PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UNWARPING DE VIDEOS OMNIDIRECCIONALES EN LA PLATAFORMA JETSON TK1

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Gustavo Manuel Silva Obregón

ASESOR: Paul Antonio Rodríguez Valderrama

Lima, 2015

Para mis seres queridos.

Resumen

El *unwarping* es un método utilizado para transformar imágenes omnidireccionales en imágenes panorámicas, el cual es empleado en aplicaciones tales como seguridad, visión robótica, geolocalización, etc. El procesamiento de imágenes omnidireccionales de alta resolución y su aplicación en dispositivos móviles se ve limitado por el costo computacional y costo energético. Para ello, se plantea como herramienta principal utilizar la plataforma Jetson TK1, la cual es un *system on chip* (SoC) creada por Nvidia que se caracteriza por su alto rendimiento computacional y bajo costo energético al tener incorporado 192 núcleos en su procesador gráfico.

En el presente trabajo se desarrolla e implementa un algoritmo para realizar el *unwarping* de videos omnidireccionales en la plataforma Jetson TK1, la cual permite optimizar las transferencias y procesamientos de datos realizados en su GPU. El algoritmo es implementado en el entorno de programación MATLAB y CUDA para evaluar error por cálculo y eficiencia computacional. Asimismo, se compara en rendimiento computacional con el método PMPA, el cual es una alternativa escrita en lenguaje C computacionalmente eficiente en comparación a otros métodos presentados en el Capítulo 1. Los resultados de la comparación muestran que la implementación propuesta es 1.35 a 8.12 veces más rápida que el algoritmo PMPA para los tipos de interpolación utilizados (interpolación vecino más cercano e interpolación bilineal).

El orden que sigue la tesis es el siguiente: En el primer capítulo se realizara un breve estado del arte sobre los métodos para realizar el *unwarping* de imágenes omnidireccionales. En el segundo capítulo se cubren los aspectos teóricos del modelo de programación CUDA necesarios para el diseño del algoritmo paralelo. En el tercer capítulo se describe de forma detallada el método propuesto y su diseño paralelo. Por último, en el cuarto capítulo se presentan los resultados computacionales seguido de las conclusiones y recomendaciones.

Finalmente, cabe señalar que el trabajo de investigación realizado fue presentado en el GPU Technology Conference 2015.

Índice general

Introducción **1**

1. Métodos de Unwarping **2**

1.1. Definición 2

1.2. Métodos universales 2

1.2.1. Mapeo Log-polar 2

1.2.2. Técnicas de Geometría Discreta (DGT) 4

1.2.3. Tabla de Mapeo Panorámico 4

1.3. Métodos a partir de la Tabla de Mapeo Panorámico 5

1.3.1. Un Octavo de Tabla de Mapeo Panorámico 5

1.3.2. Arreglo de Punteros de Mapeo Panorámico (PMPA) 6

2. CUDA (Compute Unified Device Architecture) **8**

2.1. Definición 8

2.2. Modelo de Programación 9

2.3. Modelo de Memoria 9

2.3.1. Memorias del Dispositivo 10

2.3.1.1. Memoria Global 10

2.3.1.2. Memoria Local 10

2.3.1.3. Memoria Constante 10

2.3.1.4. Memoria de Textura 11

2.3.2. Memorias On-chip 11

2.3.2.1. Memoria Compartida 11

2.3.2.2. Registros 11

2.3.3. Memoria Unificada 11

2.4. Plataforma Jetson TK1 13

2.5. Objetivos 14

3. Diseño del Algoritmo Propuesto **15**

3.1. Consideraciones de Diseño . . . 15

3.2. Diagrama de Bloques . . . 15

3.3. Descripción del Algoritmo Propuesto . . . 16

3.3.1. Calibración de la cámara . . . 16

3.3.2. Proyección de puntos . . . 17

3.3.3. Interpolación . . . 19

3.3.4. Reasignación de datos . . . 20

3.4. Descripción del Diseño Paralelo del Algoritmo Propuesto . . . 22

3.4.1. Requerimientos de Memoria . . . 22

3.4.2. Diseño Paralelo CUDA . . . 24

3.4.2.1. Accesos a memoria . . . 25

3.4.2.2. Tipos de memoria utilizadas . . . 25

3.4.2.3. Diseño y descripción de las etapas . . . 25

4. Implementación y Resultados Computacionales **29**

4.1. Consideraciones de la Implementación . . . 29

4.2. Decripción de la Implementación . . . 30

4.2.1. Implemetación en MATLAB . . . 30

4.2.2. Implemetación Paralela CUDA . . . 30

4.3. Resultados Computacionales . . . 30

4.3.1. Pruebas del algoritmo paralelo propuesto . . . 31

4.3.2. Pruebas para determinar el óptimo tamaño de bloque . . . 32

4.3.3. Análisis de resultados para el bloque seleccionado . . . 34

Conclusiones **38**

Recomendaciones **39**

Bibliografía **40**

Introducción

En la actualidad, las cámaras omnidireccionales son muy utilizadas por su gran rango de visión (360 grados) en aplicaciones como video vigilancia (seguimiento y detección de personas y vehículos) [1-4], visión robótica [5-8], reconstrucción 3D [9-11], etc [12]. Sin embargo, las imágenes obtenidas por dichas cámaras (imágenes omnidireccionales) requieren de un procesamiento previo para obtener una imagen de fácil análisis [19] para las aplicaciones mencionadas. Para ello, se realiza una transformación de imagen omnidireccional a una imagen panorámica, llamada *unwarping* [13, 16]. Este método es utilizado por su rendimiento computacional en comparación con otras técnicas de transformación o reconstrucción como transformada de Fourier esférica.

Por otro lado, el aumento de resolución de las imágenes ha producido que se busque obtener nuevos métodos y emplear nuevos dispositivos (p.e: [14] y [15]) que permitan realizar el *unwarping* en menores tiempos de procesamiento. Dispositivos como los GPUs (unidades de procesamiento gráfico) [24] son plataformas que tienen una gran capacidad computacional para el procesamiento de imágenes. Sin embargo, estas plataformas tienen un alto consumo energético, lo cual dificulta su uso en dispositivos portátiles para las aplicaciones mencionadas. La Plataforma Jetson TK1 [27] de Nvidia, que integra el procesador Tegra TK1 [28] es el más avanzado procesador móvil que tiene un rendimiento similar que los GPUs. El procesador tiene integrados 192 núcleos CUDA de tecnología Kepler que permiten un bajo consumo energético (menor a 5 Watts) y un alto rendimiento (hasta 300 GFLOPS).

En el presente trabajo de tesis, se propone diseñar e implementar un algoritmo que realice el *unwarping* de videos omnidireccionales en la plataforma Jetson TK1. Se utilizará el modelo de programación CUDA (Compute Unified Device Architecture) para tal propósito.

Finalmente, cabe señalar que el trabajo de investigación realizado fue presentado en el GPU Technology Conference 2015 [37].

Capítulo 1

Métodos de Unwarping

1.1. Definición

Unwarping es un método utilizado para transformar una imagen omnidireccional en una imagen panorámica, al calcular la relación de transformación entre ambas imágenes. El método reduce la deformación presente en las imágenes omnidireccionales, que es causada por el espejo convexo de la cámara omnidireccional. La deformación presente dificulta el directo análisis de las imágenes mediante técnicas y procesos clásicos utilizados en el área de procesamiento de imágenes digitales.

En las últimas dos décadas se han desarrollado un serie de métodos para realizar el *unwarping* de imágenes omnidireccionales tales como tabla de mapeo panorámico, un octavo de la tabla de mapeo panorámico, etc. Estos métodos fueron clasificados en dos áreas para su fácil comprensión: métodos universales [21] (mapeo log-polar, técnicas de geometría discreta y tabla de mapeo panorámico) y métodos a partir de la tabla de mapeo panorámico.

1.2. Métodos universales

1.2.1. Mapeo Log-polar

Mapeo log-polar [17, 18] es un tipo de modelo variante espacial donde la separación de pixeles en la imagen incrementa linealmente con la distancia (Figura 1.1). El método de mapeo log-polar consiste en muestrear la imagen capturada mediante una grilla variante espacial (Figura 1.1) para pasar de una forma cartesiana a una log-polar. El muestreo de coordenadas cartesianas a coordenadas log-polares está dada por la ecuación 1.1

$$\rho(x_i, y_i) \quad = \quad \log\left(\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}\right)$$

$$\begin{aligned} \theta(x_i, y_i) &= \frac{\pi}{2} - \arctan(\frac{y_i - y_c}{x_i - x_x}) \qquad \theta \in [0, \pi] \\ \theta(x_i, y_i) &= \frac{3\pi}{2} - \arctan(\frac{y_i - y_c}{x_i - x_x}) \qquad \theta \in [\pi, 2\pi], \end{aligned} \tag{1.1}$$

donde ρ es la distancia en escala logarítmica entre un punto dado y el punto central, θ es el ángulo entre una línea de referencia (eje x de la coordenada cartesiana) y la línea que pasa por el punto central y el punto dado, y (x_c, y_c) es la coordenada del punto central de la imagen original en el plano cartesiano.

La imagen muestreada de forma log-polar (imagen omnidireccional) será mapeada en una forma cartesiana para generar la imagen panorámica. El mapeo de forma log-polar a forma cartesiana está dado por la ecuación 1.2

$$\begin{aligned} x(\rho, \theta) &= \exp(\rho).cos(\theta) - x_c \\ y(\rho, \theta) &= \exp(\rho).sin(\theta) - y_c, \end{aligned} \tag{1.2}$$

donde (x, y) es coordenada cartesiana del punto mapeado en la imagen panorámica.

Un ejemplo de método log-polar se puede observar en la Figura 1.1 donde se captura una imagen omnidireccional esta es muestreada de forma log-polar y mapeada de forma cartesiana para generar la imagen panorámica. La imagen panorámica obtenida con este método se caracteriza por tener una menor resolución en comparación con la imagen omnidireccional, pero un mejor aspecto visual (imagen más fluida y con poca deformación). En tiempos de procesamiento este método es más veloz que el método de técnicas de geometría discreta pero más lento que el método de tabla de mapeo panorámico (métodos que se describen más adelante).

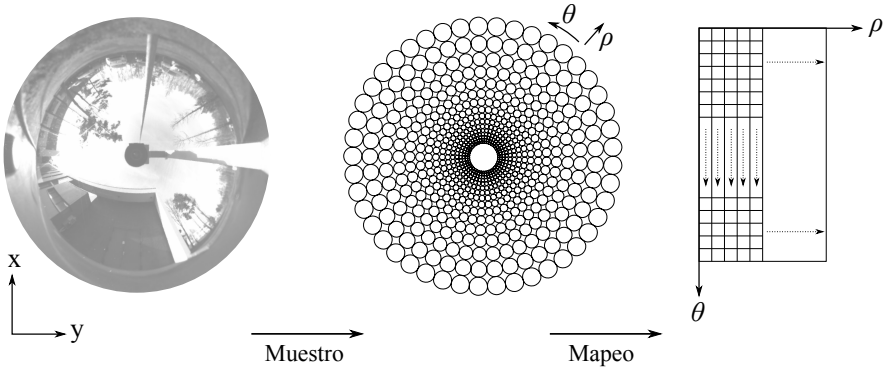

Figura 1.1: Proceso de mapeo log-polar [21].

1.2.2. Técnicas de Geometría Discreta (DGT)

Las técnicas de geometría discreta [19] son un método que permite obtener imágenes panorámicas con alta resolución. El método consiste en trabajar de forma discreta con cada uno de los pixeles para realizar el *unwarping*. El *unwarping* en este método se realiza al transformar la imagen omnidireccional en una imagen panorámica usando el modelo de remuestro PDE [19].

El método DGT consiste en definir diferentes círculos concéntricos de radios r en la imagen omnidireccional y extraer los pixeles que conforman los círculos formando una imagen con resolución no uniforme (cada círculo que conforma la imagen tiene diferente radio), como se muestra en la Figura 1.2 a. Luego se realiza un remuestreo utilizando el modelo de la cámara para espaciar los datos y formar una imagen con resolución uniforme (Figura 1.2 b). La nueva imagen (imagen panorámica) cuenta con espacios que no tienen datos, para ello se utiliza el modelo de conversión de resolución [19], el cual permite asignar un dato al pixel vacio con los pixeles de su alrededor. Asimismo, el modelo de conversión de resolución permite aumentar la resolucón de la imagen panorámica.

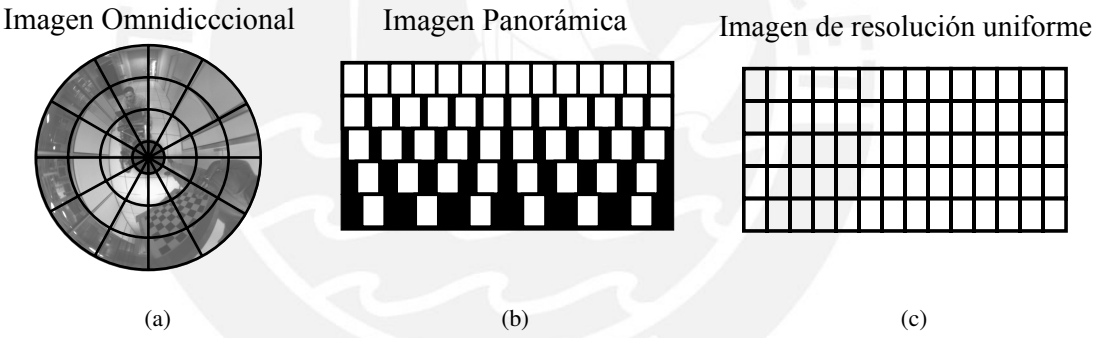

Figura 1.2: Proceso de técnicas de geometría discreta [19].

1.2.3. Tabla de Mapeo Panorámico

La tabla de mapeo panorámico [20] es un método eficiente creado por los investigadores S.W. Jeng y W.H Tsai. El método consiste en generar una tabla de mapeo panóramico, la cual relaciona los puntos del espacio real y los puntos de la imagen omnidireccional. La tabla se caracteriza por ser creada tan solo una vez, siempre y cuando no se varíen las características y las posiciones relativas entre la óptica y el sensor (cámara omnidireccional). Su genereción se realiza mediante los *landmarks*, los cuales son puntos característicos fáciles de identificar en la imagen omnidireccional y en el espacio real. Los *landmarks* (Figura 1.3 a) son una alternativa para realizar el *unwarping* sin depender de los parámetros de la cámara, que en ocasiones son difíciles de obtener.

La principal ventaja del método es la generación de la tabla de mapeo, la cual permite reducir los tiempos de procesamiento, disminuir la complejidad y obtener una imagen de mejor calidad [21] en comparación con los métodos de mapeo log-polar y técnicas de geometría discreta. En [21] se describe que al realizar el *unwarping* en MATLAB 5 veces para 5 diferentes imágenes y promediando los tiempos computacionales el método tabla de mapeo panorámico demora 1.22 segundos mientras que el método mapeo log-polar y el método de técnicas geometría discreta demoran 2.003 segundos y 3.426 segundos.

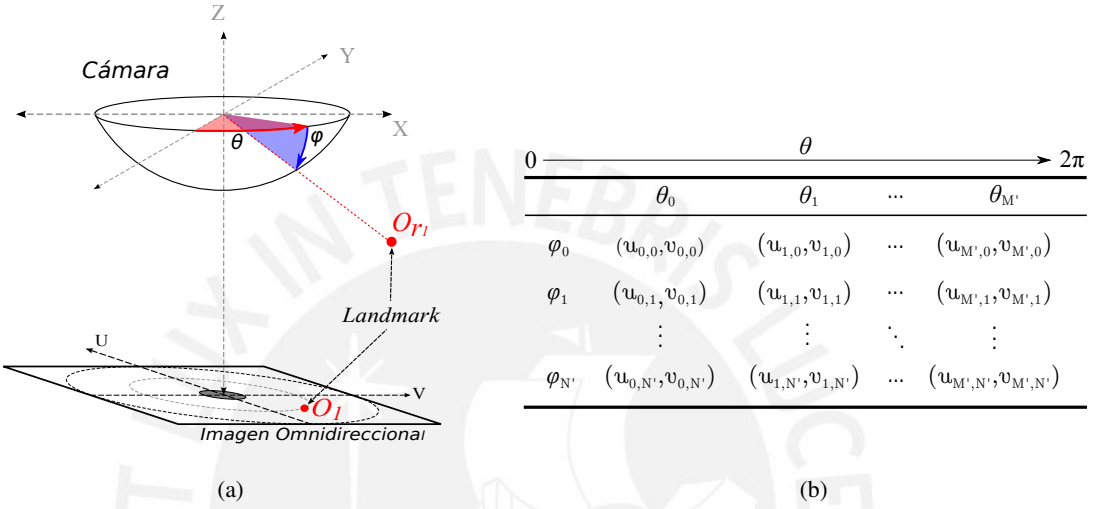

Figura 1.3: (a) Ejemplo de landmark y (b) Ejemplo de tabla de mapeo panorámico [20].

1.3. Métodos a partir de la Tabla de Mapeo Panorámico

1.3.1. Un Octavo de Tabla de Mapeo Panorámico

Un octavo de tabla de mapeo panorámico [22] es un método computaciomente eficiente que permite reducir tiempos de procesamiento y la cantidad de memoria utilizada en comparación con el método de tabla de mapeo panorámico. Este consiste en fraccionar la imagen omnidireccional en 8 sectores (Figura 1.4 a) y trabajar con un sector de forma tradicional [20] para calcular su correspondiente región de la tabla de mapeo (octava parte de la tabla de mapeo panorámico) como se observa en la Figura 1.4 b. Mediante simetría de ángulos y puntos se relacionan y se calculan las otras 7 regiones restante (regiones virtuales no almacenadas en memoria) a partir de la región de la tabla de mapeo disponible (región real que se encuentra almacenado en memoria). El cálculo de las otras 7 regiones virtuales es de baja dificultad, puesto que involucra operaciones simples, y se realiza cada vez que se genera una imagen panorámica.

Finalmente, el método tiene como ventajas menores tiempos de procesamiento al reducir

los accesos a memoria de la tabla de mapeo panorámico por cálculos simples. En [22] se describe que el método de un octavo de tabla de mapeo panorámico es 2.74 veces más rápido que el método de tabla de mapeo panorámico. Otra ventaja del método es la redución de la cantidad de memoria utilizada al calcular la octava parte de la tabla de mapeo de forma real (guardada en memoria).

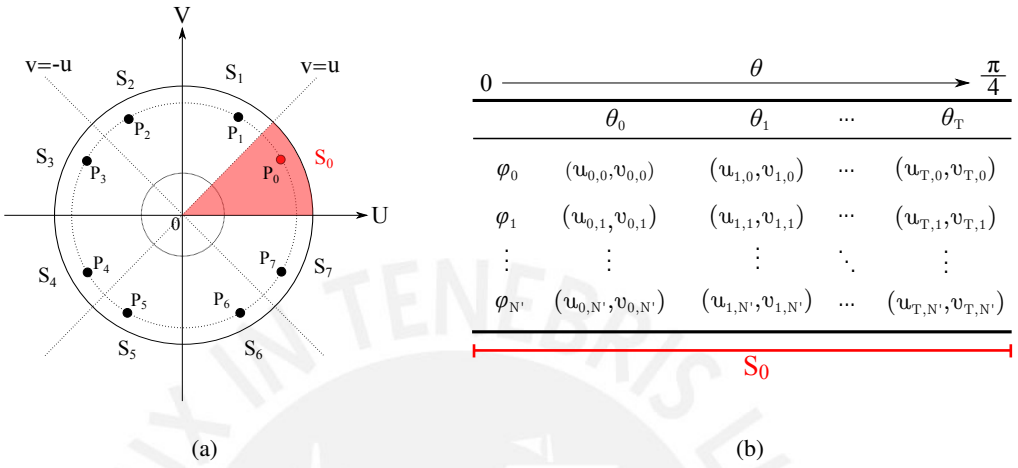

Figura 1.4: (a) Imagen omnidireccional dividida en ocho sectores simétricos y (b) Ejemplo de una octava parte de la tabla de mapeo panorámico [22].

1.3.2. Arreglo de Punteros de Mapeo Panorámico (PMPA)

PMPA [23] es una alternativa computacionalmente eficiente a la tabla de mapeo panorámico, que consiste en utilizar un arreglo de punteros (Figura 1.5 b) para realizar el *unwarping*. Este arreglo describe la relación de transformación de puntos de la imagen omnidireccional a la imagen panorámica, mediante direcciones de memoria del buffer de la imagen omnidireccional (Figura 1.5 a). El método PMPA a diferencia del método de tabla de mapeo panorámico genera la tabla de mapeo mediante una proyección clásica [34] que depende de los parámetros intrínsecos de la cámara. La tabla de mapeo es utilizada como una herramienta para generar el arreglo de punteros al interpolar sus entradas (interpolación vecino más cercano e interpolación bilineal). Este arreglo de punteros es generado tan solo una vez, siempre y cuando no se varíen los parámetros de la cámara, el tamaño de la imagen panorámica y el tipo de interpolacíon.

Para realizar el *unwarping* de imágenes omnidireccionales con PMPA tan solo se debe de reasignar la nueva imagen omnidireccional al buffer (Figura 1.5 a), puesto que la tabla mapeo ya tiene los punteros correspondientes de este buffer para generar la imagen panorámica (se disminuye el tiempo de generación de la imagen).

Las ventaja de PMPA es los menores tiempos de procesamiento en comparación con el

método de tabla de mapeo panorámico y del método de un octavo de tabla de mapeo panorámico. En [23] se describe que PMPA es 5.8 (interpolación vecino más cercano) y 2.1 (interpolación bilineal) veces más rápido que el método de tabla de mapeo panorámico y 1.7 veces más rápido que el método de un octavo de tabla de mapeo panorámico para interpolación vecino más cercano. Otra ventaja es que los pasos descritos anteriormente para generar el arreglo de punteros tan solo se realizan una vez, en otras palabras, se reduce el número de tareas para realizar el *unwarping* de imagen a imagen.

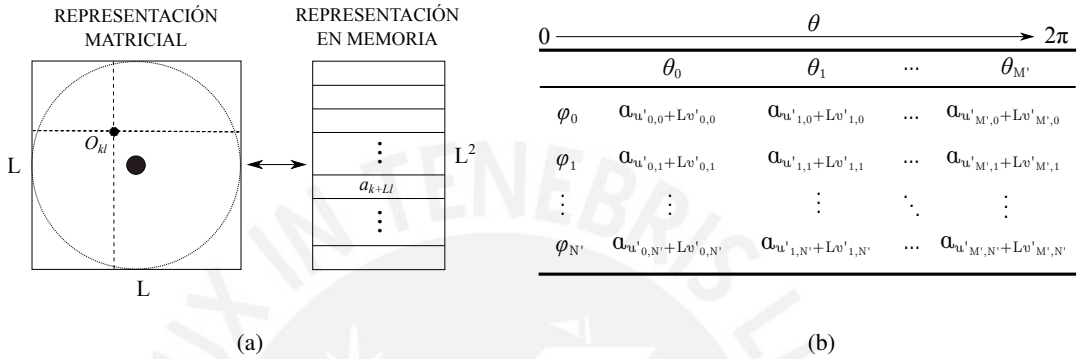

Figura 1.5: (a) Representación del buffer de la imagen omnidireccional y (b) Ejemplo del arreglo de punteros de mapeo panorámica [23].

Capítulo 2

CUDA (Compute Unified Device Architecture)

2.1. Definición

CUDA [24] es una plataforma y modelo de programación de cálculo en paralelo inventado por Nvidia en el año 2006 para aprovechar la gran capacidad de procesamiento de los GPUs (unidades de procesamiento gráfico) de la propia compañia. Esta cuenta con una extensión de lenguaje de programción de alto nivel c y c++, la cual permite al programador controlar los recursos de los GPUs (comunicación entre hilos, sincronización entre hilos, accesos a memoria, etc.) para realizar tareas de alto costo computacional. Las ventajas de CUDA son el aumento de potencial de procesamiento en comparación con los CPUs convencionales, el número de recursos a su disponibilidad como múltiples hilos, tipos de memoria, etc., y un mejor control de estos recursos en comparación con otros GPUs. Asimismo, CUDA permite disminuir los tiempos de procesamiento de grandes cantidades de datos al realizar múltiples cálculos en paralelo (gran potencial de procesamiento de las tarjetas gráficas Nvidia). Sin embargo, el procesamiento realizado en las tarjetas gráficas suele tener un elevado consumo energético y más en accesos a memoria (la arquitectura Maxwell tiene un menor consumo energético). Otra desventaja es la transferencia de datos entre CPU y GPU ya que es lenta y consume mayores recursos energéticos. Una nueva herramienta en el modelo de memoria (memoria unificada) que será explicada más adelante permite reducir los tiempos de transferencias [30].

Para entender acerca de CUDA y los recursos de los GPUs Nvidia, los cuales serán analizados y utilizados en la tesis para el diseño del algoritmo propuesto en la plataforma Jetson TK1 (Capítulo 3) se explicará en el presente capítulo el modelo de programación, los

tipos de memorias y las características de la plataforma Jetson TK1.

2.2. Modelo de Programación

El modelo de programación CUDA [24] esta diseñado para aprovechar el gran potencial de paralelización de los GPU Nvidia. Este modelo permite al programador seguir una serie pasos en lenguaje C para su utilización (inicializar *kernels*, seleccionar la memorias, realizar transferencias, etc.). El modelo abarca cuatro puntos importantes [24], los cuales son el *kernel*, la jerarquía de hilos, la jerarquía de memorias y capacidad de cómputo.

El *kernel* es el proceso o función principal que se ejecuta en el GPU. Su configuración de parámetros entrada, salida y tamaño de los bloques se realiza en el CPU al igual que su creación. Con respecto jerarquía de hilos, los bloques están conformados por una cantidad de N de hilos, los cuales son de 1 a 3 dimensiones. Cada hilo ejecuta una operación o *kernel* de forma paralela. Asimismo, los bloques también pueden estar distribuidos de 1 a 3 dimensiones y conforman una grilla. Cuando se genera un *kernel* se especifica el tamaño de la grilla y el tamaño de los bloques. Por otro lado, la capacidad de cómputo es un indicador de los recursos y funcionabilidades que tiene el GPU. Finalmente, la jeraquía de memorias se explica a partir de modelo de memoria que se presenta a continuación.

2.3. Modelo de Memoria

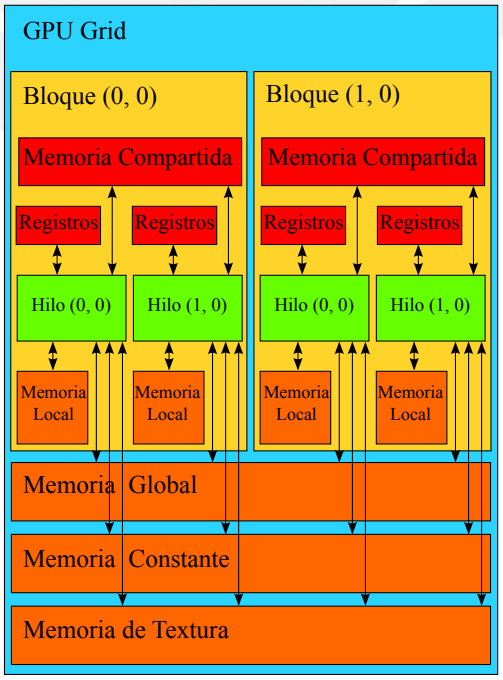

Figura 2.1: Modelo de memoria [25].

El modelo de memoria [24, 25] de los GPUs Nvidia se muestra en la Figura 2.1, en ella se puede observar los diferentes tipos de memoria disponibles. Para cada tipo de memoria hay diferentes ventajas y desventajas con respecto a los tiempos de latencia, las capacidades de memoria y los tiempos de vida. Con el fin de realizar decisiones adecuadas en el Capítulo 3 con respecto a donde colocar los datos en el diseño de la implementación paralela, se debe comprender los tipos de memoria del GPU y como estos afectan el rendimiento computacional del diseño. Por ello, a continuación se detallara cada tipo de memoria dividida en dos grupos [25]: memorias del dispositivo y memorias On-Chip.

2.3.1. Memorias del Dispositivo

2.3.1.1. Memoria Global

Memoria global es una memoria de lectura y escritura accesible para todos los hilos. Se caracteriza por ser la memoria más grande con la que cuenta el GPU, tener una vida útil hasta que la aplicación concluya y tener el mayor tiempo de latencia por acceso memoria (100 veces más lenta que la memoria compartida [31]). Un punto importante cuando se utiliza memoria global es realizar accesos colaborativos, los cuales permiten maximizar el ancho de banda hacia la memoria (maximizar el rendimiento computacional [24]).

2.3.1.2. Memoria Local

La memoria local es una memoria de lectura y escritura con elevados tiempos de latencia y bajo ancho de banda iguales a la memoria global [24]. Los espacios de la memoria son asignados automáticamente durante la ejecución de los *kernel* al tener variables que no encajan en los registros. Asimismo, la memoria está sujeta a los requerimientos de accesos colaborativos [24] para maximizar su rendimiento, sin embargo, estos se realizan de forma automática.

2.3.1.3. Memoria Constante

La memoria constante es una memoria de solo lectura accesible para todos los hilos. Se caracteriza por tener un cantidad limitada de memoria, pero tiempos de latencia por acceso menores a los de la memoria global [24]. La memoria constante cuenta con una cache (on-chip), la cual oculta los tiempos de latencia por acceso a memoria constante y la gestión de memoria (se realiza de forma automática).

2.3.1.4. Memoria de Textura

La memoria de textura es una memoria de solo lectura accesible para todos los hilos. Se caracteriza por tener una cache (on-chip), que al igual que la cache constante esta oculta los tiempos de latencia por acceso a memoria de textura. Sin embargo, la cache esta optimizada solo para textura de dos dimensiones.

2.3.2. Memorias On-chip

2.3.2.1. Memoria Compartida

Memoria compartida es una memoria de escritura y lectura accesible para los hilos por bloque. Cada SMX [24] tiene un limitado espacio de memoria compartida, pero los tiempos de latencia por acceso son mucho menores a memoria global (100 veces más rápido que memoria global [31]). Puesto a que la memoria compartida tiene un mejor rendimiento en acceso a memoria, se suele trabajar con este tipo de memoria para reducir costos computacionales por procesamiento en el *kernel*. Asimismo, tiene un tiempo de vida mientras se ejecuta el bloque.

2.3.2.2. Registros

Los registros son espacios de memoria de lectura y escrita con tiempos de acceso muy rápidos, pero con cantidades limitadas de espacio por bloque. Esto son independientes para cada hilo (privacidad a nivel de hilos) y su tiempo de vida solo se da mientras que los hilos se encuentren ejecutándoce. Asimismo, en un *kernel* las variables que son declaradas por defecto son registros.

2.3.3. Memoria Unificada

Memoria unificada [24] es una nueva mejora en el modelo de programación CUDA, que fue introducido a partir de abril del 2014. Memoria unificada consiste en que la memoria de CPU y del GPU son una solo memoria vista desde la perspectiva del programador. Físicamente las memorias son independientes y se encuentran separadas por el bus PCI-Express, pero mediante este nuevo modelo se puede trabajar como una memoria en común que desliga las transferencia y asignaciones de datos realizados por el programador de forma manual por una administración automática (manejo de memoria detrás de la perspectiva del programador). Sin embargo, existe una plataforma móvil conocida como Jetson TK1, la cual cuenta con memoria unificada física. La administración de memoria unificada es accesible por el CPU y GPU mediante un puntero en común, que permite que la data se traslade automáticamente entre CPU y GPU. Asimismo,

el puntero permite que cuando se ejecuta un código en CPU se perciba que se está accediendo a la data en la memoria del CPU y cuando se está ejecutando un *kernel* el puntero pase a disposición de este y permite tener el control de los datos de forma automática permitiendo percibir que la data se encuentra en el GPU. En la Figura 2.2 se puede observar el antiguo modelo de memoria (memorias separadas) y el nuevo modelo de memoria (memoria unificada) visto desde la perspectiva del programador.

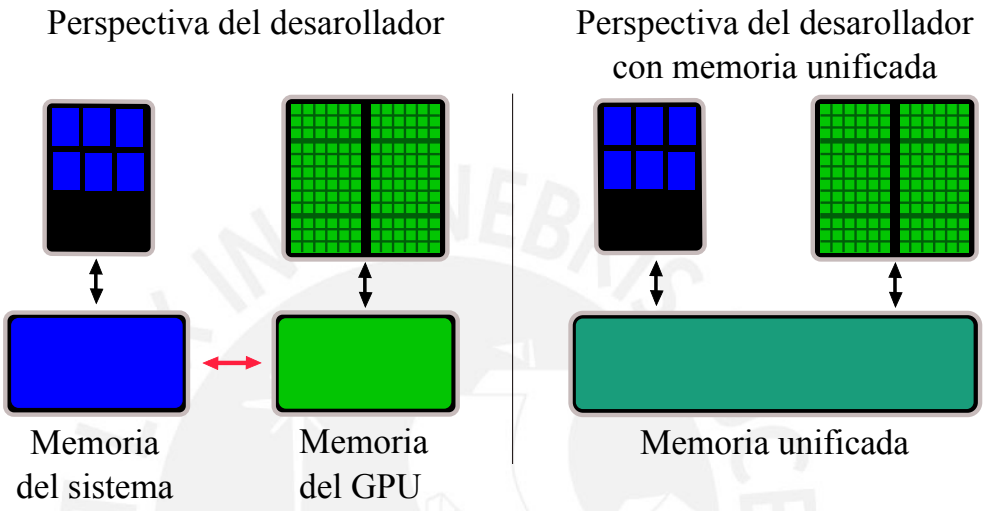

Figura 2.2: Modelo de memoria unificada [30].

Memoria unificada tiene como ventaja la simplicidad del modelo de memoria y programación, ya que el programador ya no debe de preocuparse por realizar las transferencias de memoria entre el CPU y el GPU. Asimismo, este modelo de memoria maximiza el rendimiento computacional de las transferencias de memoria y el manejo de memorias locales.

Para observar las ventajas de la memoria unificada sobre el procesamiento y las transferencias de datos de forma clásica (declaración y transferencia de datos de forma manual) se realizó un ejemplo de suma de dos vectores. El ejemplo (Figura 2.3) muestra las asignaciones y transferencias de datos realizadas mediante el modelo antiguo (transferencia de datos manuales) y mediante el nuevo modelo (memoria unificada). Como se puede observar en la Figura 2.3 el código con memoria unificada facilita y disminuye la complejidad de programación en cuanto a la cantidad de instrucciones necesarias para la utilización de memoria.

Codigo tradicional

```
void example(void *data){
CUDA_UOP *parse = ((CUDA_UOP *) data);
float *vector_A, *vector_B, *vector_C;

cudaMalloc((void **)&vector_A, ...);
cudaMalloc((void **)&vector_B, ...);
cudaMalloc((void **)&vector_C, ...);

cudaMemcpy( vector_A, parse->A, parse->nBytes,
cudaMemcpyHostToDevice);
cudaMemcpy( vector_B, parse->B, parse->nBytes,
cudaMemcpyHostToDevice);

Suma_Vectores<<< ... >>>(vector_A, vector_ B,
                         vector_ C);

cudaMemcpy( parse->C, vector_C, parse->nBytes,
cudaMemcpyDeviceToHost);

}
```

Codigo con memoria unificada

```
void example(void *data){
CUDA_UOP *parse = ((CUDA_UOP *) data);
float *vector_A, *vector_B, *vector_C;

cudaMallocManaged((void **)&vector_A, ...);
cudaMallocManaged((void **)&vector_B, ...);
cudaMallocManaged((void **)&vector_C, ...);

Suma_Vectores<<< ... >>>(vector_A, vector_ B,
                         vector_ C);

cudaDeviceSynchronize();



}
```

Figura 2.3: Ejemplo de memoria unificada para la suma de dos vectores.

2.4. Plataforma Jetson TK1

La plataforma Jetson TK1 [27] es una plataforma de desarrollo que utiliza el procesador móvil Tegra K1. Esta plataforma permite a los desarrolladores poder evaluar el gran potencial de procesamiento del procesador TK1. La plataforma se caracteriza por su reducido tamaño (5”x5”), alto rendimiento computacional (mayor a 300 GFLOPS), bajo consumo de energía por procesamiento (menor a 12.3 Watts) y sus puertos para periféricos. Estas características permiten que la plataforma sea utilizada en aplicaciones móviles o embebidas para áreas como visión robótica, medicina, redes de seguridad, etc.

El procesador Tegra K1 [28] es el más avanzado procesador móvil creado por Nvidia. Este se caracteriza por tener 192 núcleos CUDA de arquitectura Nvidia Kepler de igual rendimiento que los GPUs de las supercomputadoras. La arquitectura Nvidia Kepler permite al procesador tener un alto rendimiento computacional (mayor a 300 GFLOPS) y un bajo consumo energético (menor a 5 Watts). La arquitectura del procesador [27] se caracteriza por estar conformada por 4+1 cortex A15 (arquitectura CPU con gran rendimiento y control energético), GPU con arquitectura Kepler (192 núcleos CUDA que ofrecen gran potencial de procesamiento y una capacidad de cómputo de 3.2), dual core ISP (1.2 Giga pixeles por segundo de capacidad de procesamiento) y avanzado motor de vizualización (manejo simultáneo de pantalla local y 4k a 4k monitores externos por HDMI).

2.5. Objetivos

Con la utilización de la plataforma Jetson TK1 se plantea como objetivo principal implementar el *unwarping* de videos omnidireccional. Asimismo, se tiene como objetivo específico comparar y mejorar en tiempos de procesamiento con respecto al método PMPA. Para llevarlo a cabo primero se implementará una aplicación en el software MATLAB capaz de realizar el *unwarping* de imágenes omnidireccionales, para luego implementar una aplicación CUDA (implementación paralela computacionalmente eficiente). Ambas implementaciones en MATLAB y CUDA también son objetivos específicos.

Capítulo 3

Diseño del Algoritmo Propuesto

3.1. Consideraciones de Diseño

1. **Tamaño de la imagen:** La imágenes utilizadas en el diseño del algoritmo propuesto son de tamaño L×L para imágenes omnidireccionales y de M×N para imágenes panorámicas generadas. Se trabajara con imágenes omnidireccionales de un rango de 512×512 a 4096×4096 (tamaños estándar para las aplicaciones mencionadas anteriormente). Sin embargo, los tamaños de imágenes panorámicas con las que se puede trabajar están limitadas por capacidad de memoria de la plataforma.

2. **Capacidad de cómputo:** La capacidad de cómputo es un factor muy importante cuando se diseña un algoritmo CUDA puesto que esta es un indicador de la capacidad de recursos que tiene el dispositivo. La plataforma Jetson TK1 cuenta con una capacidad de cómputo de 3.2. Este indicador permite tener disponible el recurso memoria unificada, la cual permite realizar transferencias de datos de forma más eficiente.

3. **Tipos de memoria del GPU:** En el diseño de un algoritmo se debe de priorizar la reducción de accesos a memoria global, para ello en el diseño se tiene como principal objetivo utilizar memoria compartida y/o de textura para reducir los tiempo de latencia por acceso a memoria.

3.2. Diagrama de Bloques

En la Figura 3.1 se presenta el diagrama de bloques del método propuesto. Como argumentos de entrada se recibe una imagen omnidireccional y los parámetros intrínsecos de la cámara. Estos parámetros son calculados en la etapa de calibración de la cámara, pero para efectos de la tesis solo serán datos de entradas precalculados. En las etapas siguientes, primero

se obtiene una tabla de mapeo panorámico (mediante la etapa de calibración y proyección). Luego se aplica una interpolación (interpolación de vecino más cercano o interpolación bilineal) a la tabla de mapeo panorámico para obtener la tabla de desplazamientos de mapeo panorámico. Por último, se utiliza la tabla de desplazamientos que describe la posición de los datos necesarios para la generación de la imagen panorámica.

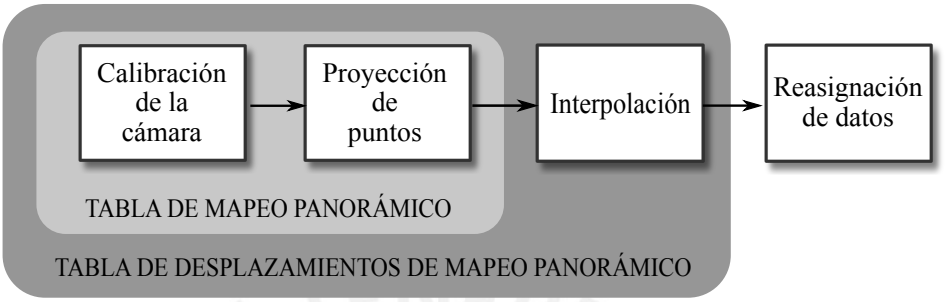

Figura 3.1: Diagrama de bloques del método de unwarping propuesto.

3.3. Descripción del Algoritmo Propuesto

El algoritmo propuesto consta de cuatro etapas: calibración de la cámara, proyección de puntos, interpolación, y reasignación de datos (etapa de *unwarping*). A continuación se detallará cada una de estas etapas.

3.3.1. Calibración de la cámara

En esta etapa se calculan los parámetros intrínsecos de la cámara, los cuales son necesarios para el modelo de proyección de la siguiente etapa. Sin embargo, cabe recalcar que esta etapa no es implementada en la tesis, sino que tan solo se consideran a los parámetros intrínsecos de la cámara como datos de entrada para el proceso de *unwarping*.

Parámetros intrínsecos de la cámara [32]: son los parámetros inherentes a ella que definen la óptica y la geometría interna de la cámara. Son constantes en tanto no varíen las características y posiciones relativas entre la óptica y el sensor. Estos parámetros son los siguientes:

- Distancia focal: La distancia focal (γ_u , γ_v) de una cámara es la distancia existente entre ésta y su foco.
- Punto principal: Son las coordenadas (u_o,v_o) del punto central del plano de la cámara.
- Coeficiente de inclinación (α): Es el parámetro que describe la inclinación del eje u y del eje v.

3.3.2. Proyección de puntos

La proyección de la imagen omnidireccional permite obtener la relación de coordenadas entre el punto del mundo real y su correspondiente punto en la imagen omnidireccional. El punto en el mundo real tiene como par ordenado de coordenadas esféricas (θ, φ), donde θ representa el ángulo azimuth y φ el ángulo de elevación ($\theta \in [0°, 360°]$ y $\varphi \in [70°, -20°]$). El punto en la imagen omnidireccional tiene como par ordenado de coordenadas cartesianas (u,v).

Para esta etapa se utilizará el modelo de proyección unificada [33] ya que su simplicidad permite que sea paralelizable de forma práctica y que se pueda evaluar de forma más sencilla el método propuesto. Además, se pueden utilizar otros tipos de proyección como proyección cúbica [35], cilíndrica [16], etc., ya que el método propuesto se adapta al tipo de proyección sin interferir con las demas etapas.

El modelo de proyección unificada se muestra en la Figura 3.2 y consta de una serie de etapas que son descritas a continuación.

O_r es un punto dentro de mundo real el cual se encuentra ubicado en [X, Y, Z]^T = [cos(θ), sen(θ), tan(φ)]^T.

$$O_r = \begin{bmatrix} cos(\theta) \\ sen(\theta) \\ tan(\varphi) \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.1}$$

Se proyecta el punto del mundo real O_r sobre la esfera unitaria generando un nuevo punto O_S con centro de coordenadas C_m.

$$O_S = \frac{O_r}{||O_r||} = \begin{bmatrix} X_S \\ Y_S \\ Z_S \end{bmatrix} \tag{3.2}$$

El punto O_S es ubicado en un nuevo centro de coordenada $C_p = \begin{bmatrix} 0 & 0 & \zeta \end{bmatrix}^T$ (ζ es la distancia entre el centro de la esfera unitaria C_m y el punto focal del espejo de la cámara omnidireccional). Luego ello, se mapea dentro de un plano normalizado.

$$O_m = \begin{bmatrix} \frac{X_S}{Z_S+\zeta} \\ \frac{Y_S}{Z_S+\zeta} \\ \frac{Z_S+\zeta}{Z_S+\zeta} \end{bmatrix} = \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} \tag{3.3}$$

Finalmente, se calcula la proyección de la cámara multiplicando la matriz de calibración H con el punto O_m. Esta multiplicación $H \times O_m$ genera como resultado el punto O perteneciente a la imagen omnidireccional. Los parámetros de la matriz H fueron descritos en la etapa de calibración.

$$O = H \times O_m = \begin{bmatrix} \gamma_u & \alpha & u_0 \\ 0 & \gamma_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.4}$$

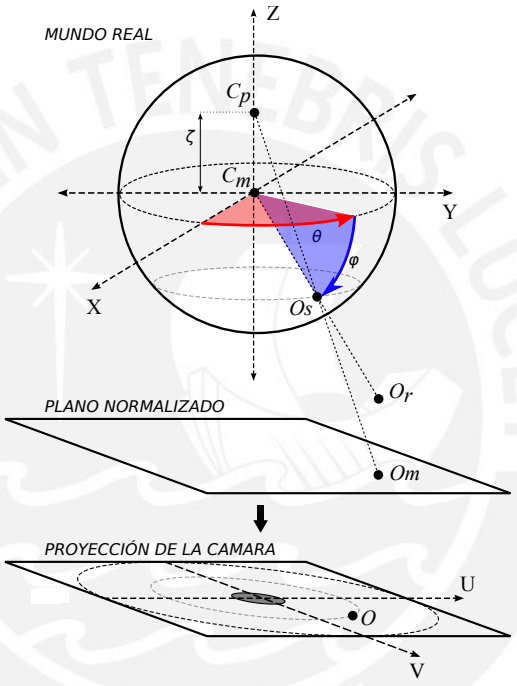

Figura 3.2: Modelo de proyección unificada.

Mediante la correspondencia de coordenadas esféricas y coordenadas cartesianas de los puntos en el mundo real y en la imagen omnidireccional se prosigue a generar la tabla de mapeo panorámico, la cual fue explicada en capítulos anteriores. Para ello, se discretiza los ángulos θ y φ en M y N muestra respectivamente. Las cantidad de muestras va a definir la resolución de la imagen panorámica resultante (imagen de N filas por M columnas).

$$\begin{aligned} \theta_i &= \frac{2\pi i}{M} && i = 0, 1, ..., M-1 \\ \varphi_j &= \varphi_{max} - \frac{-(\varphi_{max} - \varphi_{min})j}{N} && j = 0, 1, ..., N-1 \end{aligned} \tag{3.5}$$

La tabla de mapeo panorámico (Tabla 3.1) relaciona los puntos del mundo real Or_{ij} con coordenadas (θ_i y φ_j) con los puntos de la imagen omnidireccional O_{ij} con coordenadas (u_{ij} y v_{ij}). Las entradas de la tabla están defininidas como $E_{ij} = (u_{ij}, v_{ij})$.

$0 \longrightarrow \theta \longrightarrow 2\pi$				
	θ_0	θ_1	$\cdots$	$\theta_{M'}$
φ_0	$(u_{0,0},v_{0,0})$	$(u_{1,0},v_{1,0})$	$\cdots$	$(u_{M',0},v_{M',0})$
φ_1	$(u_{0,1},v_{0,1})$	$(u_{1,1},v_{1,1})$	$\cdots$	$(u_{M',1},v_{M',1})$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\varphi_{N'}$	$(u_{0,N'},v_{0,N'})$	$(u_{1,N'},v_{1,N'})$	$\cdots$	$(u_{M',N'},v_{M',N'})$

Tabla 3.1: Ejemplo de tabla de mapeo panorámico, en donde M' = M - 1 y N' = N - 1.

3.3.3. Interpolación

En este paso se genera la tabla de desplazamientos de mapeo panorámico (Tabla 3.2) utilizando los datos de la tabla de mapeo panorámico (Tabla 3.1). Los valores de la tabla de mapeo panorámico (entradas E_{ij}) no son valores enteros, los cuales son necesarios para la tabla de desplazamientos de mapeo panorámico (PMOT). Para ello, se interpola los valores de las entradas E_{ij} antes de generar la PMOT. En el método propuesto se trabaja con dos tipos de interpolación (intepolación vecino más cercano y bilineal) que fueron seleccionadas simplemente para tener las mismas referencias (tipos de interpolación) que el método PMPA, el cual será comparado en tiempos computacionales en el Capítulo 4.

Para la implementación de la interpolación vecino más cercano f(x) = round(x) y para la interpolación bilineal f(x) = floor(x):

$$u'_{ij} = f(u_{ij}), \qquad v'_{ij} = f(v_{ij}) \tag{3.6}$$

PMOT es una alternativa computacionalmente eficiente a la tabla de mapeo panorámico. Las entradas de PMOT definidas como S_{ij} permiten generar los puntos de la imagen panorámica mediante desplazamientos. Por otro lado, cabe recalcar que la tabla de desplazamientos (PMOT) se genera una sola vez (los procesos para generarla se realizan una vez), siempre y cuando no se varíen los parámetros de la cámara, el tamaño de las imágenes y el tipo de interpolación utilizada.

$$S_{ij} = u'_{ij} + v'_{ij} \cdot L \tag{3.7}$$

$0 \longrightarrow \theta \longrightarrow 2\pi$

	θ_0	θ_1	...	$\theta_{M'}$
φ_0	$u'_{0,0}+Lv'_{0,0}$	$u'_{1,0}+Lv'_{1,0}$	...	$u'_{M',0}+Lv'_{M',0}$
φ_1	$u'_{0,1}+Lv'_{0,1}$	$u'_{1,1}+Lv'_{1,1}$	...	$u'_{M',1}+Lv'_{M',1}$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\varphi_{N'}$	$u'_{0,N'}+Lv'_{0,N'}$	$u'_{1,N'}+Lv'_{1,N'}$	...	$u'_{M',N'}+Lv'_{M',N'}$

Tabla 3.2: Ejemplo de tabla de desplazamientos de mapeo panorámico, en donde M' = M - 1 y N' = N - 1.

3.3.4. Reasignación de datos

Esta etapa consiste en reasignar los datos de la imagen omnidireccional para generar la imagen panorámica como se observa en la Figura 3.3, en otras palabras, se realiza el *unwarping* de imágenes omnidireccionales. Dependiendo del tipo de interpolación utilizado se realizará una diferente reasignación de datos. Para el caso de interpolación de vecino más cercano, la reasignación de datos consiste en asignar de forma directa los datos de la tabla de desplazamientos ($P_{ij} = O[S_{ij}]$).

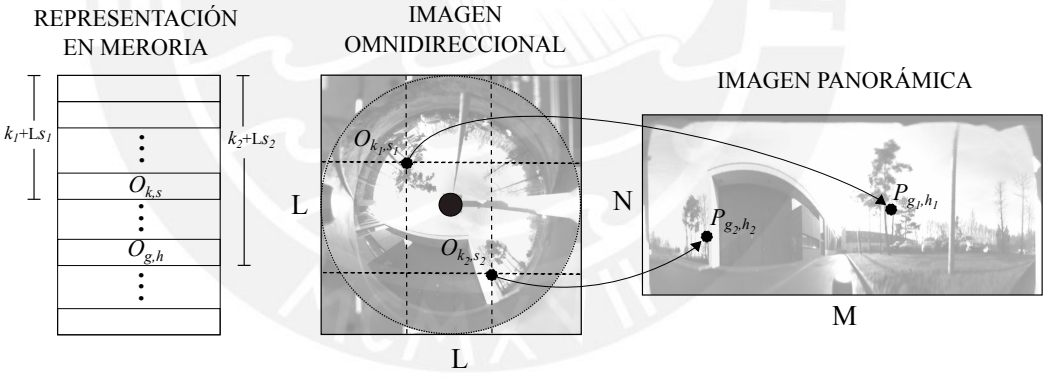

Figura 3.3: Relación entre la imagen omnidireccional e imagen panorámica mediante desplazamientos en memoria.

Para el caso de interpolación bilineal [36] se calculan cuatro pesos correspondiente a la vecindad del punto que se desea determinar, el cual es una entrada de la PMOT (Tabla 3.2). A partir del cálculo de todos los puntos se generan cuatro matrices de pesos (W1, W2, W3 y W4).

$$
\begin{aligned}
W1_{ij} &= (1-(u_{ij}-u'_{ij}))\times(1-(v_{ij}-v'_{ij})) & S1 &= S_{ij} = u'_{ij} + v'_{ij}L \\
W2_{ij} &= (u_{ij}-u'_{ij})\times(1-(v_{ij}-v'_{ij})) & S2 &= u'_{ij} + v'_{ij}L + 1 \\
W3_{ij} &= (1-(u_{ij}-u'_{ij}))\times(v_{ij}-v'_{ij}) & S3 &= u'_{ij} + v'_{ij}L + L \\
W4_{ij} &= (u_{ij}-u'_{ij})\times(v_{ij}-v'_{ij}) & S4 &= u'_{ij} + v'_{ij}L + L + 1
\end{aligned} \qquad (3.8)
$$

Para este tipo de interpolación la reasignación de datos consiste en generar los puntos de la imagen panorámica utilizando las cuatro matrices creadas anteriormente (W1, W2, W3 y W4) y los cuatro valores (S1, S2, S3 y S4) calculados a partir de la PMOT. La reasignación de datos se realiza al calcular la ecuación 3.9. Por otro lado, en la Figura 3.4 se observa de manera gráfica, que el cálculo de los puntos de la imagen panorámica (resignación de datos) consiste en multiplicar los cuatro pesos con sus correspondientes pixeles de la imagen omnidireccional (pixeles que corresponde a la vecindad de cada punto de la tabla PMOT).

$$P_{ij} = W1_{ij} \cdot O[S1] + W2_{ij} \cdot O[S2] + W3_{ij} \cdot O[S3] + W4_{ij} \cdot O[S4] \tag{3.9}$$

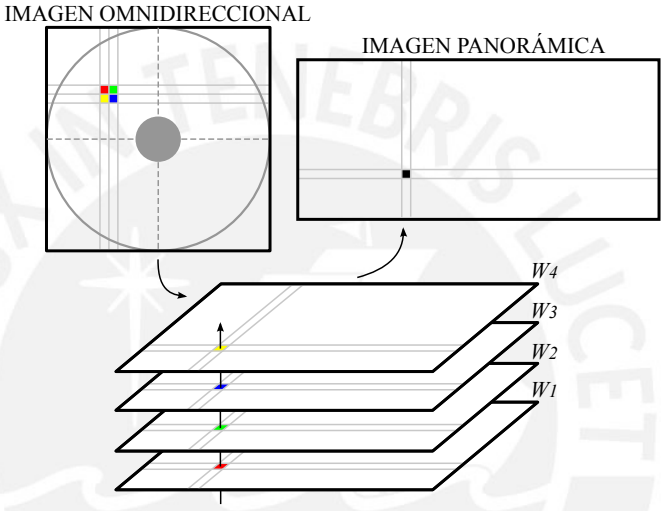

Figura 3.4: Ejemplo de reasignación de datos para interpolación bilineal.

El proceso de reasignación de datos que se realiza mediante la PMOT para interpolación de vecino más cercano y bilineal puede ser representada como una multiplicación matriz vector ($W \times O = P$), donde W es una matriz dispersa (contiene mayor número de ceros). En la Figura 3.5 se observa la representación del unwarping para los tipos de interpolaciones utilizados como una multiplicación matriz vector.

$$\begin{bmatrix}
0 & W1_1 & W2_1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & W1_2 & W2_2 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & W1_3 & W2_3 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & 0 & W1_4 & W2_4 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & W1_{n-3} & W2_{n-3} & 0 & \dots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & W1_{n-2} & \dots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & W1_n & W2_n & \dots & 0
\end{bmatrix}
\times
\begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \\ \vdots \\ Om_{-4} \\ Om_{-3} \\ Om_{-2} \\ Om_{-1} \\ O_m \end{bmatrix}
=
\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ \vdots \\ P_{n-4} \\ P_{n-3} \\ P_{n-2} \\ P_{n-1} \\ P_n \end{bmatrix}$$

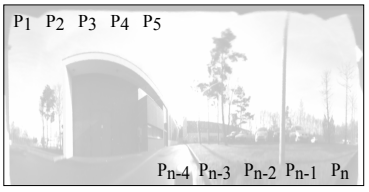

Figura 3.5: Representación de matriz dispersa para relacionar la imagen omnidireccional y la imagen panorámica, donde m = L × L y n = M × N.

3.4. Descripción del Diseño Paralelo del Algoritmo Propuesto

Antes de diseñar la implementación paralela en el modelo de programación CUDA, se describirá el algoritmo serial para tener una base con la cual partir. En el siguiente algoritmo 1 se describe de forma general los pasos que involucran el método propuesto.

Algorithm 1: Método propuesto de unwarping imágenes omnidireccionales.

Entrada : Imagen Omnidireccional, parámetros intrínsecos de la cámara

Salida : Imagen Panorámica

Paso 1 — Proyección

Cálculo del punto en el mundo real ([X, Y, Z] = [cos(θ), sen(θ), tan(φ)]).

$X_s = X/||X||;$ $Y_s = Y/||Y||;$ $Z_s = Z/||Z||;$

$X_m = X_s/(Z_s + \zeta);$ $Y_m = Y_s/(Z_s + \zeta);$

$u = X_m.\gamma_u + Y_m.\alpha + u_0;$ $v = Y_m.\gamma_v + v_0;$

Paso 2 — Interpolación

if Intepolación = vecino más cercano **then**
  PMOT = round(u) + L · round(v);

else
  % Interpolación = bilineal
  Se calcula los pesos W1, W2, W3, W4 de la forma descrita en la ecuación 3.8.
  W = [W1', W2', W3', W4'];
  S1 = floor(u) + L · floor(v); S2 = floor(u) + L · floor(v) + 1;
  S3 = floor(u) + L · floor(v) + L; S4 = floor(u) + L · floor(v) + L + 1;
  PMOT = [S1, S2, S3, S4];

Paso 3 — Reasignación de dato (unwarping)

if Intepolación = vecino más cercano **then**
  P = O[PMTO];

else
  % Interpolación = bilineal
  P = (W[1] · O[PMTO[1]]) + (W[2] · O[PMTO[2]]) + (W[3] · O[PMTO[3]]) +
  (W[4] · O[PMTO[4]]);

3.4.1. Requerimientos de Memoria

Mediante el algoritmo 1 se evalúan las espacios de memoria necesarios para las variables y datos que utiliza el método propuesto. Con el cálculo de espacios de memoria necesarios y las características de las capacidades de memoria con las que cuenta la plataforma TK1 se conocerá el máximo tamaño de datos que se puede procesar con el algoritmo paralelo

propuesto para la plataforma. Asimismo, se evitarán conflictos en exceso de memoria por un mal diseño. En la Tabla 3.3 se muestra la cantidad de espacio de memoria necesaria para el algoritmo paralelo. Como se observa en la Tabla 3.3 el total de espacio de memoria necesaria dependerá más del tamaño de la imagen panorámica y del tipo de interpolación. Como se mencionó para no sobrepasar las capacidades de memoria disponibles en la TK1, lo que se realizará es calcular el tamaño máximo de imagen panorámica que se puede generar para los dos tipos de interpolación.

Tabla 3.3: Tabla de cantidades de espacio de memoria necesaria para el unwarping de imágenes omnidireccionales.

Variable	Interpolación vecino más cercano		Interpolación bilineal	
	Número de datos (flotante)	Cantidad de memoria (bytes)	Número de datos (flotante)	Cantidad de memoria (bytes)
I. Entrada	$L \times L$	$4 \times L \times L$	$L \times L$	$4 \times L \times L$
I. Salida	$N \times M$	$4 \times N \times M$	$N \times M$	$4 \times N \times M$
X	M	$4 \times M$	M	$4 \times M$
Y	M	$4 \times M$	M	$4 \times M$
Z	N	$4 \times N$	N	$4 \times N$
PMOT	$N \times M$	$4 \times N \times M$	$4 \times N \times M$	$16 \times N \times M$
W			$4 \times N \times M$	$16 \times N \times M$

En la ecuación 3.10 se expresa el total de memoria necesaria para interpolación vecino más cercano y bilineal. Con el total de memoria necesaria por tipo de interpolación se calcula el máximo tamaño de las imágenes panorámicas. Para ello se asumirá que el total de espacio ocupado por las variables X, Y y Z, igual a $8 \times M + 4 \times N$, en los dos tipos de interpolación es despreciable. En la ecuación 3.11 se muestra el nuevo máximo de espacio de memoria utilizado.

$$\begin{aligned} \text{Interp. vecino más cercano} &= 4 \times L \times L + 8 \times N \times M + 8 \times M + 4 \times N \\ \text{Interp. bilineal} &= 4 \times L \times L + 36 \times N \times M + 8 \times M + 4 \times N \end{aligned} \quad (3.10)$$

$$\begin{aligned} \text{Interp. vecino más cercano} &\approx 4 \times L \times L + 8 \times N \times M \\ \text{Interp. bilineal} &\approx 4 \times L \times L + 36 \times N \times M \end{aligned} \quad (3.11)$$

En las ecuaciones 3.12 y 3.13 se calculan el máximo tamaño de imagen panorámica que se puede generar, conociendo que la plataforma Jetson TK1 cuenta con una máximo de 1746 Mbytes de memoria global, el tamaño de la imágenes panorámicas es N×M (M = 4×N) y el máximo tamaño de imagen omnidireccional con el cual se va a trabajar es 4096×4096 (máximo tamaño estándar utilizado en las aplicaciones mencionadas en capítulos anteriores).

$$\begin{aligned} \text{Interp. vecino más cercano} \approx 4 \times \mathrm{L} \times \mathrm{L} + 2 \times \mathrm{M}^2 &< 1746 \times 2^{20} \\ 4 \times 4096 \times 4096 + 2 \times \mathrm{M}^2 &< 1746 \times 2^{20} \\ \mathrm{M} &< 29696 \end{aligned} \tag{3.12}$$

$$\begin{aligned} \text{Interp. bilineal} \approx 4 \times \mathrm{L} \times \mathrm{L} + 9 \times \mathrm{M}^2 &< 1746 \times 2^{20} \\ 4 \times 4096 \times 4096 + 9 \times \mathrm{M}^2 &< 1746 \times 2^{20} \\ \mathrm{M} &< 13998,83 \end{aligned} \tag{3.13}$$

Con los cálculos hechos anteriormente se observa que el máximo tamaño de imagen panorámica que se puede generar para intepolación de vecino más cercano es menor a 7424×29696 y para interpolación bilineal es menor a 3499×13996.

3.4.2. Diseño Paralelo CUDA

Para el diseñar una implementación paralela se tuvo en cuenta las siguientes recomendaciones [26], las cuales permiten mejorar los resultados computacionales de la implementación.

- Reducir al mínimo la transferencia de datos entre el *host* y el dispositivo.
- Ajustar la configuración de lanzamiento del *kernel* para maximizar la utilización del dispositivo.
- Asegurarse que se realicen accesos colaborativos a memoria global.
- Minimizar accesos redundantes a la memoria global siempre que sea posible.

El diseño se basó en cumplir las recomendaciones mencionadas. Para la primera recomendación, lo que se realizó fue utilizar el nuevo modelo de memoria (memoria unificada) para reducir el tiempo de transferencias de datos entre el *host* y el dispositivo. Para la segunda recomendación, maximizar el rendimiento de los *kernel* mediante su configuración de bloque, lo que se realizó fue definir diferentes tamaños de bloques para poder evaluar y encontrar el tamaño de bloque óptimo. Las pruebas para hallar de forma empírica el tamaño de bloque óptimo se encuentran en el Capítulo 4. El diseño basado en las dos últimas recomendaciones se explicara a continuación.

3.4.2.1. Accesos a memoria

Para agilizar y maximizar el ancho de banda de accesos a memoria global se realizaran lecturas y escrituras a memoria con accesos colaborativos. Los accesos colaborativos para el diseño consiste en que cada hilo va acceder de forma ordenada a cada uno los espacios de 32 bits de la memoria global.

3.4.2.2. Tipos de memoria utilizadas

Como se mencionó en la sección 2.3 existen diferentes tipos de memoria en los GPUs Nvidia. Para obtener el mejor rendimiento computacional se remplazó el uso memoria global por memoria compartida y registros. Los tipos de memoria seleccionados tienen un menor tiempo de latencia de acceso a memoria y su utilización exacta se especificada en el diseño y descripción de las etapas. Por otro lado, se hizo uso de memoria global en etapas que necesitaban tener datos entre *kernels*, pero teniendo en cuenta accesos colaborativos.

3.4.2.3. Diseño y descripción de las etapas

El diseño CUDA fue dividido en tres etapas (*kernels*), en las cuales se consideraron tiempos de accesos a memoria, tipos de memoria y tareas que se pueden realizar en paralelo.

Etapa1: Cálculo de los puntos en el plano real

En esta parte se priorizo guardar los resultados de los puntos del plano real de forma ordenada (alineada) para tener accesos colaborativos a estos en la siguiente etapa. El proceso de la etapa consiste en que cada hilo realiza de forma independiente y paralela el cálculo de las coordenadas del puntos del plano real ([X, Y, Z] = [cos(θ), sen(θ), tan(φ)]) y cada hilo guarda su correspondiente dato de forma alineada en memoria. En la Figura 3.6 se puede observar el diseño del cálculo de puntos del mundo real para accesos colaborativos.

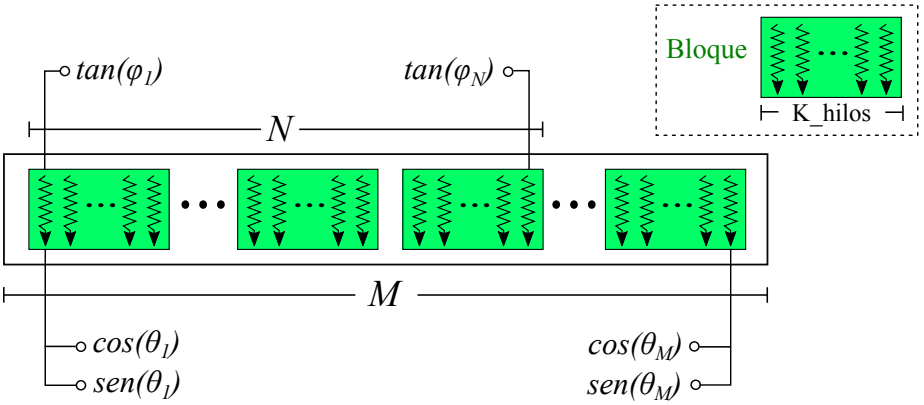

Figura 3.6: Cálculo de los puntos en el plano real.

Etapa2: Cálculo de la tabla de desplazamientos de mapeo panorámico

En esta etapa se trabajó con registros y memoria global para generar la tabla de desplazamiento y la matriz de pesos en el caso de interpolación bilineal. Ambas fueron guardas en memoria global de tal forma que lo hilos pueda acceder a la data de forma colaborativa. En la Figura 3.7 se puede observar el diseño para el cálculo de la tabla desplazamientos. En este diseño se define un bloque con hilos 2D para maximizar el rendimiento, en donde cada hilo realiza un pequeño cálculo que es almacenado en registros. Por último el resultado final (PMOT: tabla de desplazamientos, W: matriz de pesos) es almacenada de forma ordenada en memoria global para tener accesos colaborativos a memoria.

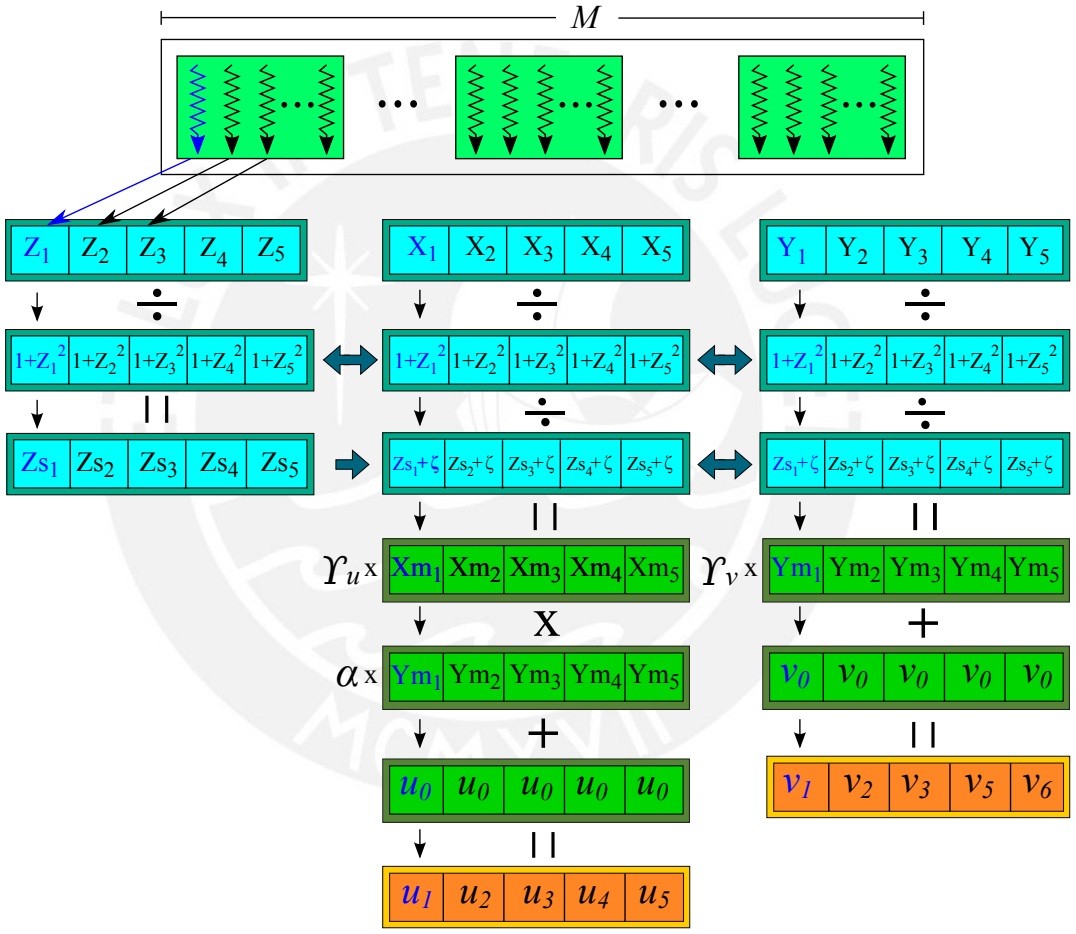

Figura 3.7: Cálculo de la tabla de desplazamientos de mapeo panorámico.

Etapa3: Reasignación de data (*unwarping*)

En esta etapa se plantean dos diseños, los cuales pertenecen a la interpolación vecino más cercano e interpolación bilineal. El diseño para la interpolación vecino más cercano consiste en tener la tabla de desplazamientos en la memoria de global alineado con su correspondientes hilos de tal forma de obtener accesos colaborativos (maximizar el ancho de banda por acceso). Como se observa en la Figura 3.8 cada hilo lee un dato de la tabla de desplazamientos y

mediante este accede a su correspondiente dato de la imagen omnidireccional, el cual es un dato de la imagen panorámica. Por otro lado, las transferencias de los datos de la imagen omnidireccional e imagen panorámica son realizados con el modelo de memoria unificada, con el propósito de minimizar los tiempo de latencia por transferencia de datos entre CPU y GPU.

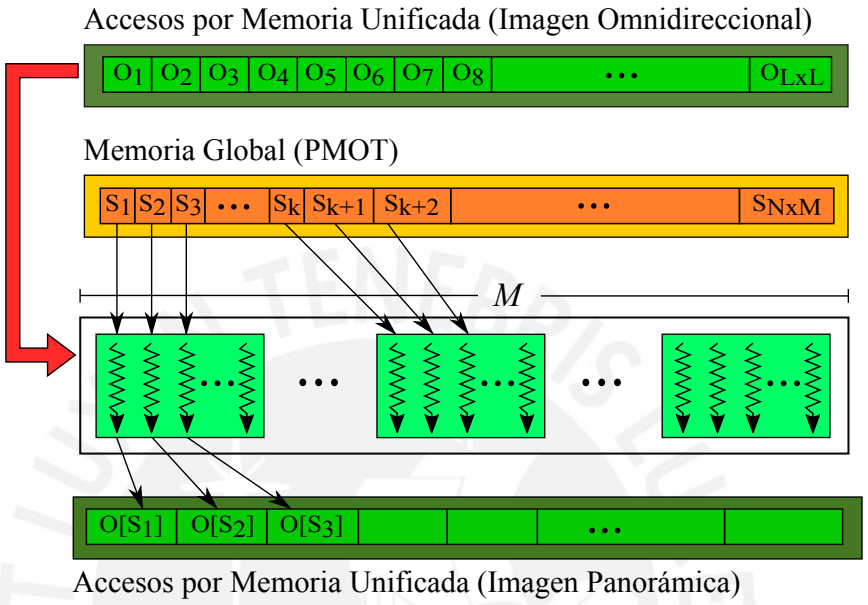

Figura 3.8: Cálculo de la imagen panorámica con interpolación vecino más cercano.

Para el diseño de la reasignación de datos por interpolación bilineal lo que se realizó fue acceder tanto a los datos de la imagen omnidireccional mediante memoria unificada, como de forma alineada a los datos de la tabla de desplazamiento y del arreglo de pesos en memoria global para obtener accesos colaborativos. Para maximizar el rendimiento computacional, lo que se planteo fue que cada hilo realizara una pequeña multiplicación en paralelo (involucra accesos a memoria global en paralelo) y realizar una reducción mediante memoria compartida. En la Figuras 3.9 y 3.10 se puede observar la lógica mencionada del diseño y los tipos de memoria utilizados. Lo que se realiza primero en esta etapa fue que cada hilo lea un dato de la tabla de desplazamiento y de la tabla de pesos. Luego de ello, se evalúa el desplazamiento para acceder al dato de imagen omnidireccional correspondiente. Se realiza un multiplicación entre el pesos correspondiente y el dato adquirido de la imagen omnidireccional. Cada cuatro datos resultantes de la multiplicación se realiza un reducción para calcular el pixel correspondiente que conforma a la imagen panorámica. La reducción realizada consiste en ir sumando datos continuos por hilo e ir guardando en memoria compartida (Figura 3.10). El tipo de dato *float* permite que no haya conflicto de bancos ya que cada hilo accederá a un banco de memoria diferente.

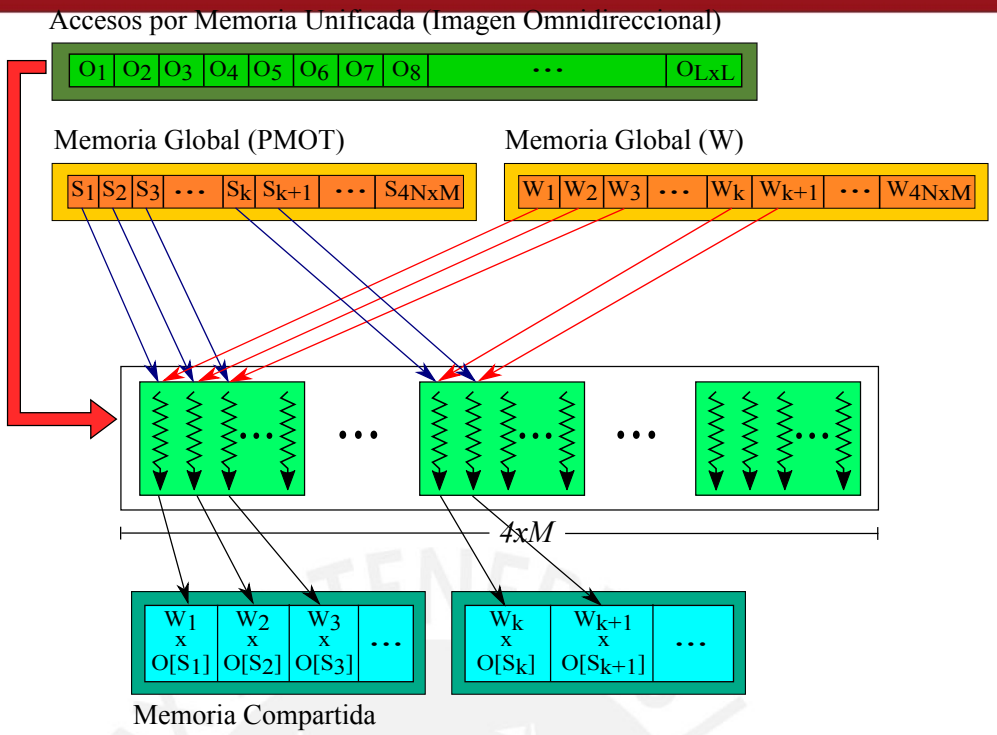

Figura 3.9: Accesos a memoria para la reasignación de datos por interpolación bilineal.

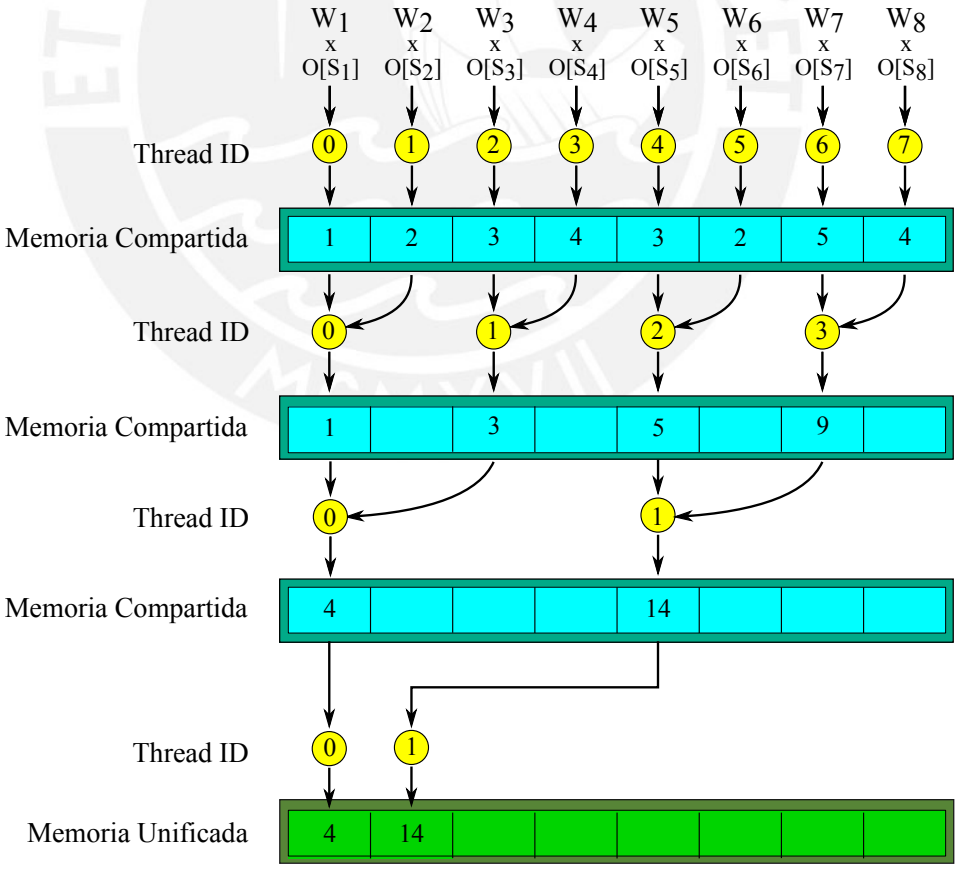

Figura 3.10: Reducción para el cálculo de la imagen panorámica con interpolación bilineal.

Capítulo 4

Implementación y Resultados Computacionales

El código de las implementaciones en MATLAB y CUDA del *unwarping* de imágenes omnidireccionales se encuentra en el CD adjuntado al documento.

4.1. Consideraciones de la Implementación

1. **Plataforma de implementación:** Se utilizará la plataforma Jetson TK1 de Nvidia, con un sistema operativo Linux para Tegra (L4T).

2. **Software y herramientas de desarrollo:** Se utilizó el entorno MATLAB® desarrollado por MathWorks en la versión 8.2, el kit de herramientas CUDA desarrollado por Nvidia en la versión 6.5 y el lenguaje ANSI-C con las herramientas de desarrollo propias del compilador GCC 4.4.

3. **Tamaño del bloque óptimo:** Se definieron diferentes tamaños de bloques para calcular de forma empírica el tamaño óptimo. El tamaño máximo de los bloques para las pruebas no será mayor a 1024 hilos por bloque ya que este es el número máximo de hilos por bloque con el que cuenta la plataforma (Procesador TK1).

4. **Características de las imágenes y datos:** Las imágenes omnidireccionales utilizadas para la implementación están en escala de grises y tiene resoluciones de 512×512 a 4096×4096 pixeles (medidas estándar). Se utilizan datos máximos con precisión simple ya que el procesador TK1 tiene una arquitectura de 32 bits, el cual permite un manejo eficiente de datos no mayores a 32 bits (precisión simple).

4.2. Decripción de la Implementación

Se realizó una implementación inicial en el entorno de programación MATLAB para verificar funcionalidad del algoritmo propuesto. La implementación final del algoritmo propuesto se realizó en el modelo de programación CUDA en una librería estática [38], de la cual se utilizó herramientas para la inicialización de variables y el cálculo de tiempos de procesamiento. La implementación utiliza como datos de entrada una imagen omnidireccional y los parámetros intrínsecos de la cámara en formatos RAW.

4.2.1. Implemetación en MATLAB

Para probar el método propuesto se realizó una implementación en el entorno de prográmación MATLAB, la cual realiza el *unwarping* de la imágenes omnidireccionales en imágenes panorámicas. Se utiliza la implementación MATLAB como referencia para comparar resultados en exactitud (error relativo entre los resultados del algoritmo MATLAB y algoritmo paralelo propuesto). El programa recibe como argumentos de entrada una imagen omnidireccional (formato RAW con datos en punto flotante), los parámetros intrínsecos de la cámara, el tipo de interpolación usada, el tamaño de la imagen omnidireccional y el tamaño de la imagen panorámica. Como salida devuelve una imagen panorámica con las características ingresadas (tamaño y tipo de interpolación). Los resultados de la implementación se muestran en la sección de pruebas del algoritmo paralelo propuesto (Figuras 4.1 y 4.2).

4.2.2. Implemetación Paralela CUDA

La implementación paralela en el modelo de programación CUDA al igual que la implementación MATLAB recibe los mismos datos de entrada, sin embargo, genera como salida la imagen panorámica correspondiente a los datos de entrada y el tiempo de procesamiento del *unwarping*. Los resultados de las imágenes panorámicas generadas se muestran en las Figuras 4.1 y 4.2. Los tiempos de procesamiento de la implementación paralela CUDA se muestran en la sección de resultados computacionales.

4.3. Resultados Computacionales

La pruebas se realizaron en la plataforma Jetson TK1 con 2.33 GHz de frecuencia de reloj de CPU, 852 MHz de frecuencia de reloj del GPU, 924 MHz de frecuencia de reloj de la memória, 2 GB de memoria DDR3L. El sistema operativo es Ubuntu 14.04 L4T de 32 bits, kernel 3.10.24 y escritorio XFCE.

4.3.1. Pruebas del algoritmo paralelo propuesto

En las Figuras 4.1 y 4.2 se muestran las imágenes desdobladas a partir de la implementación MATLAB y la implemntación paralela CUDA. En las figuras se puede observar que los resultados obtenidos mediante la implementación MATLAB y CUDA no se diferencian visualmente (visualmente son iguales). Sin embargo se debe evaluar la exactitud de los resultados para validar que la implementación paralela sea correcta.

(a) (b) (c)

Figura 4.1: (a) Imagen omnidireccional, (b) Unwarp realizado con la implementación MATLAB para el caso de interpolación vecino más cercano y (c) Unwarp realizado con la implementación CUDA para el caso de interpolación vecino más cercano.

(a) (b) (c)

Figura 4.2: (a) Imagen omnidireccional, (b) Unwarp realizado con la implementación MATLAB para el caso de interpolación bilineal y (c) Unwarp realizado con la implementación CUDA para el caso de interpolación bilineal.

Para evaluar la exactitud del algoritmo paralelo propuesto se calculó el error relativo ($||\text{valor}_{\text{medido}} - \text{valor}_{\text{real}}|| \ / \ ||\text{valor}_{\text{real}}||$) entre éste y la implementación de referencia (MATLAB). Los resultados de la Tabla 4.1 muestran el error relativo entre la implementación de referencia y la implementación paralela (CUDA) para el *unwarping* de imágenes omnidireccionales de $512\times512 \sim 4096\times4096$ en imágenes panorámicas de $128\times512 \sim$

1024×4096. Se observa que el error relativo para la interpolación vecino más cercano e interpolación bilineal (Tabla 4.1) están en el orden de 10^{-6}, esto indica que el *unwarping* de las imágenes omnidireccionales mediante la implementación paralela es correcta.

Tabla 4.1: Tabla de error relativo entre implementación CUDA e implementación MATLAB para interpolación vecino más cercano e interpolación bilineal.

Interpolación vecino más cercano		
Imagen Omnidireccional	Imagen Panorámica	Error Relativo
512×512	128×512	1,72 e-6
512×512	256×1024	2,06 e-6
512×512	512×2048	2,11 e-6
512×512	1024×4096	2,18 e-6
1024×1024	128×512	1,62 e-6
1024×1024	256×1024	2,05 e-6
1024×1024	512×2048	2,08 e-6
1024×1024	1024×4096	2,24 e-6
2048×2048	128×512	1,68 e-6
2048×2048	256×1024	2,02 e-6
2048×2048	512×2048	2,07 e-6
2048×2048	1024×4096	2,27 e-6
4096×4096	128×512	1,74 e-6
4096×4096	256×1024	2,01 e-6
4096×4096	512×2048	2,21 e-6
4096×4096	1024×4096	2,33 e-6

Interpolación bilineal		
Imagen Omnidireccional	Imagen Panorámica	Error Relativo
512×512	128×512	4,48 e-6
512×512	256×1024	4,75 e-6
512×512	512×2048	5,08 e-6
512×512	1024×4096	5,36 e-6
1024×1024	128×512	3,94 e-6
1024×1024	256×1024	4,81 e-6
1024×1024	512×2048	5,32 e-6
1024×1024	1024×4096	5,50 e-6
2048×2048	128×512	3,97 e-6
2048×2048	256×1024	4,97 e-6
2048×2048	512×2048	5,30 e-6
2048×2048	1024×4096	5,48 e-6
4096×4096	128×512	3,92 e-6
4096×4096	256×1024	4,88 e-6
4096×4096	512×2048	5,05 e-6
4096×4096	1024×4096	5,32 e-6

4.3.2. Pruebas para determinar el óptimo tamaño de bloque

Para determinar el óptimo tamaño de bloques que permitan obtener los mejores rendimientos computacionales para la implementación paralela propuesta con interpolación vecino más cercano e interpolación bilineal, se realizaron una serie de pruebas con diferentes tamaños de bloques (16×16, 32×16, 64×8, 128×4, 256×2 y 512×1). Para calcular el rendimiento computacional se ejecutó 100 veces el código para cada casos de imágenes y se hallo la media de las 100 interacciones.

En las las Tablas 4.2 y 4.3 se muestran los resultados computacionales para los diferentes tamaños de bloques. Como se observa en la Tabla 4.2, para el caso interpolación de vecino más cercarno el tamaño del bloque para el cual se obtiene mejores resultados computacionales (menores tiempos de procesamiento) es el tamaño 64×8. En la Tabla 4.3 se muestran los resultados computacionales para la interpolación bilineal. El tamaño de bloque en el cual se obtiene un mejor rendimiento computacional (menores tiempos de procesamiento) es el tamaño 256×2.

Tabla 4.2: Tablas de rendimiento computacional para interpolación vecino más cercano con diferentes tamaños de bloques.

Resolución		**Tiempos de procesamientos (ms)**					
Imagen Omni	**Imagen Pano**	**Bloque de 16x16**	**Bloque de 32x16**	**Bloque de 64x8**	**Bloque de 128x4**	**Bloque de 256x2**	**Bloque de 512x1**
512×512	128×512	0,53	0,53	0,59	0,54	0,53	0,57
512×512	256×1024	0,78	0,85	0,84	0,85	0,84	0,85
512×512	512×2048	1,61	1,58	1,53	1,49	1,49	1,59
512×512	1024×4096	5,04	4,44	4,15	4,02	3,74	3,75
1024×1024	128×512	0,76	0,76	0,78	0,80	0,79	0,84
1024×1024	256×1024	1,14	1,15	1,16	1,20	1,43	1,38
1024×1024	512×2048	2,24	2,29	2,17	2,12	2,22	4,27
1024×1024	1024×4096	5,84	5,66	5,25	5,00	4,83	5,42
2048×2048	128×512	1,11	1,12	1,13	1,15	1,15	1,22
2048×2048	256×1024	2,05	2,01	2,17	2,14	2,35	2,31
2048×2048	512×2048	3,84	3,63	3,88	3,66	5,04	7,20
2048×2048	1024×4096	8,06	8,57	8,01	7,43	7,60	16,30
4096×4096	128×512	2,04	2,08	2,09	2,31	2,28	2,37
4096×4096	256×1024	3,59	3,74	4,03	3,90	5,37	4,94
4096×4096	512×2048	7,79	7,64	7,79	8,71	9,94	18,97
4096×4096	1024×4096	15,61	14,26	13,75	15,45	22,32	34,22
Tiempo total		62,03	60,32	59,30	60,79	71,92	106,21

Tabla 4.3: Tablas de rendimiento computacional para interpolación bilineal con diferentes tamaños de bloques.

Resolución		**Tiempos de procesamientos (ms)**					
Imagen Omni	**Imagen Pano**	**Bloque de 16x16**	**Bloque de 32x16**	**Bloque de 64x8**	**Bloque de 128x4**	**Bloque de 256x2**	**Bloque de 512x1**
512×512	128×512	0,81	0,80	0,82	0,78	0,75	0,77
512×512	256×1024	2,16	1,81	1,73	1,64	1,61	1,65
512×512	512×2048	7,73	7,30	6,83	6,15	5,44	4,90
512×512	1024×4096	38,10	31,30	29,76	25,37	22,66	18,50
1024×1024	128×512	1,16	1,09	1,09	1,07	1,07	1,17
1024×1024	256×1024	2,70	2,36	2,22	2,06	2,01	2,12
1024×1024	512×2048	8,24	7,51	7,02	6,27	5,88	5,58
1024×1024	1024×4096	36,72	31,61	29,73	25,52	22,88	20,41
2048×2048	128×512	1,81	1,64	1,62	1,62	1,64	1,72
2048×2048	256×1024	4,32	3,64	3,39	3,25	3,27	3,64
2048×2048	512×2048	10,67	9,56	8,42	8,00	7,28	7,73
2048×2048	1024×4096	38,97	34,54	30,25	26,27	23,72	23,05
4096×4096	128×512	3,13	2,90	2,76	2,80	2,77	3,15
4096×4096	256×1024	7,02	5,98	5,80	5,82	6,14	6,44
4096×4096	512×2048	16,75	14,66	13,02	12,48	12,95	16,92
4096×4096	1024×4096	45,23	41,89	36,49	32,25	29,53	37,45
Tiempo total		225,52	198,60	180,94	161,33	149,60	155,20

4.3.3. Análisis de resultados para el bloque seleccionado

En la Tabla 4.4 se muestran los resultados de los tiempos de procesamiento del *unwarping* de imágenes omnidireccionales para los bloques seleccionados anteriormente (bloque óptimo para interpolación vecino más cercano y bilineal). Asimismo, se muestran los tiempos de procesamiento del algoritmo PMPA y el aumento de velocidad entre el algoritmo propuesto y PMPA. Para un mejor análisis y visualización de los resultados estos fueron representados de forma gráfica en las Figuras 4.3, 4.4, 4.5 y 4.6.

Tabla 4.4: Tabla de rendimiento computacional de los bloques seleccionados para interpolación vecino más cercano e interpolación bilineal.

		Interp. vecino más cercano			**Interp. bilineal**		
Imagen Omni	Imagen Pano	PMPA (ms)	Propuesto (ms)	Speed Up	PMPA (ms)	Propuesto (ms)	Speed Up
512×512	128×512	0,53	0,59	0,91	1,41	0,75	1,88
512×512	256×1024	1,58	0,84	1,88	3,66	1,61	2,27
512×512	512×2048	3,88	1,53	2,54	11,43	5,44	2,10
512×512	1024×4096	11,26	4,15	2,71	40,80	22,66	1,80
1024×1024	128×512	1,20	0,78	1,54	2,64	1,07	2,47
1024×1024	256×1024	2,89	1,16	2,50	6,64	2,01	3,31
1024×1024	512×2048	7,54	2,17	3,47	17,19	5,88	2,92
1024×1024	1024×4096	18,63	5,25	3,55	52,15	22,88	2,28
2048×2048	128×512	1,99	1,13	1,76	4,98	1,64	3,04
2048×2048	256×1024	6,63	2,17	3,06	15,71	3,27	4,80
2048×2048	512×2048	20,49	3,88	5,29	41,37	7,28	5,68
2048×2048	1024×4096	59,24	8,01	7,40	100,97	23,72	4,26
4096×4096	128×512	2,82	2,09	1,35	7,54	2,77	2,72
4096×4096	256×1024	11,49	4,03	2,85	28,13	6,14	4,58
4096×4096	512×2048	40,17	7,79	5,16	87,24	12,95	6,74
4096×4096	1024×4096	111,61	13,75	8,12	220,47	29,53	7,47

En las Figuras 4.3 y 4.4 se muestran tiempo de procesamiento del algoritmo propuesto y del algoritmo PMPA para *unwarping* de imágenes omnidireccionales de 512×512 $\sim$ 4096×4096 en imágenes panorámicas de 128×512 $\sim$ 1024×4096. En las Figuras se puede observar que el algoritmo propuesto requiere menor tiempo de procesamiento para realizar el *unwarping* de imágenes omnidireccionales en comparación con el algoritmo PMPA. Sin embargo, en el caso de interpolación vecino más cercano para el *unwarping* de imágenes omnidireccionales en imágenes panorámicas de tamaño 128×512 el algoritmo PMPA tiene un menor tiempo de procesamiento. Esto era de esperarse ya que cuando se implementa un algoritmo paralelo CUDA el procesamiento de la data tiene que compensar las transferencias de datos, caso que en imágenes panorámicas de tamaño 128×512 el procesamiento no es lo suficientemente grande para compensar las transferencias. Por otro lado, se muestran en las figuras los cuadros por segundo que toma el algoritmo propuesto en realizar el *unwarping*. El rango de cuadros por segundo para los dos tipos de interpolación van de 1697 $\sim$ 34 FPS.

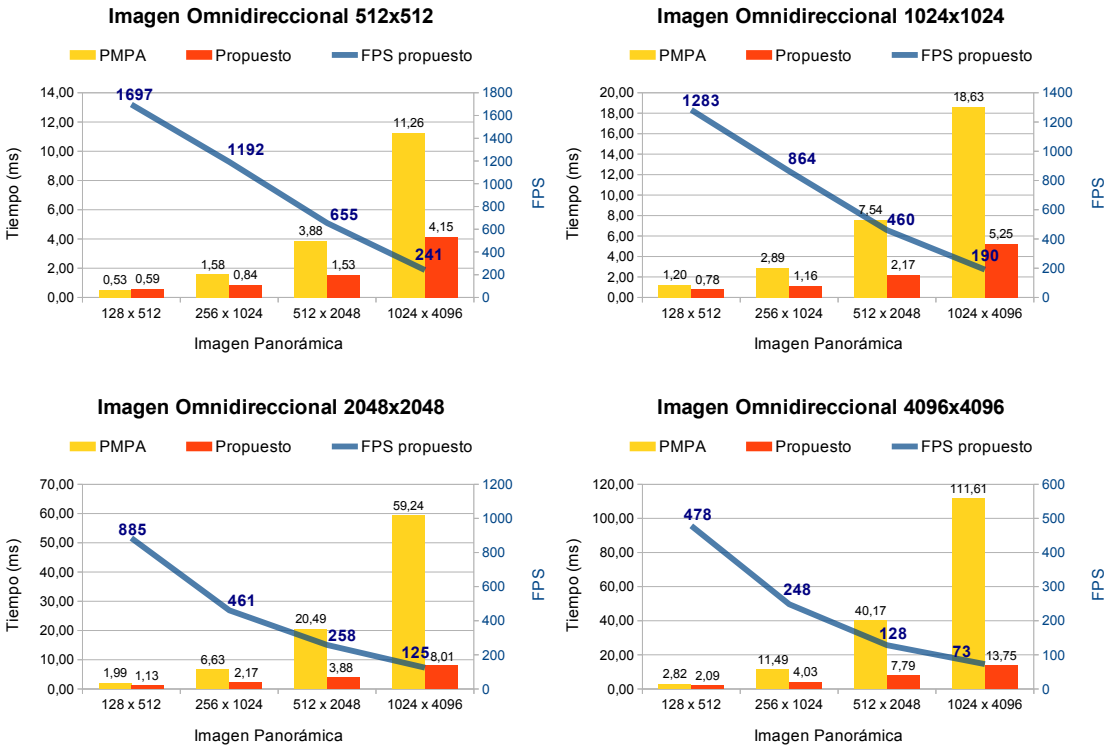

Figura 4.3: Tiempos de procesamiento del algoritmo propuesto y PMPA para interpolación vecino más cercano en la plataforma Jetson TK1. Propuesto (implementación CUDA) y PMPA (implementación ANSI-C).

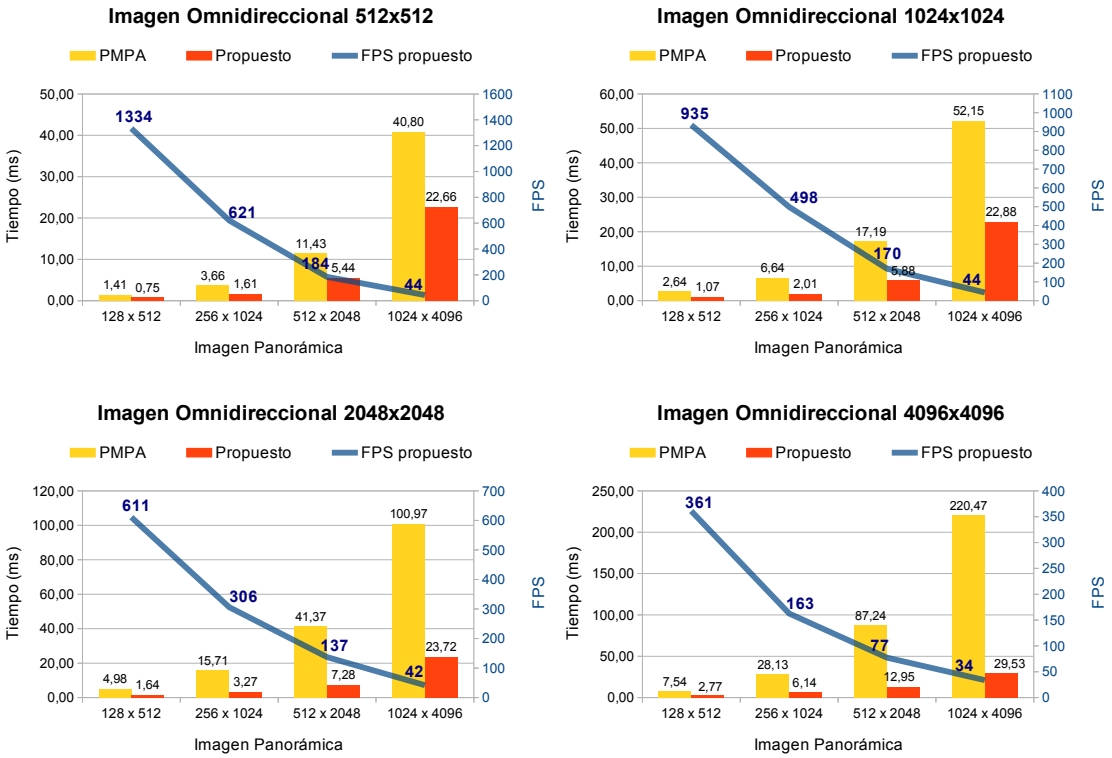

Figura 4.4: Tiempos de procesamiento del algoritmo propuesto y PMPA para interpolación bilineal en la plataforma Jetson TK1. Propuesto (implementación CUDA) y PMPA (implementación ANSI-C).

En las Figuras 4.5 y 4.6 se muestran los aumentos de velocidad entre el algoritmo propuesto y el algoritmo PMPA para *unwarping* de imágenes omnidireccionales de 512×512 $\sim$ 4096×4096 en imágenes panorámicas de 128×512 $\sim$ 1024×4096. En la interpolación de vecino más cercano (Figura 4.5) se observa que el rango de aumento de velocidad va de 1.35 a 8.12, sin contar los casos en los cuales no se obtuvieron ganancia (imágenes panorámicas de 128×512). En la interpolación bilineal se observa un rango de aumento de velocidad de 1.88 a 7.47. En este tipo de interpolación siempre se obtuvo ganancia ya el procesamiento de la data compensó los tiempos de transferencias de datos.

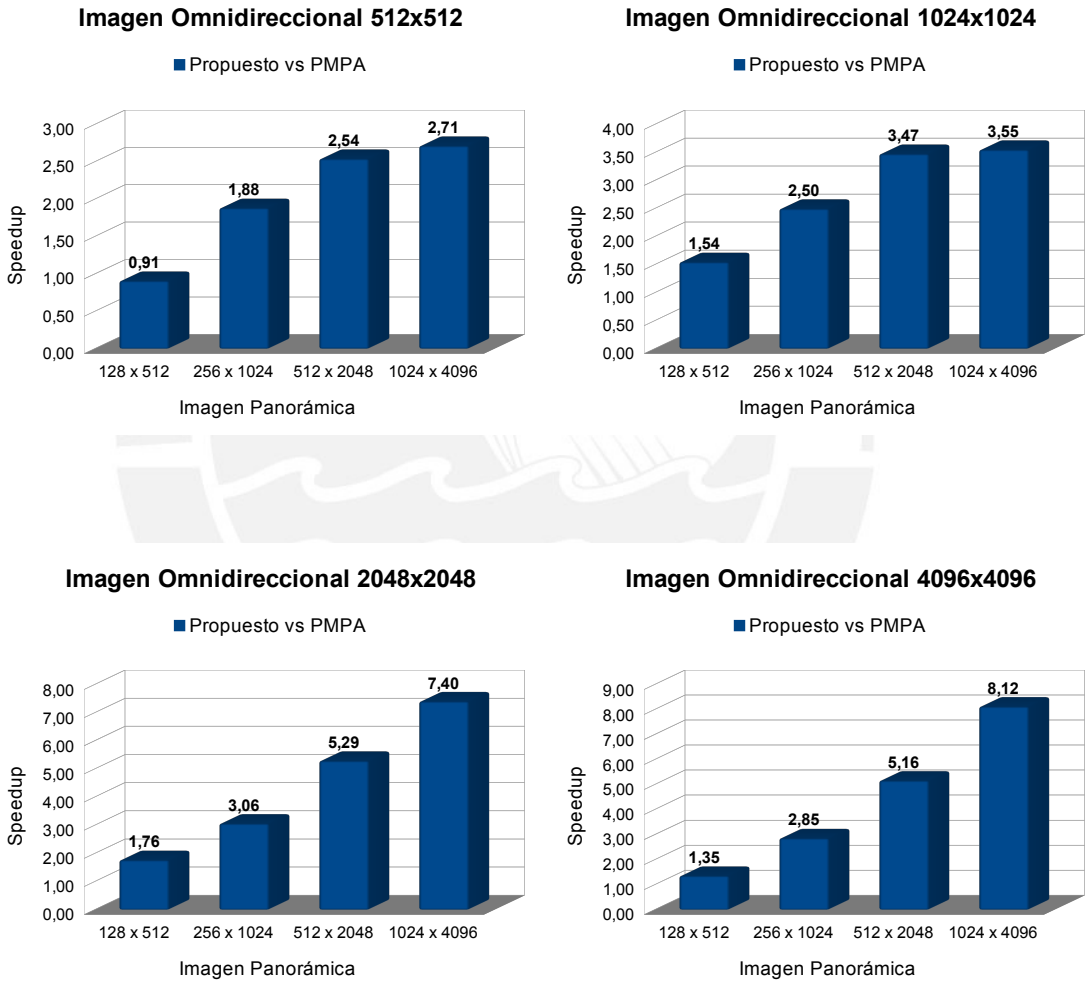

Figura 4.5: Resultados de aumento de velocidad entre el algoritmo propuesto y PMPA para interpolación vecino más cercano en la plataforma Jetson TK1. Propuesto (implementación CUDA) y PMPA (implementación ANSI-C).

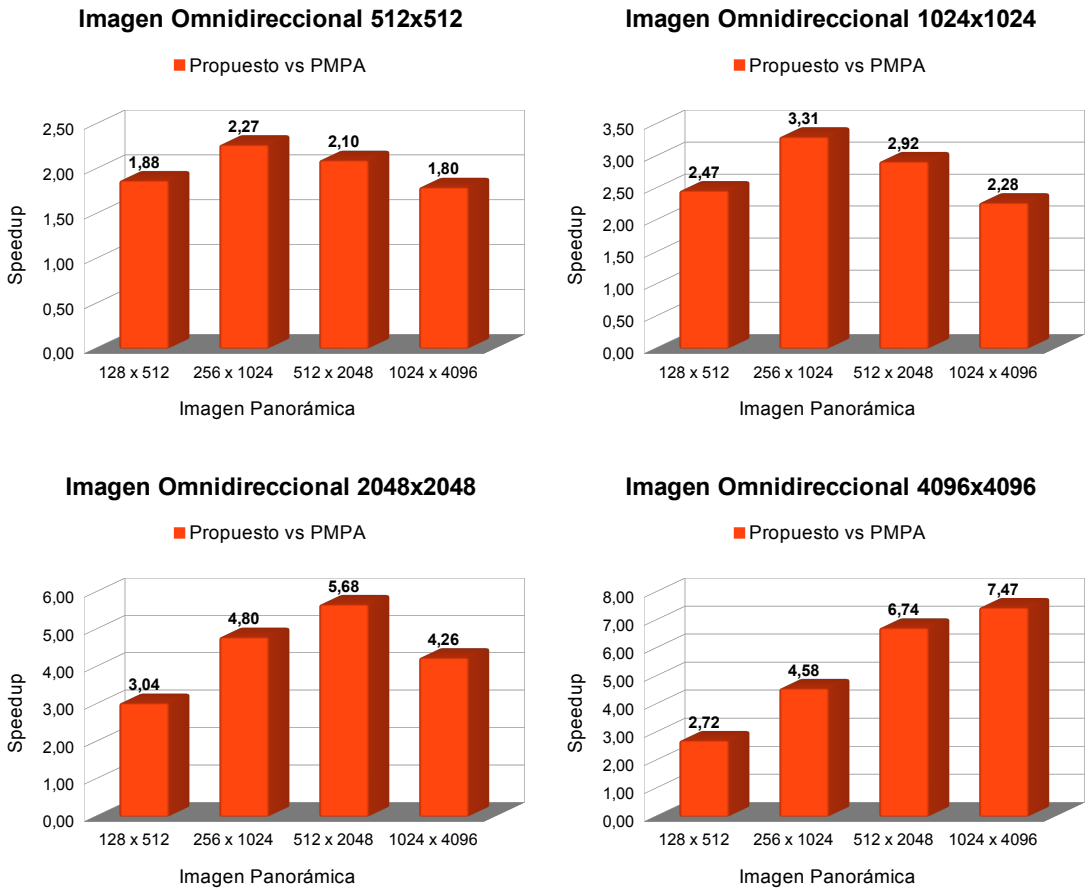

Figura 4.6: Resultados de aumento de velocidad entre el algoritmo propuesto y PMPA para interpolación bilineal en la plataforma Jetson TK1. Propuesto (implementación CUDA) y PMPA (implementación ANSI-C).

Conclusiones

- En la plataforma Jetson TK1, la implementación paralela CUDA realizada mejora el rendimiento computacional en comparación con la implementación PMPA (implementación ANSI-C) en un rango de 1.35 $\sim$ 8.12 para el caso *unwarping* de imágenes omnidireccionales de 512×512 $\sim$ 4096×4096 en 128×516 $\sim$ 1024×4096. Asimismo, se puede concluir, que al procesar imágenes más grandes y obtener resultados con mayor resolución el aumento de velocidad del algoritmo propuesto va incrementando en comparación con el algoritmo PMPA.
- Se puede observar en los resultados que para imágenes panorámicas de tamaño 128×516 (imágenes resultantes pequeñas) no se obtiene un aumento de velocidad considerable (mayor a 2) en comparación con la implementación PMPA. Esto responde al hecho que al ser pequeña las imágenes resultantes, el procesamiento para generarlas no compensa los tiempos de transferencia entre el CPU y el GPU.
- El tiempo de procesamiento obtenido al realizar el *unwarping* de imágenes omnidireccionales con el diseño propuesto y la implementación realizada es lo suficientemente bajo para ser utilizado en aplicaciones de procesamiento de imágenes en tiempo real, puesto a que en los casos de *unwarp* presentados se tiene un rango de 1697 $\sim$ 34 cuadros por segundo (mayor a 30 cuadros por segundo).

Recomendaciones y observaciones

- Como se mencionó la plataforma utilizada es la Jetson TK1, la cual tiene una capacidad de computo de 3.2 similar en recursos a la capacidad 3.0 más el recurso de memoria unificada física. Se recomienda utilizar GPUs o plataformas moviles Nvidia con arquitectura Maxwell (capacidades de computo de 5.X) o arquitectura Kepler (capacidad de computo de 3.5) ya que estas capacidades permiten usar operaciones intrínsecas u otros recursos que disminuyan los tiempos de procesamiento. Con relación a este tipo de operaciones se recomienda utilizar memoria de solo lectura (__ldg()) para acceder a la data con menores tiempos de latencia . Asimismo, evaluar las operaciones intrínsecas de multiplicación (__fmul_rd, __fmaf_rn, etc.) que mejorarían el rendimiento computacional pero se podria perder precisión.

- Para el caso que se necesiten imágenes de baja resolución menores a 128×512 o imágenes de igual tamaño en las cuales la implementación propuesta no tiene ganancia con respecto a la implementación PMPA (implementación ANSI-C) se recomienda utilizar PMPA. Sin embargo, para mejorar los resultados computacionales del *unwarping* con PMPA de imágenes panorámicas de baja resolución se recomienda implementar un algoritmo en PTHREADS.

- Se recomienda realizar investigaciones en *unwarping* por geometría simétrica (un octavo de tabla panorámica) para aprovechar la idea principal de reducir los accesos a memoria de la tabla de mapeo. Esta idea sería un punto muy importante si se implementa en CUDA ya que memores accesos a memoria es igual a menores tiempos de procesamiento.

Bibliografía

[1] Y. Tang, Y. Li, T. Bai, X. Zhou, and Z. Li, "Human tracking in thermal catadioptric omnidirectional vision," in *IEEE International Conference on Information and Automation*, pp. 97–102, 2011.

[2] C.-S. Fahn and C.-S. Lo, "A high-definition human face tracking system using the fusion of omni-directional and ptz cameras mounted on a mobile robot," in *IEEE Conference on Industrial Electronics and Applications*, pp. 6–11, 2010.

[3] S. Jeng and W. Tsai, "Construction of perspective and panoramic images from omni-images taken from hypercatadioptric cameras for visual surveillance," in *IEEE International Conference on Networking, Sensing and Control*, vol. 1, pp. 204–209, 2004.

[4] P.-H. Yuan, K.-F. Yang, and W.-H. Tsai, "Real-time security monitoring around a video surveillance vehicle with a pair of two-camera omni-imaging devices," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 8, pp. 3603–3614, 2011.

[5] J. Gaspar, N. Winters, and J. Santos-Victor, "Vision-based navigation and environmental representations with an omnidirectional camera," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 890–898, 2000.

[6] H. Liu, N. Dong, and H. Zha, "Omni-directional vision based human motion detection for autonomous mobile robots," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2236–2241, 2005.

[7] T. Gandhi and M. Trivedi, "Vehicle surround capture: survey of techniques and a novel omni-video-based approach for dynamic panoramic surround maps," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 293–308, 2006.

[8] N. Winters, J. Gaspar, G. Lacey, and J. Santos-Victor, "Omni-directional vision for robot navigation," in *IEEE Workshop on Omnidirectional Vision*, pp. 21–28, 2000.

[9] D. Michel, A. A. Argyros, and M. I. Lourakis, "Horizon matching for localizing unordered panoramic images," *Computer Vision and Image Understanding*, vol. 114, no. 2, pp. 274–285, 2010.

[10] M. Fiala and A. Basu, "Panoramic stereo reconstruction using non-svp optics," in *16th International Conference on Pattern Recognition*, vol. 4, pp. 27–30, 2002.

[11] M. Schönbein, H. Rapp, and M. Lauer, "Panoramic 3d reconstruction with three catadioptric cameras," in *Intelligent Autonomous Systems 12*, pp. 345–353, 2013.

[12] S. Ramalingam, S. Bouaziz, P. Sturm, and M. Brand, "Geolocalization using skylines from omni-images," in *IEEE International Conference on Computer Vision Workshops*, pp. 23–30, 2009.

[13] S. Peleg, Y. Pritch, and M. Ben-Ezra, "Cameras for stereo panoramic imaging," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 208–214, 2000.

[14] L. Chen, M. Zhang, B. Wang, Z. Xiong, and G. Cheng, "Real-time fpga-based panoramic unrolling of high-resolution catadioptric omnidirectional images," in *International Conference on Measuring Technology and Mechatronics Automation*, vol. 1, pp. 502–505, 2009.

[15] N. S. Chong, M. Wong, and Y. H. Kho, "A parallel root-finding method for omnidirectional image unwrapping," in *Visual Communications and Image Processing (VCIP)*, pp. 1–6, 2013.

[16] N. Chong, Y. Kho, and M. Wong, "A closed form unwrapping method for a spherical omnidirectional view sensor," *EURASIP Journal on Image and Video Processing*, vol. 2013, pp. 1–5, 2013.

[17] Q. Wang, K. Zhang, Y. Jiang, and X. Xiong, "The discrete algorithm of log-polar transformation," in *1st International Symposium on Systems and Control in Aerospace and Astronautics*, pp. 4–7, 2006.

[18] G. Wolberg and S. Zokai, "Robust image registration using log-polar transform," in *International Conference on Image Processing*, vol. 1, pp. 493–496, 2000.

[19] A. Torii and A. Imiya, "Panoramic image transform of omnidirectional images using discrete geometry techniques," in *2nd International Symposium on 3D Data Processing, Visualization and Transmission*, pp. 608–615, 2004.

[20] S. Jeng and W. Tsai, "Using pano-mapping tables for unwarping of omni-images into panoramic and perspective-view images," *IET Image Processing*, vol. 1, pp. 149–155, 2007.

[21] W. Wong, W. ShenPua, C. Loo, and W. Lim, "A study of different unwarping methods for omnidirectional imaging," in *IEEE International Conference on Signal and Image Processing Applications*, pp. 433–438, 2011.

[22] Z. Xiong, I. Cheng, A. Basu, W. Wang, W. Xu, and M. Zhang, "Efficient omni-image unwarping using geometric symmetry," *Machine Vision and Applications*, vol. 23, pp. 725–737, 2012.

[23] J. Reategui, P. Rodriguez, and N. Ragot, "Fast omni-image unwarping using pano-mapping pointers array," in *IEEE International Conference on Image Processing (ICIP)*, pp. 5811–5815, 2014.

[24] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, August 2014.

[25] NVIDIA Corporation, *CUDA C best practice Guide*, August 2014.

[26] NVIDIA Corporation, *Tuning CUDA applications for Kepler*, August 2014.

[27] NVIDIA Corporation, *Technical Brief NVIDIA Jetson TK1 Development Kit Bringing GPU-accelerated computing to Embedded Systems*, April 2014.

[28] NVIDIA Corporation, *Whitepaper NVIDIA Tegra K1 - A New Era in Mobile Computing*, January 2014.

[29] J. Jean and S. Graillat, "A parallel algorithm for dot product over word-size finite field using floating-point arithmetic," *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 80–87, 2010.

[30] R. Landaverde, T. Zhang, A. K. Coskun, and M. Herbordt, "An investigation of unified memory access performance in CUDA," 2014.

[31] J. Lv, G. Li, A. Humphrey, and G. Gopalakrishnan, "Performance degradation analysis of gpu kernels," in *International Workshop on Exploiting Concurrency Efficiently and Correctly*, 2011.

[32] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, 2000.

[33] C. Mei and P. Rives, "Calibration between a central catadioptric camera and a laser range finder for robotic applications," in *IEEE International Conference on Robotics and Automation*, pp. 532–537, 2006.

[34] C. Mei and P. Rives, "Single view point omnidirectional camera calibration from planar grids," in *IEEE International Conference on Robotics and Automation*, pp. 3945–3950, IEEE, 2007.

[35] J. Lei, X. Du, Y.-f. Zhu, and J.-l. Liu, "Unwrapping and stereo rectification for omnidirectional images," *Journal of Zhejiang University SCIENCE A*, vol. 10, no. 8, pp. 1125–1139, 2009.

[36] D. Phillips, *Image Processing in C: Analyzing and Enhancing Digital Images*. R and D Publications, 1994.

[37] G. Silva, J. Reátegui, and P. Rodríguez, "Fast omni-image unwarping on the Jetson TK1," in *GPU Technology Conference (GTC)*, (San Jose, CA, USA), 2015.

[38] P. Rodriguez, *Yupana*. Grupo de Procesamiento Digital de Señales e Imágenes (GPDSI-PUCP), 2014.