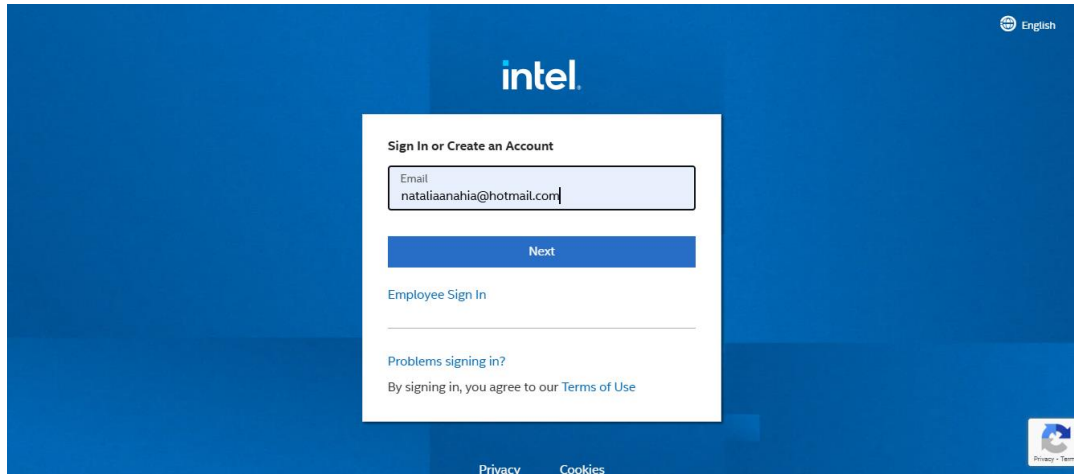
Paso 1: Ingresar a Intel® Tiber™ AI Cloud

<https://www.intel.com/content/www/us/en/developer/tools/tiber/ai-cloud.html>

Luego hacer click en Click Here to Sign In.

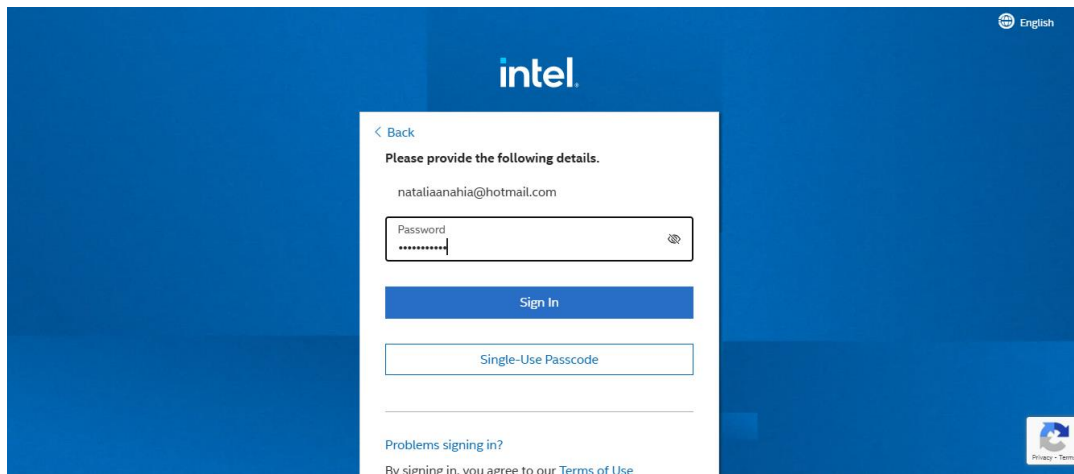


Paso 2: Ingresar el email del usuario registrado. Click en Sign In.



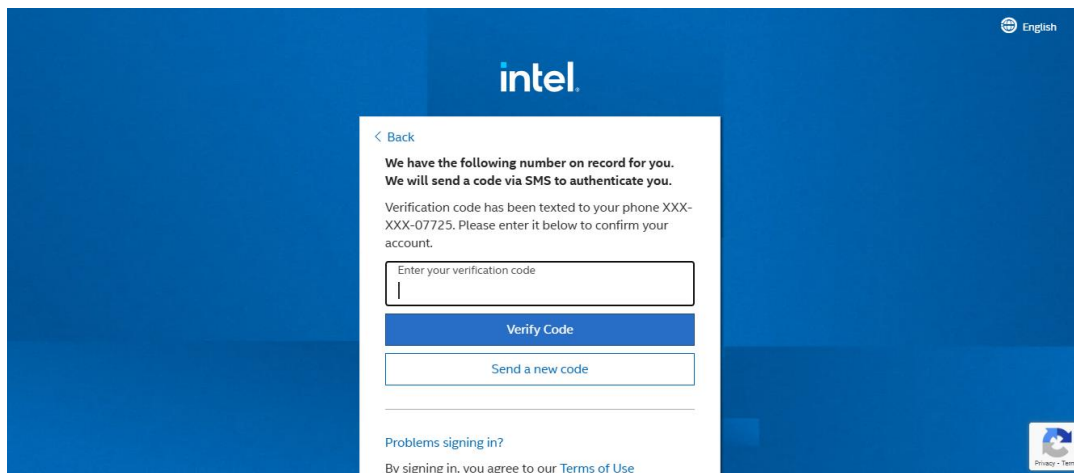
The screenshot shows the Intel sign-in page. At the top, the Intel logo is centered. Below it, a white box contains the text "Sign In or Create an Account". Inside this box, there is an email input field with "nataliaanahia@hotmail.com" entered. Below the input field is a blue "Next" button. Underneath the button, there is a link for "Employee Sign In". At the bottom of the white box, there are links for "Problems signing in?" and "By signing in, you agree to our Terms of Use". The background is a solid blue color. In the top right corner, there is a language selector set to "English". In the bottom right corner, there is a small icon for "Privacy & Terms".

Paso 3: Ingresar la contraseña correspondiente al usuario. Click en Sign In.



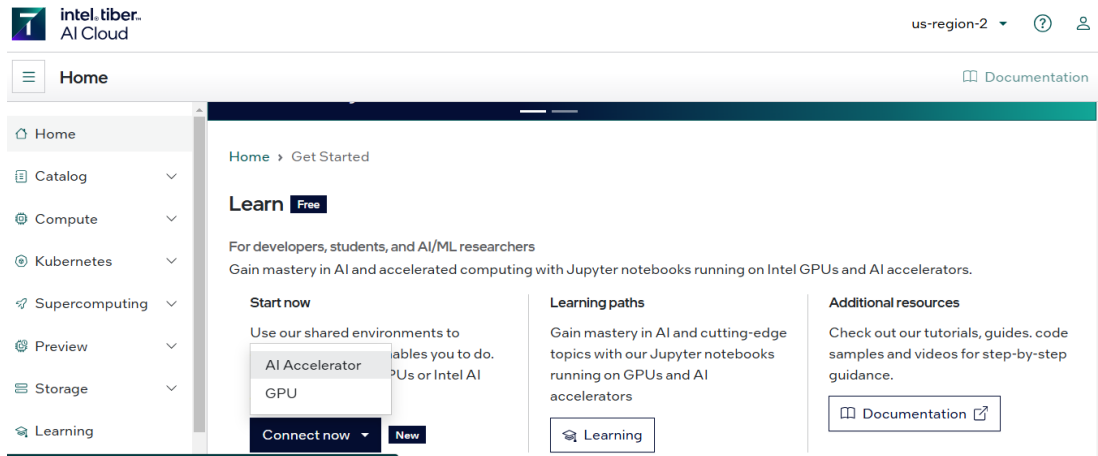
The screenshot shows the Intel sign-in page after the email has been entered. A "< Back" link is at the top left of the white box. The text "Please provide the following details." is followed by the email "nataliaanahia@hotmail.com". Below this is a password input field with "*****" entered. A blue "Sign In" button is below the password field. Underneath the button is a link for "Single-Use Passcode". At the bottom of the white box, there are links for "Problems signing in?" and "By signing in, you agree to our Terms of Use". The background is a solid blue color. In the top right corner, there is a language selector set to "English". In the bottom right corner, there is a small icon for "Privacy & Terms".

Paso 4: Ingresar el código de autenticación. Click en Verify Code.

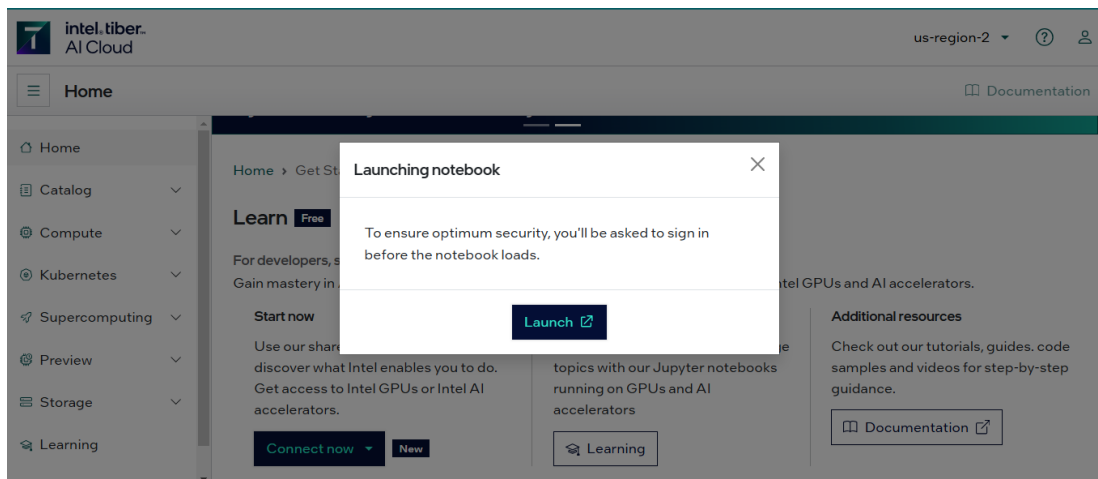


The screenshot shows the Intel verify code screen. A "< Back" link is at the top left of the white box. The text "We have the following number on record for you. We will send a code via SMS to authenticate you." is followed by "Verification code has been texted to your phone XXX-XXX-07725. Please enter it below to confirm your account." Below this is a verification code input field with "Enter your verification code" placeholder text. A blue "Verify Code" button is below the input field. Underneath the button is a link for "Send a new code". At the bottom of the white box, there are links for "Problems signing in?" and "By signing in, you agree to our Terms of Use". The background is a solid blue color. In the top right corner, there is a language selector set to "English". In the bottom right corner, there is a small icon for "Privacy & Terms".

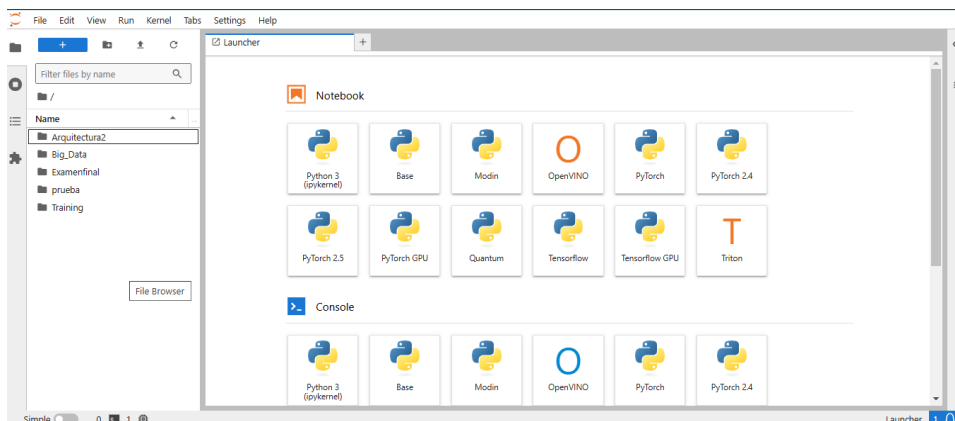
Paso 5: Seleccione el acceso a las GPU de Intel o a los aceleradores de IA de Intel.



Paso 6: Hacer clic en Launch



Paso 7: Una vez ingresado en el laboratorio, buscar la carpeta correspondiente a la materia Arquitectura2



Paso 8: En la carpeta src creamos un archivo denominado gpu_practice.cpp.

En este archivo vamos a mostrar los datos del dispositivo que utilizaremos.

```

1 // Arquitectura2/src/gpu_practice.cpp
2 #include <sycl/sycl.hpp>
3
4 using namespace sycl;
5
6 int main() {
7     /// Create a device queue with device selector
8     try {
9         queue q(gpu_selector_v);
10        std::cout << "Device: " << q.get_device().get_info<info::device::name>() << "\n";
11    } catch (const sycl::exception& e) {
12        std::cerr << "Error: " << e.what() << "\n";
13    }
14    return 0;
15 }
16 |

```

Paso 9: Dentro de la carpeta lab, también crearemos un archivo llamado

gpu_practice.cpp, acá vamos a desarrollar nuestro programa.

```

1 // Arquitectura2/lab/gpu_practice.cpp
2 #include <CL/sycl.hpp>
3 #include <iostream>
4 #include <vector>
5 #include <chrono> /// Para medir tiempos
6
7 using namespace std;
8 using namespace sycl;
9
10 /// Función secuencial de multiplicación de matrices
11 void matrix_multiply_sequential(const std::vector<float>& A, const std::vector<float>& B,
12                                std::vector<float>& C, size_t N) {
13     for (size_t i = 0; i < N; i++) {
14         for (size_t j = 0; j < N; j++) {
15             float sum = 0;
16             for (size_t k = 0; k < N; k++) {
17                 sum += A[i * N + k] * B[k * N + j];
18             }
19             C[i * N + j] = sum;
20         }
21     }
22 }
23

```



```

24 // Función paralela de multiplicación de matrices con SYCL
25 void matrix_multiply_parallel(const std::vector<float>& A, const std::vector<float>& B,
26                             std::vector<float>& C, size_t N, queue& q) {
27     buffer a_buf(A.data(), range<1>(N * N));
28     buffer b_buf(B.data(), range<1>(N * N));
29     buffer c_buf(C.data(), range<1>(N * N));
30
31     q.submit([&](handler& h) {
32         auto a = a_buf.get_access<access::mode::read>(h);
33         auto b = b_buf.get_access<access::mode::read>(h);
34         auto c = c_buf.get_access<access::mode::write>(h);
35
36         h.parallel_for(range<2>(N, N), [=](id<2> idx) {
37             size_t row = idx[0];
38             size_t col = idx[1];
39             float sum = 0;
40
41             for (size_t k = 0; k < N; k++) {
42                 sum += a[row * N + k] * b[k * N + col];
43             }
44             c[row * N + col] = sum;
45         });
46     });
47
48     q.wait(); // Sincroniza las operaciones
49 }

```

```

50
51 // Función para imprimir matrices
52 void print_matrix(const std::vector<float>& matrix, size_t N) {
53     for (size_t i = 0; i < N; i++) {
54         for (size_t j = 0; j < N; j++) {
55             std::cout << matrix[i * N + j] << " ";
56         }
57         std::cout << "\n";
58     }
59 }
60
61 int main() {
62     const size_t N = 4; // Tamaño de la matriz (NxN)
63     std::vector<float> A(N * N, 1.0); // Inicializa con 1s
64     std::vector<float> B(N * N, 2.0); // Inicializa con 2s
65     std::vector<float> C(N * N, 0.0); // Resultado
66
67     // Crear una cola SYCL con el selector predeterminado
68     queue q{sycl::default_selector_v};
69     std::cout << "Usando dispositivo: " << q.get_device().get_info<info::device::name>() << "\n";
70
71     // Medir el tiempo para la multiplicación secuencial
72     auto start = chrono::high_resolution_clock::now();
73     matrix_multiply_sequential(A, B, C, N);
74     auto end = chrono::high_resolution_clock::now();
75     chrono::duration<double> sequential_duration = end - start;
76     cout << "Tiempo de ejecución secuencial: " << sequential_duration.count() << " segundos.\n";

```

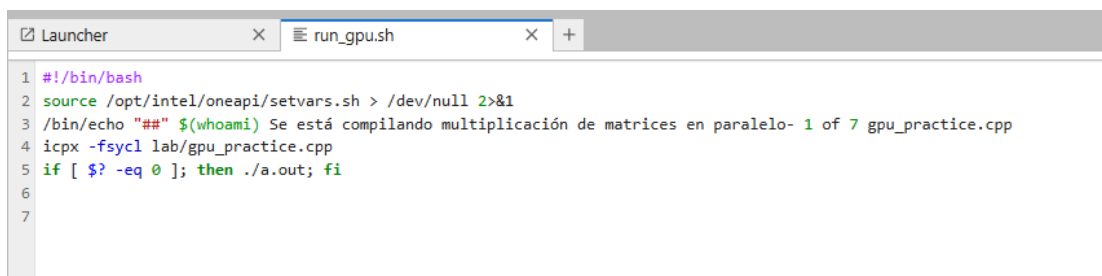


```

77
78 // Imprimir el resultado de la multiplicación secuencial
79 //cout << "Resultado de la multiplicación secuencial:\n";
80 //print_matrix(C, N);
81
82 // Resetear la matriz C para el cálculo paralelo
83 fill(C.begin(), C.end(), 0.0);
84
85 // Medir el tiempo para la multiplicación paralela (con SYCL)
86 start = chrono::high_resolution_clock::now();
87 matrix_multiply_parallel(A, B, C, N, q);
88 end = chrono::high_resolution_clock::now();
89 chrono::duration<double> parallel_duration = end - start;
90 cout << "Tiempo de ejecución paralelo (SYCL): " << parallel_duration.count() << " segundos.\n";
91
92 // Imprimir el resultado de la multiplicación paralela
93 cout << "Resultado de la multiplicación paralela (SYCL):\n";
94 print_matrix(C, N);
95
96 return 0;
97 }
98

```

Paso 10: Crear archivo `run_gpu.sh`. En este archivo vamos a crear un script de bash. Este script configura el entorno de compilación para Intel OneAPI, compila un programa en C++ que utiliza SYCL para programación paralela en GPUs, y luego ejecuta el programa(`gpu_practice.cpp`) si la compilación fue exitosa. Este script está diseñado para ejecutar en un entorno que tiene el compilador Intel OneAPI instalado y configurado.



```

1 #!/bin/bash
2 source /opt/intel/oneapi/setvars.sh > /dev/null 2>&1
3 /bin/echo "##" $(whoami) Se está compilando multiplicación de matrices en paralelo- 1 of 7 gpu_practice.cpp
4 icpx -fsycl lab/gpu_practice.cpp
5 if [ $? -eq 0 ]; then ./a.out; fi
6
7

```

Paso 11: Creamos el archivo `q`. En este archivo se crea un script de bash que se utiliza para enviar un trabajo (job) a la Intel DevCloud, un entorno en línea proporcionado por Intel para el desarrollo y la ejecución de trabajos en sus arquitecturas de hardware, incluidas las GPU.

```

1  #!/bin/bash
2  #=====
3  # Copyright © Intel Corporation
4  #
5  #
6  # SPDX-License-Identifier: MIT
7  #=====
8  # Script to submit job in Intel(R) DevCloud
9  # Version: 0.71
10 #=====
11 if [ -z "$1" ]; then
12     echo "Missing script argument, Usage: ./q run.sh"
13 elif [ ! -f "$1" ]; then
14     echo "File $1 does not exist"
15 else
16     echo "Job has been submitted to Intel(R) DevCloud and will execute soon."
17     echo ""
18     script=$1
19     # Remove old output files
20     rm *.sh.* > /dev/null 2>&1
21     # Submit job using qsub
22     qsub_id=qsub -l nodes=1:gpu:ppn=2 -d . $script
23     job_id=$(cut -d'.' -f1 <<"$qsub_id")
24     # Print qstat output
25     qstat
26     # Wait for output file to be generated and display
27     echo ""
28     echo -ne "Waiting for Output "
29     until [ -f $script.o$job_id ]; do
30         sleep 1
31         echo -ne "■"
32         ((timeout++))
33         # Timeout if no output file generated within 60 seconds
34         if [ $timeout == 70 ]; then
35             echo ""
36             echo ""
37             echo "Timeout 60 seconds: Job is still queued for execution, check for output file later ($script.o$job_id)"
38             echo ""
39             break
40         fi
41     done
42     # Print output and error file content if exist
43     if [ -n "$(find -name "*.sh.o$job_id")" ]; then
44         echo " Done!"
45         cat $script.o$job_id
46         cat $script.e$job_id
47         echo "Job Completed in $timeout seconds."
48         rm *.sh.*$job_id > /dev/null 2>&1
49     fi
50 fi

```

Paso 12: Generamos el archivo practica.ipynb, este archivo será el que nos permitirá ejecutar nuestro programa. Estas sentencias ajustan los permisos de ejecución de dos archivos (q y run_gpu.sh) y luego ejecutan uno u otro dependiendo de la disponibilidad del comando qsub. Si qsub está disponible, usa un script llamado q para ejecutar run_gpu.sh, y si no está disponible, ejecuta directamente run_gpu.sh.

```

[1]: ! chmod 755 q; chmod 755 run_gpu.sh; if [ -x "$(command -v qsub)" ]; then ./q run_gpu.sh; else ./run_gpu.sh; fi

## u0068f821c48e34a98dc6c80a500121f Se está compilando multiplicación de matrices en paralelo- 1 of 7 gpu_practice.cpp
Usando dispositivo: Intel(R) Data Center GPU Max 1100
Tiempo de ejecución secuencial: 2.14e-07 segundos.
Tiempo de ejecución paralelo (SYCL): 0.10158 segundos.
Resultado de la multiplicación paralela (SYCL):
8 8 8 8
8 8 8 8
8 8 8 8
8 8 8 8

```

Conclusión

Este proyecto ilustra cómo **OneAPI** y **SYCL** simplifican la programación paralela para operaciones intensivas como la multiplicación de matrices. Comparar los tiempos de ejecución entre la implementación secuencial y paralela resalta las ventajas de aprovechar hardware moderno en aplicaciones prácticas.

Bibliografia

- oneAPI—What is It? Demystifying oneAPI for Developers (June 20, 2024)
<https://www.intel.com/content/www/us/en/developer/articles/technical/oneapi-what-is-it.html>
- Intel® oneAPI Programming Guide (10/31/2024)
<https://www.intel.com/content/www/us/en/docs/oneapi/programming-guide/2025-0/overview.html>
- Intel oneApi Programming Guide
<https://danysoft.com/estaticos/free/Libros%20en%20formato%20GRATUITO/oneapi-programming-guide.pdf>
- Intel. (2024). oneAPI GPU Optimization Guide.
<https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2024-0/overview.html>