

Stencil y experimentos en GPU

Auxiliar #3

Profesor: Nancy Hitschfeld

Auxiliar: Vicente González

15 / 04 / 2024

~/CC7515 – Computación en GPU/Aux3

> Introducción

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria
- Search
 - Búsqueda de elementos en una estructura de datos

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria
- Search
 - Búsqueda de elementos en una estructura de datos

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria
- Search
 - Búsqueda de elementos en una estructura de datos
- Sort
 - Ordenación de elementos de una estructura de datos

Operaciones comunes

- Map
 - Aplicar una función a cada elemento de una estructura de datos.
- Reduction
 - Reducción a un solo valor mediante la aplicación de una operación binaria.
- Scan
 - Construcción de una estructura de datos acumulativa mediante la aplicación de una operación binaria
- Search
 - Búsqueda de elementos en una estructura de datos
- Sort
 - Ordenación de elementos de una estructura de datos

~/CC7515 – Computación en GPU/Aux3

> Stencil

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.
- Se basa en actualizar cada punto de una imagen o conjunto de datos en función de los valores de sus vecinos cercanos.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.
- Se basa en actualizar cada punto de una imagen o conjunto de datos en función de los valores de sus vecinos cercanos.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.
- Se basa en actualizar cada punto de una imagen o conjunto de datos en función de los valores de sus vecinos cercanos.
- CUDA y OpenCL son frameworks que permiten implementar esta técnica de manera paralela.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.
- Se basa en actualizar cada punto de una imagen o conjunto de datos en función de los valores de sus vecinos cercanos.
- CUDA y OpenCL son frameworks que permiten implementar esta técnica de manera paralela.

La técnica Stencil

- Es una técnica de programación que se utiliza para procesamiento de imágenes, simulaciones de fluidos, entre otras aplicaciones.
- Se basa en actualizar cada punto de una imagen o conjunto de datos en función de los valores de sus vecinos cercanos.
- CUDA y OpenCL son frameworks que permiten implementar esta técnica de manera paralela.
- Al utilizar la técnica de stencil en paralelo, se pueden obtener mejoras significativas en el rendimiento y tiempo de procesamiento de las aplicaciones que la utilizan.

Ejemplo

Input



Contour Stencil Interpolation



Ejemplo

Una convolución es un buen ejemplo de un Stencil.

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada píxel de la imagen.

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada píxel de la imagen.
- El stencil se define de la siguiente manera:

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada píxel de la imagen.
- El stencil se define de la siguiente manera:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada pixel de la imagen.
- El stencil se define de la siguiente manera:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

- Se aplica iterativamente en cada punto de la imagen, actualizando el valor de cada pixel en función de los valores de sus vecinos cercanos y el stencil.

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada píxel de la imagen.
- El stencil se define de la siguiente manera:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

- Se aplica iterativamente en cada punto de la imagen, actualizando el valor de cada píxel en función de los valores de sus vecinos cercanos y el stencil.
- Al utilizar CUDA u OpenCL para implementar esta técnica de manera paralela, se pueden obtener mejoras significativas en el rendimiento y tiempo de procesamiento de la imagen.

Ejemplo

Una convolución es un buen ejemplo de un Stencil.

- Aplicaremos la técnica de stencil para suavizar una imagen. Para ello, utilizaremos un stencil de 3x3 que aplicará una media ponderada de los valores de los píxeles vecinos a cada píxel de la imagen.
- El stencil se define de la siguiente manera:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

- Se aplica iterativamente en cada punto de la imagen, actualizando el valor de cada píxel en función de los valores de sus vecinos cercanos y el stencil.
- Al utilizar CUDA u OpenCL para implementar esta técnica de manera paralela, se pueden obtener mejoras significativas en el rendimiento y tiempo de procesamiento de la imagen.

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \quad \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \quad \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

- La operación de suavizado para ese píxel sería:

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

- La operación de suavizado para ese píxel sería:

$$(1/9 \cdot 1) + (1/9 \cdot 2) + (1/9 \cdot 3) + (1/9 \cdot 4) + (1/9 \cdot X) + (1/9 \cdot 6) + (1/9 \cdot 7) + (1/9 \cdot 8) + (1/9 \cdot 9) \\ = (X + 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9)/9$$

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

- La operación de suavizado para ese píxel sería:

$$(1/9 \cdot 1) + (1/9 \cdot 2) + (1/9 \cdot 3) + (1/9 \cdot 4) + (1/9 \cdot X) + (1/9 \cdot 6) + (1/9 \cdot 7) + (1/9 \cdot 8) + (1/9 \cdot 9) \\ = (X + 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9)/9$$

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

- La operación de suavizado para ese píxel sería:

$$(1/9 \cdot 1) + (1/9 \cdot 2) + (1/9 \cdot 3) + (1/9 \cdot 4) + (1/9 \cdot X) + (1/9 \cdot 6) + (1/9 \cdot 7) + (1/9 \cdot 8) + (1/9 \cdot 9) \\ = (X + 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9)/9$$

- Donde X es el valor del píxel actual. El resultado de esta operación es el nuevo valor del píxel actual después de aplicar la operación de suavizado.

Ejemplo

- Por ejemplo, si el stencil se coloca en el siguiente píxel:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \quad \begin{matrix} 1 & 2 & 3 \\ 4 & X & 5 \\ 6 & 7 & 8 \end{matrix}$$

- La operación de suavizado para ese píxel sería:

$$(1/9 \cdot 1) + (1/9 \cdot 2) + (1/9 \cdot 3) + (1/9 \cdot 4) + (1/9 \cdot X) + (1/9 \cdot 6) + (1/9 \cdot 7) + (1/9 \cdot 8) + (1/9 \cdot 9) \\ = (X + 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9)/9$$

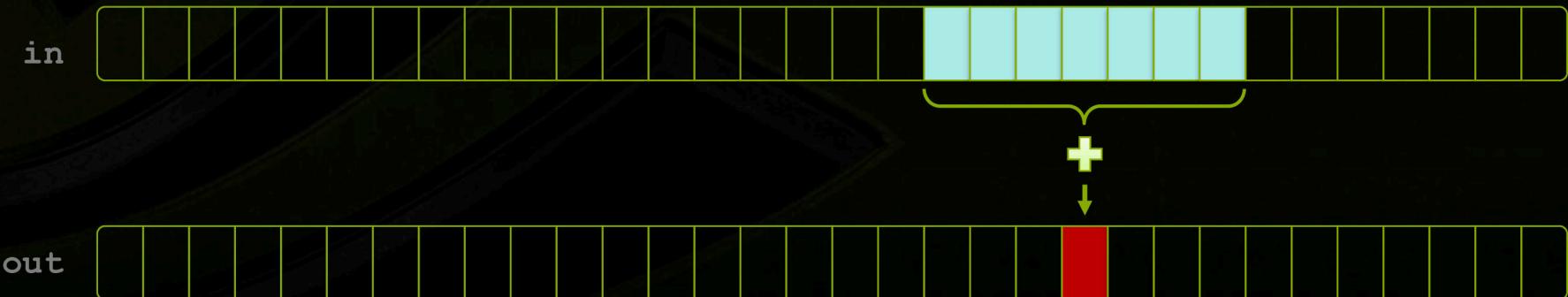
- Donde X es el valor del píxel actual. El resultado de esta operación es el nuevo valor del píxel actual después de aplicar la operación de suavizado.
- Este proceso se repite para cada píxel de la imagen, utilizando el stencil para calcular el nuevo valor de cada píxel en función de los valores de sus vecinos cercanos.

~/CC7515 – Computación en GPU/Aux3

> Optimizaciones

1D Stencil

- Consider applying a 1D stencil to a 1D array of elements
 - Each output element is the sum of input elements within a radius
- If radius is 3, then each output element is the sum of 7 input elements:



Implementing Within a Block

- Each thread processes one output element
 - `blockDim.x` elements per block
- Input elements are read several times
 - With radius 3, each input element is read seven times



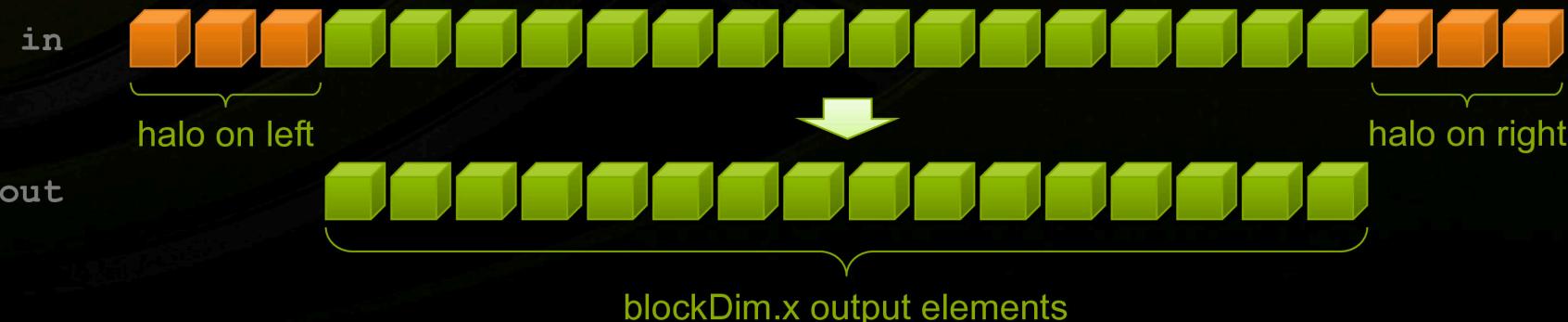


Sharing Data Between Threads

- Terminology: within a block, threads share data via **shared memory**
- Extremely fast on-chip memory
 - By opposition to device memory, referred to as **global memory**
 - Like a user-managed cache
- Declare using **`__shared__`**, allocated per block
- Data is not visible to threads in other blocks

Implementing With Shared Memory

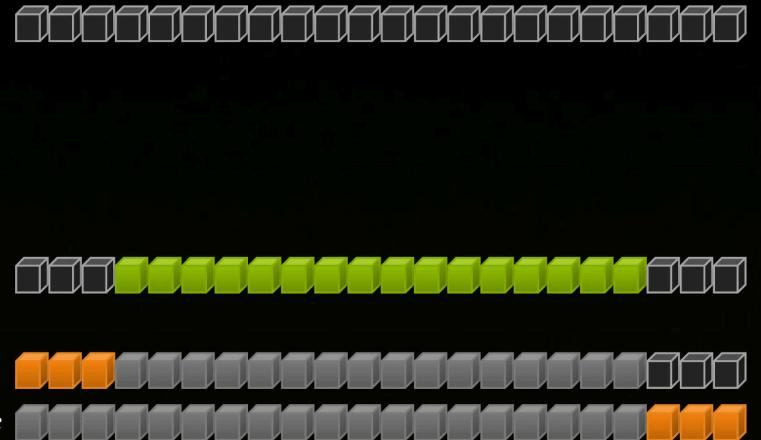
- Cache data in shared memory
 - Read $(blockDim.x + 2 * radius)$ input elements from global memory to shared memory
 - Compute $blockDim.x$ output elements
 - Write $blockDim.x$ output elements to global memory
- Each block needs a halo of $radius$ elements at each boundary



Stencil Kernel

```
__global__ void stencil_1d(int *in, int *out) {
    __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
    int gindex = threadIdx.x + blockIdx.x * blockDim.x;
    int lindex = threadIdx.x + RADIUS;

    // Read input elements into shared memory
    temp[lindex] = in[gindex];
    if (threadIdx.x < RADIUS) {
        temp[lindex - RADIUS] = in[gindex - RADIUS];
        temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
    }
}
```





Stencil Kernel

```
// Apply the stencil
int result = 0;
for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
    result += temp[lindex + offset];

// Store the result
out[gindex] = result;
}
```

Data Race!

- The stencil example will not work...
- Suppose thread 15 reads the halo before thread 0 has fetched it...

```
...
temp[lindex] = in[gindex];
if (threadIdx.x < RADIUS) {
    temp[lindex - RADIUS] = in[gindex - RADIUS];
    temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
}
int result = 0;
for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
    result += temp[lindex + offset];  Load from temp[19]
...

```

Store at temp[18]



Skipped since threadIdx.x > RADIUS





__syncthreads()

- `void __syncthreads();`
- Synchronizes all threads within a block
 - Used to prevent RAW / WAR / WAW hazards
- All threads must reach the barrier
 - In conditional code, the condition must be uniform across the block



Stencil Kernel

```
__global__ void stencil_1d(int *in, int *out) {
    __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
    int gindex = threadIdx.x + blockIdx.x * blockDim.x;
    int lindex = threadIdx.x + radius;

    // Read input elements into shared memory
    temp[lindex] = in[gindex];
    if (threadIdx.x < RADIUS) {
        temp[lindex - RADIUS] = in[gindex - RADIUS];
        temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
    }

    // Synchronize (ensure all the data is available)
    __syncthreads();
```



Stencil Kernel

```
// Apply the stencil
int result = 0;
for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
    result += temp[lindex + offset];

// Store the result
out[gindex] = result;
}
```



Review (1 of 2)

- Launching parallel threads
 - Launch N blocks with M threads per block with `kernel<<<N,M>>>(...);`
 - Use `blockIdx.x` to access block index within grid
 - Use `threadIdx.x` to access thread index within block
- Allocate elements to threads:

```
int index = threadIdx.x + blockIdx.x * blockDim.x;
```



Review (2 of 2)

- Use `__shared__` to declare a variable/array in shared memory
 - Data is shared between threads in a block
 - Not visible to threads in other blocks

- Use `__syncthreads()` as a barrier
 - Use to prevent data hazards

~/CC7515 – Computación en GPU/Aux3

> Implementación

Implementación



<https://github.com/Seivier/StencilCL>

(pauta en rama reference)

~/CC7515 – Computación en GPU/Aux3

>

Experimentos

Recomendaciones

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device
 - Ejecución del kernel

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device
 - Ejecución del kernel
 - Copia de datos de device a host

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device
 - Ejecución del kernel
 - Copia de datos de device a host
 - Tiempo total de toda la operación

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device
 - Ejecución del kernel
 - Copia de datos de device a host
 - Tiempo total de toda la operación
2. Correr experimento más de una vez (5-10)

Recomendaciones

1. Calcular el tiempo de las distintas etapas por separado
 - Generación de datos aleatorios
 - Copia de datos de host a device
 - Ejecución del kernel
 - Copia de datos de device a host
 - Tiempo total de toda la operación
2. Correr experimento más de una vez (5-10)
3. Para calcular el speed-up:

$$s = \frac{t_{GPU}}{t_{CPU}}$$

Recomendaciones

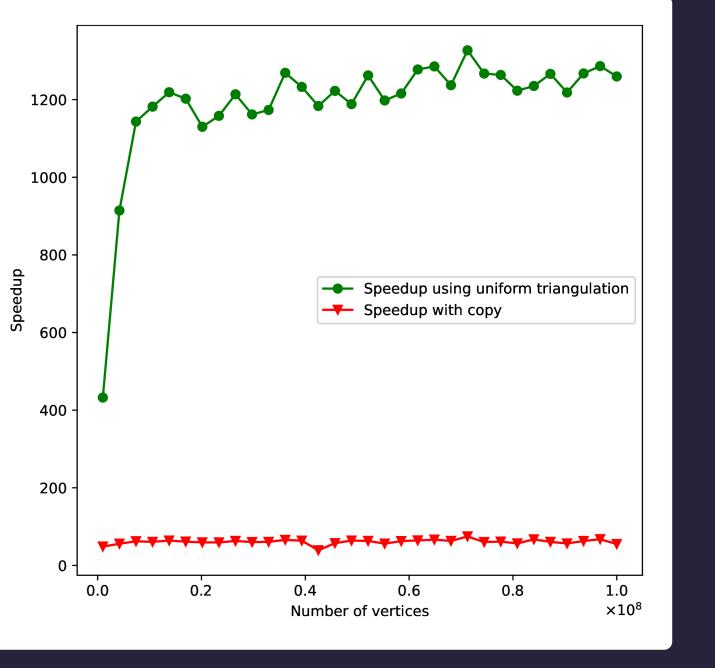


Figure 1: Un solo experimento

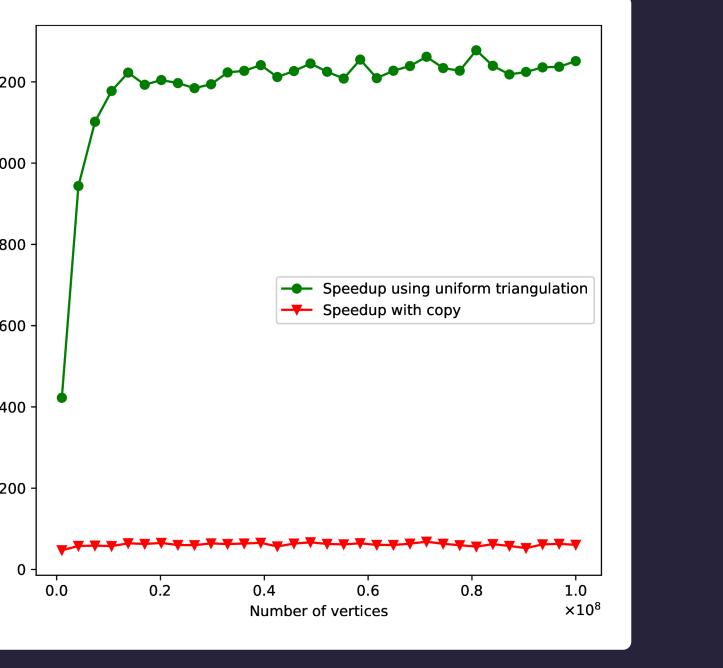


Figure 2: Promedio de 5 experimentos

~/CC7515 – Computación en GPU/Aux3

> Referencias

- Blog de Nvidia

<https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/>

- Presentación de Nvidia

<https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf>

- Experimentos GPolylla:

<https://github.com/ssalinasfe/GPolylla/tree/main/experiments>

- Presentación de Sergio:

https://www.u-cursos.cl/ingenieria/2023/1/CC7515/1/material_docente/bajar?bajar=1&id=6539157