Paralelización de Algoritmos

▼ Primer Parcial

▼ 08-08-2022

Docente: Julio César Ponce Gallegos

Leyendo el programa

▼ 09-08-2022

Programación en paralelo

Hacer varios procesos al mismo tiempo

Normalmente uso de la programación secuencial, ya que dentro de ésta se encuentra el uso de hilos pero se tiene algo distinto a nivel interno

Para distribuir procesos/tareas en varios núcleos

Un objetivo de distribuir las tareas es para disminuir tiempos

Hay que tomar en cuenta tareas que se repiten en especial en ciclos anidados

Analizar si es necesario paralelizar según el problema o la cantidad de información que se va a manejar

▼ 10-08-2022

Programación Paralela

La programación paralela se utiliza para resolver problemas en los que los recursos de una sola máquina no son suficientes. La finalidad de paralelizar un algoritmo es disminuir el tiempo de procesamiento mediante la distribución de tareas entre los procesadores disponibles.

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaiones pueden ocurrir simutáneamente.

Un clásico uso del apralelismo es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran

relevancia, los cuales requieren de una gran capacidad de procesamiento de tiempo y de espacio de memoria, debido a las complejas operaciones que deben realizar en problemas NP-Duros o NP-completos

Otro uso es el de gráficos generados por computadora. La generación de fotogramas requiere de una gran cantidad de cálculos matemáticos. Eso supone una tarea muy compleja para un solo procesador, por lo que es necesario que haya algún tipo de paralelismo, para distribuir la tarea para que esta sea realizada eficiente y eficazmente muchas veces en un tiempo muy pequeño

¿Qué implica?

La computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control-coordinación

La programación paralela se considera un paradigma de programación, razón por la cual no ha dejado de ser necesaria o no ha sido totalmente automatizable, igualmente hay otras razones interesantes detrás para entender la existencia, actualidad y contemporaneidad de la programación paralela.

Clases de computadoras paralelas

- Computación multinúcleo
    - Un procesador multinúcleo es un procesador que incluye múltiples unidades de ejecución (núcleos) en el mismo procesador. Los procesadores superescalares pueden ejecutar múltiples instrucciones por ciclo de un flujo de instrucciones (hilo), a diferencia de este, un procesador multinúcleo puede ejecutar múltiples instrucciones por ciclo de secuencias de instrucciones múltiples. Cada núcleo en un procesador multinúcleo potencialmente puede ser superescalar, es

decir, en cada ciclo, cada núcleo puede ejecutar múltiples instrucciones de un flujo de instrucciones.

    - El multithreading simultáneo - de la cual Intel HyperThreading es el más conocido - es una forma de psudo multinúcleo. Un procesador con capacidad de multithreading simultáneo tiene una sola unidad de ejecución (núcleo), pero cuando esa unidad de ejecución está desocupada - por ejemplo durante un error de caché - se utiliza para procesar un segundo hilo. El microprocesador Cell de IBM es otro prominente procesador multinúcleo.

- Multiprocesamiento simétrico
    - Un procesamiento simétrico (SMP) es un sistema computacional con múltiples procesadores idénticos que comparten memoria y se conectan a través de un bus. La contención del bus previene el escalado de esta arquitectrua. Como resultado, estos no comprenden más de 32 procesadores debido al pequeño tamaño de los procesadores y de la significativa reducción en los requisitos de ancho de banda de bus, tales multiprocesadores simétricos son extremadamente rentables, siempre que exista una cantidad suficiente de ancho de banda.

Un procesador **multinúcleo** es aquel que combina dos o más microprocesadores independientes en un solo paquete, a menudo un solo circuito integrado. Un dispositivo de **doble núcleo** contiene solamente dos microprocesadores independientes. En general, los microprocesadores multinúcleo permiten que un dispositivo computacional exhiba una cierta forma del paralelismo a nivel de thread (*thread-level parallelism*) (**TLP**) sin incluir múltiples microprocesadores en paquetes físicos separados. Esta forma de TLP se conoce a menudo como multiprocesamiento a nivel de chip (*chip-level multiprocessing*) o **CMP**.

Multiprocesamiento es un Procesamiento simultáneo con dos o más procesadores en un computador. Estos procesadores se unen con un canal de alta velocidad y comparten la carga de trabajo general entre ellos. En caso de que uno falle el otro se hace cargo. El multiprocesamiento también se efectúa en computadores de propósitos especiales, como procesadores vectoriales, los cuales proveen procesamiento simultáneo de conjunto de datos. Aunque los computadores se construyen con diversas características que se superponen, como ejecutar instrucciones mientras se ingresan y se sacan datos, el multiprocesamiento se refiere específicamente a la ejecución de instrucciones simultáneas.

La arquitectura SMP (también llamada **UMA**, del inglés ***U**niform **M**emory **A**ccess*, en español "acceso uniforme a memoria") se caracteriza por el hecho de que varias unidades de procesamiento comparten el acceso a la memoria, compitiendo en igualdad de condiciones por dicho acceso, de ahí la denominación "simétrico".

Los sistemas SMP permiten que cualquier procesador trabaje en cualquier tarea sin importar su localización en memoria; con un propicio soporte del sistema operativo, estos sistemas pueden mover fácilmente tareas entre los procesadores para garantizar eficientemente el trabajo.

Una computadora SMP se compone de microprocesadores independientes que se comunican con la memoria a través de un bus compartido. Dicho *bus* es un recurso de uso común. Por tanto, debe ser arbitrado para que solamente un microprocesador lo use en cada instante de tiempo. Si las computadoras con un único microprocesador tienden a gastar considerable tiempo esperando a que lleguen los datos desde la memoria, SMP empeora esta situación, ya que hay varios parados en espera de datos.

▼ 11-08-2022

- Computación en cluster

    - Un clúster es un grupo de ordenadores débilmente acoplados que trabajan en estrecha colaboración de modo que en en algunos aspectos pueden considerarse como un solo equipo. Los clústeres se componen de varias máquinas independientes conectadas por una red. Mientras que las máquinas de un clúster tienen que ser simétricas, de no serlo, el balance de carga es más difícil de lograr. El tipo más común de clúster es el clúster Beowulf que es uno implementado con múltiples ordenadores comerciales idénticos conectados a una red de área local TCP/IP Ethernet. La tecnología Beowulf fue desarrollada originalmente por Thomas Sterling y Donald Becker. La gran mayoría de los superordenadores TOP500 son clústeres.
- Procesamiento paralelo masivo
    - Un MPP es un solo equipo con varios más conectados a red. Tienen muchas de las características de los clúster pero cuentan con redes especializadas de interconexión. En cada MPP cada CPU tiene su propia memoria y una copia del sistema operativo y de la aplicación. Cada subsistema se comunica con los demás a través de una interconexión a alta velocidad
- Computación distribuida
    - Es la forma más distribuida de la computación paralela. Se hace uso de ordenadores que se comunican a través de internet para trabajar en un problema dado.
    - La mayoría de la computación distribuida utiliza middleware que es software que se encuentra entre el sistema operativo y la aplicación. El más común es la interfaz de puerta abierta de Berkeley (BOINC)
- Computadoras paralelas especializadas
    - No son específicos, sólo para problemas muy específicos
- Cómputo de propósito general en unidades de procesamiento gráfico (GPGPU)
    - Tendencia relativamente reciente en la investigación de cómputo paralelo que utiliza álgebra lineal y operaciones con matrices

Otras arquitecturas de programación en paralela:

- Paralelismo del pedacito-nivel
- Paralelismo de instrucción-nivel
- Paralelismo de los datos
- Paralelismo de las tareas

▼ **12-08-2022**

Continuación de la presentación de ayer

▼ **16-08-2022**

Investigación de artículos de la materia

Algunas fuentes de la biblioteca de la UAA son:

- EBSCO
- ACM
- IEEE
- DOAJ
- Dictnet
- Latindex
- Xplorer
- Redalyc

▼ **17-08-2022**

Revisión de artículos investigados

▼ **18-08-2022**

Artículo web: ¿Por qué las GPUs consiguen más rendimiento entre generaciones que las CPUs?

▼ **19-08-2022**

No hubo clase

▼ **22-08-2022**

Repaso de la clase anterior

Usaremos el libro de CUDA de NVIDIA

Review del libro de CUDA

▼ **23-08-2022**

Vimos más tarjetas gráficas

▼ **24-08-2022**

Más análisis de tarjetas gráficas

▼ **25-08-2022**

No fui a clase

▼ **26-08-2022**

No hubo clase

▼ **29-08-2022**

Comenzamos la unidad 3: Programación Paralela CUDA

CUDA: Compute Unified Device Architecture

Es de NVIDIA

Permite la programación en paralelo mediante el uso de GPUs

Inició en 2007

▼ **30-08-2022**

Más sobre CUDA

▼ **31-08-2022**

Todavía más sobre CUDA

▼ **01-09-2022**

No entré a clase

▼ 02-09-2022

Ejemplo de hola mundo con CUDA:

```
#include <iostream>
#include <stdio.h>
#include <book.h>

__global__ void kernel(void)
{

}

int main(void)
{
  kernel<<<1,1>>>();
  printf("Hello World\n");
  getchar();
  return 0;
}
```

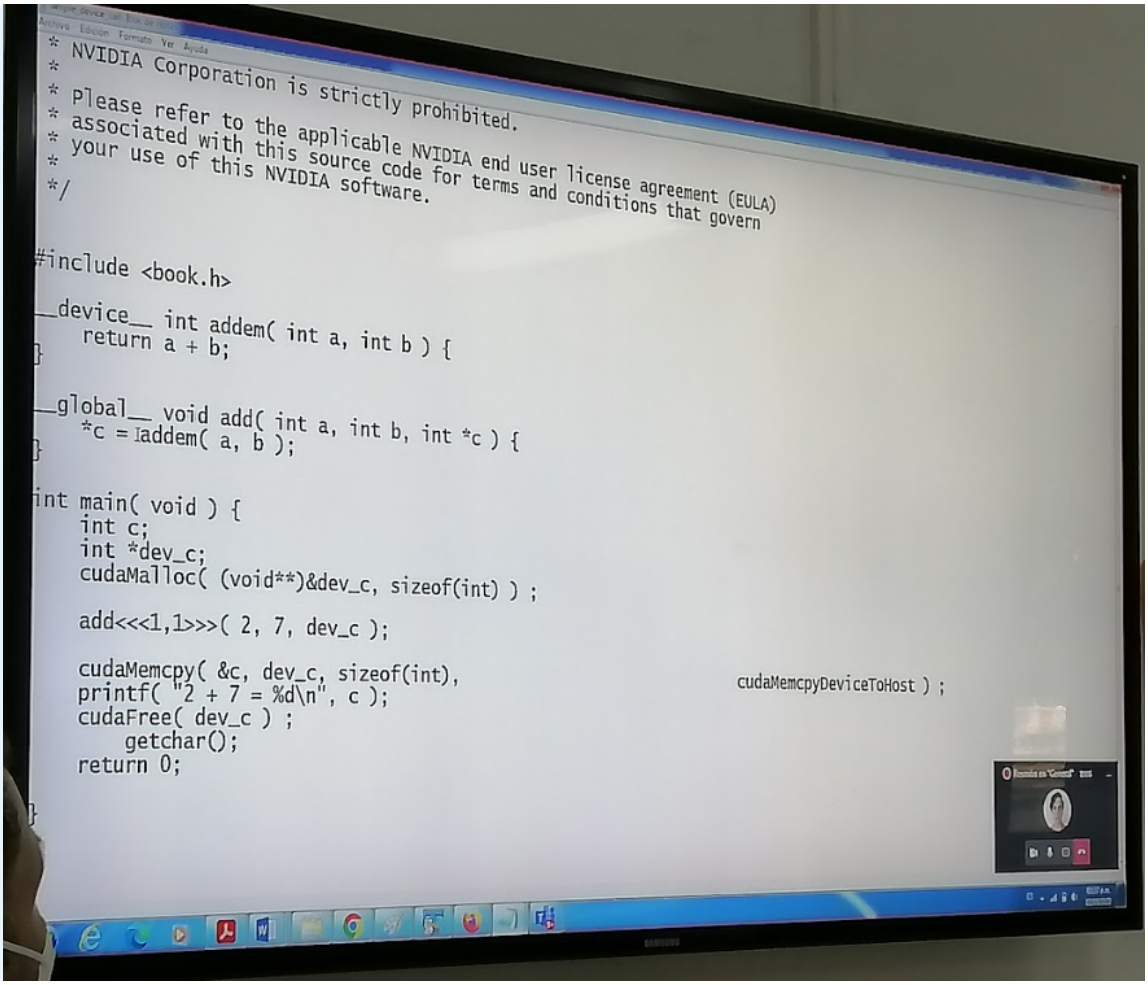
El tipo de función tiene que ser global, device o host

Será entonces: “el tipo de función” void “nombre de la función” (”lista de parámetros”)

Luego posteriormente: “nombre de la función” <<<1,1>>> (”lista de parámetros”)

Ambas listas de parámetros deben ser iguales

Y el <<<1,1>>> es la estructura de la memoria, donde el primer número es el bloque y el segundo es el hilo



▼ **05-09-2022**

Revisando el libro CUDA by example

Viendo el capítulo 1 sobre why cuda why now

Los ejemplos de programación CUDA en C++ están sacados desde el capítulo 3

▼ **06-09-2022**

Hablando del examen tentativamente el viernes

▼ **07-09-2022**

▼ **08-09-2022**

No hubo clase

▼ **Guía de estudio para el examen**

▼ Unidad 1

¿Qué es paralelización?

Es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente.

¿Para qué sirve la paralelización?

Se utiliza para resolver problemas en los que los recursos de una sola máquina no son suficientes. La finalidad de paralelizar un algoritmo es disminuir el tiempo de procesamiento mediante la distribución de tareas entre los procesadores disponibles.

Modelos o tipos de arquitecturas con base en el hardware

- Computación multinúcleo: Un procesador que incluye múltiples unidades de ejecución (núcleos) en el mismo procesador
- Multiprocesamiento simétrico: Sistema computacional con múltiples procesadores idénticos que comparten memoria y se conectan a través de un bus
- Computación en clúster: Grupo de ordenadores débilmente acoplados que trabajan en estrecha colaboración, de modo que en algunos aspectos pueden considerarse como un solo equipo
- Procesamiento paralelo masivo: Un solo equipo con varios procesadores conectados en red

- Computación distribuida: Se hace uso de ordenadores que se comunican a través de la Internet para trabajar en un problema dado
- Computadoras paralelas especializadas: Existen dispositivos paralelos especializados que generan interés
- Cómputo reconfigurable con arreglos de compuertas programables: Es el uso de un arreglo de compuertas programables (FPGA) como coprocesador de un ordenador de propósito general. Un FPGA es, en esencia, un chip de computadora que puede reconfigurarse para una tarea determinada.
- Cómputo de propósito general en unidades de procesamiento gráfico (GPGPU): Es una tendencia relativamente reciente en la investigación de ingeniería informática. Los GPUs son co-procesadores que han sido fuertemente optimizados para procesamiento de gráficos por computadora. El procesamiento de gráficos por computadora es un campo dominado por operaciones sobre datos en paralelo, en particular de algebra lineal y operaciones con matrices.

Aplicaciones

Un uso principal es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran relevancia, los cuales requieren de una gran capacidad de procesamiento de tiempo y de espacio de memoria, debido a las complejas operaciones que se deben realizar en problema NP-Duros/NP-Completos.

Otro uso es el de gráficos generadas por computadora. La generación de fotogramas requiere de una gran cantidad de cálculos matemáticos. Esto supone una tarea muy compleja para un solo procesador, por lo que es necesario que haya algún tipo de paralelismo

▼ Unidad 2

¿Qué es una GPU?

Una unidad de procesamiento gráfico o procesador gráfico es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador/CPU central en aplicaciones como los videojuegos o aplicaciones 3D interactivas.

Marcas

- NVIDIA
- AMD

- MSI
- ASUS
- GIGABYTE

Modelos

- MSI NVIDIA GeForce RTX 3080 Gaming Z Trio 10G
- EVGA NVIDIA RTX 3090 FTW3 Ultra Gaming 24G
- Asus ROG STRIX RTX 3090 O24G Gaming

Características

Características importantes:

- Frecuencia
- Memoria
- Teraflops

GPU vs CPU

▼ Unidad 3

> CUDA

CUDA son las siglas de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo) que hace referencia a una plataforma de computación en paralelo que incluye un compilador y un conjunto de herramientas de desarrollo creadas por Nvidia que permiten a los programadores usar una variación del lenguaje de programación C (CUDA C) para codificar algoritmos en GPU de Nvidia

> ## Instrucciones

- host
- device
- kernel

- host to device
- host to host

Ejemplo de hola mundo con CUDA:

```
#include <iostream>
#include <stdio.h>
#include <book.h>

__global__ void kernel(void)
{

}

int main(void)
{
  kernel<<<1,1>>>();
  printf("Hello World\n");
  getchar();
  return 0;
}
```

El tipo de función tiene que ser global, device o host

Será entonces: “el tipo de función” void “nombre de la función” (”lista de parámetros”)

Luego posteriormente: “nombre de la función” <<<1,1>>> (”lista de parámetros”)

Ambas listas de parámetros deben ser iguales

> Memoria

En el código anterior el <<<1,1>>> es la estructura de la memoria, donde el primer número es el bloque y el segundo es el hilo

▼ **09-09-2022**

Primer examen parcial

▼ Segundo Parcial

▼ **12-09-2022**

▼ **13-09-2022**

No hubo clase por el examen de Ornelas

▼ **14-09-2022**

▼ **15-09-2022**

▼ **19-09-2022**

El profe llegó y de la manera más random empezó a anotar código en el pizarrón:

```
__device__ int factorial(int n)
{
    if(n < 2)
    {
        return 1;
    }
    return n * factorial(n - 1);
```

```
}

__global__ void kernel(int *dev_num, int *dev_fac)
{
  dev_fac = factorial(dev_num);

}

int main(void)
{
    int numero;
    int *dev_num, *dev_fac;

    cudaMalloc((void**), &dev_num, sizeOf(int));
    cudaMalloc((void**), &dev_fac, sizeOf(int));

    printf("Inserte un número: ");
    scanf("%d", &numero);

    cudaMemcpy(dev_num, numero, sizeOf(int),cudaMemcpyHostToDevice);

    kernel<<<1, 1>>>(numero, dev_fac);
    cudaMemcpy(fact, dev_fac, sizeOf(int), cudaMemcpyDeviceToHost);

    printf("El factorial de %d es %d", numero, fact);

    cudaFree(dev_num);
    cudaFree(dev_fac);

    return 0;
}
```

▼ **20-09-2022**

No sé, no entré a clase

▼ **21-09-2022**

Empezamos a hablar de Dim3

Grids

Blocks

Threads

DimGrid()

DimBlock()

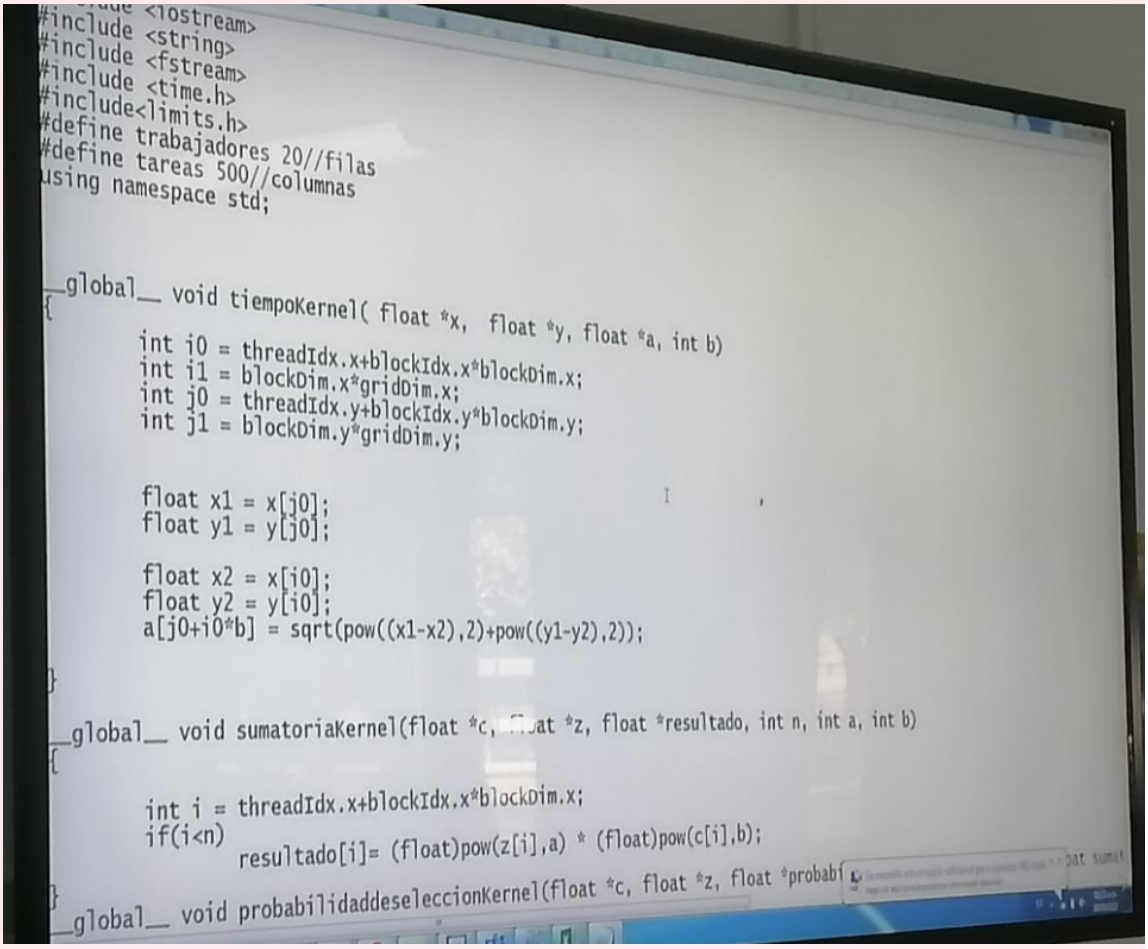
▼ **22-09-2022**

No sé, creo que me salí

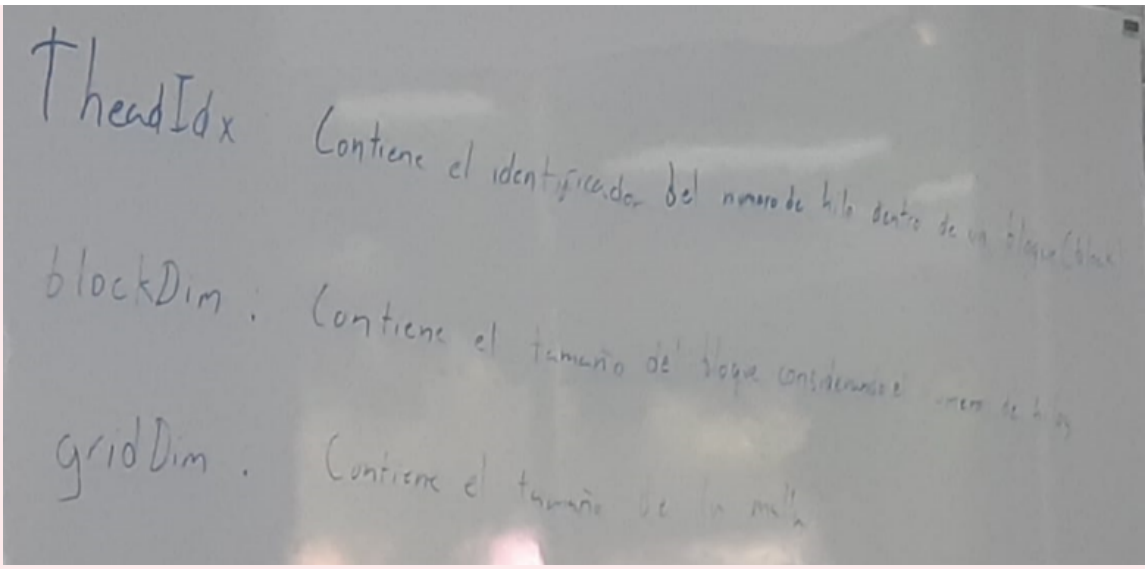
▼ **23-09-2022**

No fuimos a clase

▼ **26-09-2022**



```
#include <iostream>
#include <string>
#include <fstream>
#include <time.h>
#include<limits.h>
#define trabajadores 20//filas
#define tareas 500//columnas
using namespace std;


__global__ void tiempoKernel( float *x,  float *y, float *a, int b)
{
        int i0 = threadIdx.x+blockIdx.x*blockDim.x;
        int i1 = blockDim.x*gridDim.x;
        int j0 = threadIdx.y+blockIdx.y*blockDim.y;
        int j1 = blockDim.y*gridDim.y;


        float x1 = x[j0];
        float y1 = y[j0];

        float x2 = x[i0];
        float y2 = y[i0];
        a[j0+i0*b] = sqrt(pow((x1-x2),2)+pow((y1-y2),2));

}

__global__ void sumatoriaKernel(float *c, float *z, float *resultado, int n, int a, int b)
{

        int i = threadIdx.x+blockIdx.x*blockDim.x;
        if(i<n)
                resultado[i]= (float)pow(z[i],a) * (float)pow(c[i],b);

}
__global__ void probabilidaddeseleccionKernel(float *c, float *z, float *probabi
```



👍

▼ **27-09-2022**

▼ **28-09-2022**

▼ **29-09-2022**

▼ **30-09-2022**

No hubo clase

▼ **03-10-2022**

No hubo clases por el congreso de ICI

▼ **04-10-2022**

No hubo clases por el congreso de ICI

▼ **05-10-2022**

No hubo clases por el congreso de ICI

▼ **06-10-2022**

No hubo clase

▼ **07-10-2022**

No hubo clase

▼ **10-10-2022**

No hubo clase

▼ **11-10-2022**

No hubo clase

▼ **12-10-2022**

No hubo clase

▼ **13-10-2022**

No hubo clase

▼ **14-10-2022**

No hubo clase

▼ **17-10-2022**

No hubo clase

▼ **18-10-2022**

No hubo clase

▼ **19-10-2022**

No hubo clase

▼ **20-10-2022**

No hubo clase

▼ **21-10-2022**

No hubo clase

▼ **24-10-2022**

Avances para el examen

▼ **25-10-2022**

Avances para el examen

▼ **26-10-2022**

Avances para el examen

▼ **27-10-2022**

Avances para el examen

▼ **28-10-2022**

Avances para el examen

▼ Tercer Parcial

▼ **31-10-2022**

Hablamos de las optativas

▼ **01-11-2022**

?

▼ **03-11-2022**

?

▼ **04-11-2022**

No hubo clases

▼ **07-11-2022**

?

▼ **08-11-2022**

Revisión del segundo parcial

▼ **09-11-2022**

Revisión del segundo parcial

▼ **10-11-2022**

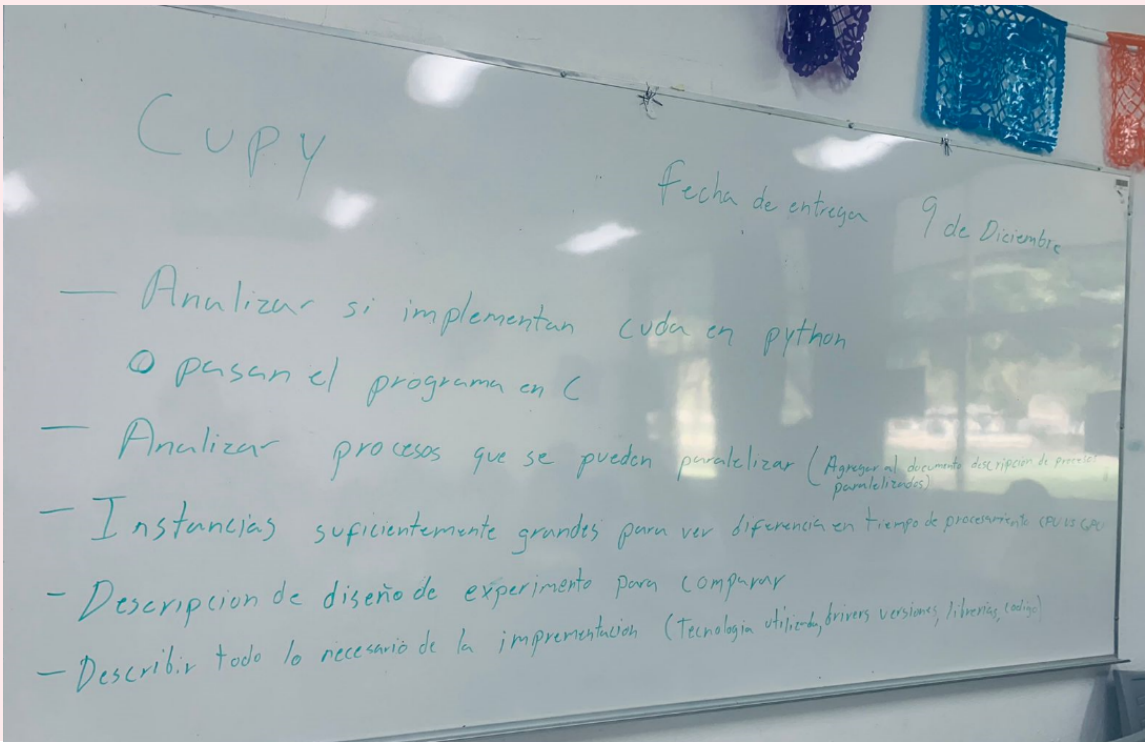
Revisión del segundo parcial

▼ **11-11-2022**

Revisión del segundo parcial

▼ **14-11-2022**

No entré a clase



▼ **15-11-2022**

No entré a clase

▼ **16-11-2022**

No entré a clase

▼ **17-11-2022**

No entré a clase

▼ **18-11-2022**

No entré a clase

▼ **22-11-2022**

No entré a clases

▼ **23-11-2022**

No entré a clases

▼ **24-11-2022**

No entré a clases

▼ **25-11-2022**

No entré a clases