MARCAS DE AGUA DE IMÁGENES EN PARALELO

Por

Alcibíades Bustillo Zárate

Tesis presentada en cumplimiento parcial de los requisitos para el grado de:

MAESTRÍA EN CIENCIAS

en

COMPUTACIÓN CIENTÍFICA

UNIVERSIDAD DE PUERTO RICO
RECINTO UNIVERSITARIO DE MAYAGÜEZ

Junio, 2015

Aprobada por:

Marko Schutz, Ph.D Miembro, Comité Graduado	Fecha
Xuerong Yong, Ph.D Miembro, Comité Graduado	Fecha
Dorothy Bollman, Ph.D Presidente, Comité Graduado	Fecha
Manuel Rodriguez-Martinez, Ph.D Representante de Estudios Graduados	Fecha
Olgamary Rivera, Ph.D Director Interino del Departamento	Fecha

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

MARCAS DE AGUA DE IMÁGENES EN PARALELO

Por

Alcibíades Bustillo Zárate

Junio 2015

Consejero: Dorothy Bollman
Departamento: Ciencias Matemáticas

Mientras que el Internet ha hecho posible para el consumidor obtener de manera fácil archivos de tipo digital como imágenes, audio, vídeo, etc. ha hecho también posible obtener de manera ilegal material con derechos de autor. Las marcas de agua digitales son una solución parcial a este problema. Insertar una marca de agua en una versión legal puede ayudar al autor identificar quien tiene una copia ilegal. Debido al enorme incremento del flujo de la información, es necesario insertar las marcas de agua en los archivos en el menor tiempo posible. Por esta razón es natural pensar en computación en paralelo. Diferentes técnicas para insertar marcas de agua en imágenes digitales han aparecido en la literatura durante los últimos veinte años, sin embargo, sólo unos pocos han considerado la posibilidad de aplicar computación en paralelo y los que lo hacen, sólo tienen en cuenta el uso de GPUs. No se tiene en cuenta el uso de otros modelos de computació en paralelo como OpenMP o MPI. En este trabajo damos un algoritmo embarazosamente paralelo para una familia de uso general de algoritmos de marcas de agua en el dominio de la frecuencia y comparamos el rendimiento de las implememntaciones secuencial, OpenMP, MPI y CUDA de un sencillo representante de esta familia, con especial énfasis en OpenMP

y MPI. Nuestros experimentos muestran que con una solo GPU CUDA es casi 300 veces más rápido que la versión secuencial y muchas veces más rápido que OpenMP y MPI utilizando 1-8 nodos.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

MARCAS DE AGUA DE IMÁGENES EN PARALELO

Por

Alcibíades Bustillo Zárate

Junio 2015

Consejero: Dorothy Bollman
Departamento: Ciencias Matemáticas

While the Internet has made it possible for the consumer to easily obtain images, audio, video, etc. in digital form, it has also made it easier to illegally obtain copyrighted material. Digital watermarking is a partial solution to this problem. Embedding a watermark in a legal version of material can help the copyright owner to identify who has an illegal copy. Because of the ever increasing enormity of the flow of information, it becomes necessary to watermark files in the least amount of time possible. For this reason it is natural to turn to parallel computing. Many different techniques for embedding watermarks in digital images have appeared in the literature for at least the last twenty years, However, only a few have considered the possibility of applying parallel computing and those that do, consider only the use of GPUs. Not one considers the use of other models of parallel computation such as OpenMP or MPI. In this work we give an embarrassingly parallel algorithm for a commonly used family of watermarking algorithms in the frequency domain and we compare performance of sequential, OpenMP, MPI and CUDA implementations of a simple representative of this family, with particular emphasis on OpenMP and MPI. Our experiments show that CUDA with one GPU is almost 300 times faster

than the sequential version and many times faster than OpenMP and MPI using 1 to 8 nodes.

Dedico este trabajo a mis padres, hermanos y a ti Liz por todo el apoyo y compresión que me han brindado.

AGRADECIMIENTOS

Este trabajo uso Ciencia Extrema e Ingeniería del Medio Ambiente (XSEDE por sus siglas en inglés) con financiamiento de la Fundación Nacional para la Ciencia con número de concesión ACI-1053575

Quiero dar un agradecimiento especial a la Dra. Dorothy Bollman, mi asesor por sus incontables horas de reflexión, lectura, y aliento, y sobre todo la paciencia durante todo este proceso.

Quiero reconocer y agradecer a todos mi compañeros y amigos que me han acompañado durante este tiempo, pero quiero hacer mención especial a Einstein Morales por su continuo apoyo

Índice general

página

Índice de cuadros

Tabla página

Índice de figuras

Figura página

LISTA DE ABREVIATURAS

DCT	Discrete Cosine Transform.
IDCT	Inverse Discrete Cosine Transform.
DWT	Discrete Wavelet Transform.
EZW	Embedding Zerotree Wavelet.
DFT	Discrete Fourier Transform.
OpenMP	Open Multi Processing.
MPI	Message Passing Interface.
CUDA	Compute Unified Device Architecture.
CPU	Central Processing Unit.
GPU	Graphic Processing Unit.
PSNR	Peak Signal to Noise Ratio.
SMPs	Shared-Memory Parallel Computer.

Capítulo 1
INTRODUCCIÓN

Las marca de aguas son el proceso de insertar información dentro de archivos tipo multimedia, incluyendo texto, imágenes, vídeo o audio, con el propósito de mostrar autenticidad. Una marca de agua puede ser perceptualmente visible o invisible al ojo humano. Las marcas de agua visibles suelen ser logotipos de empresas o firmas digitales personales, que se utilizan para declarar la propiedad de las imágenes digitales y disminuir el uso no autorizado de los originales [18]. Por otro lado, marcas de agua invisibles son usualmente usadas para detectar el uso fraudulento del material. Por ejemplo, un vendedor puede querer identificar a una persona que ha utilizado su material sin pagar derechos de autor o el gobierno podría querer detectar la identidad de una persona que divulgó material clasificado.

Para que una marca de agua sea útil, debe ser detectable o extraíble por el propietario. También debe ser "robusta" o resistente a los ataques, ya sea intencionales o no intencionales. Es decir, que la marca de agua debe permanecer intacta después de ser atacada.

Entre las aplicaciones de las marcas de agua se encuentran: monitoreo de emisión, identificación del propietario, prueba de propiedad, autenticación, marcas de agua transaccionales, control de copia y comunicación encubierta [7].

Para monitoreo de emisiones, se puede colocar una única marca de agua en cada clip de vídeo o sonido previamente de ser emitido. Estaciones de monitoreo automatizadas pueden recibir la señal y buscar esas marcas de agua, e identificar cuando y donde cada clip aparece.

Los avisos de derecho de autor (copyright) no garantizan los derechos de copia, pero todavía se recomiendan. Estos avisos por lo general son de la forma "©fecha, propietario". En libros y fotografías, este aviso esta a la vista. En las películas es colocado al final de los créditos. Y en música pre-grabada este es colocado en el empaquetado. Una desventaja de este tipo de avisos es que pueden ser retirados del material protegido. El paquete puede ser perdido, los créditos pueden ser cortados y las imágenes pueden ser recortadas. Colocando una marca de agua digital puede ser un complemento a estos avisos ya que esta se convierte en parte integral del contenido.

Propietarios de archivos tipo multimedia pueden querer usar marcas de agua no sólo para identificar la propiedad de los derechos de autor, sino para probar realmente la propiedad. Es decir, insertando una marca de agua digital puede ayudar a demostrar quien es el dueño.

Existen muchas aplicaciones donde la veracidad de una imagen es crucial, especialmente casos legales y en imágenes médicas. Se han creado firmas criptográficas que son asociadas con una imagen. Pero si incluso un bit de un píxel de la imagen es modificado, este no coincidirá con la firma. Una posible solución es insertar la firma directamente en la imagen con marcas de agua. Esto elimina el problema de garantizar que la firma coincida con la imagen.

Aplicaciones de monitoreo e identificación de propietario colocan una misma marca de agua en todas las copias del mismo contenido. Sin embargo, la distribución electrónica de contenidos permite que cada copia distribuida sea personalizada para cada destinatario. Esta capacidad permite a una marca de agua única ser insertada en cada copia individual. Marcas de agua transaccionales o también llamadas huellas digitales permiten a un propietario o distribuidor del contenido identificar la fuente de una copia ilegal.

Marcas de agua transaccionales, así como marcas de agua para la vigilancia, identificación y prueba de propiedad no impiden la copia ilegal. Por el contrario, sirven como poderosos elementos disuasorios y herramientas de investigación. Sin embargo, también es posible para los dispositivos de grabación y reproducción reaccionar a señales incrustadas. De esta manera, un dispositivo de grabación podría inhibir la grabación de una señal si se detecta una marca de agua que indica que la grabación está prohibida.

Aunque diferentes técnicas para insertar marcas de agua en imágenes digitales han aparecido en la literatura durante los últimos veinte años con el animo de proteger la autenticidad e integridad de la información. Usualmente estos procesos son ejecutados en equipos de muy poca memoria que pueden procesar pequeñas cantidades de imágenes. Cuando el volumen de imágenes se hace muy grande, se necesita más espacio en disco para almacenar datos a gran escala pero estos no pueden ser procesados de manera eficaz. Si se dividen las imágenes en una gran cantidad de pedazos y procesarlos paso a paso por separado, estos pueden ser procesados, pero se consume mucho tiempo. Por lo tanto, muy pocos han considerado la posibilidad de aplicar la computación en paralelo para agilizar el proceso de inserción debido a que estamos atravesando una época donde el flujo de información esta creciendo de manera muy rápida. En este trabajo se considera el uso de la computación en paralelo con el fin de facilitar la inserción de marcas de agua en el dominio de la frecuencia de imágenes digitales y comparamos el rendimiento de implementaciones secuenciales, OpenMP, MPI, y CUDA de un representante de una familia de algoritmos en el dominio de frecuencia.

Capítulo 2
TRABAJOS PREVIOS

Algoritmos de marcas de agua para imágenes digitales pueden ser clasificados de acuerdo al dominio en el cual se inserta una marca de agua. Existen dos técnicas muy populares, marcas de agua en el dominio espacial y marcas de agua en el dominio de frecuencia. Además se pueden clasificar en marcas de agua visibles o invisibles. Por ejemplo, Mohanty et al. [19] desarrolló un algoritmo de marca de agua visible en el dominio de frecuencia , el cual consistía en reemplazar cada coeficiente $c_{ij}(n)$ en el $n-$ésimo bloque de la imagen por $\alpha_n c_{ij}(n) + \beta_n w_{ij}(n)$ donde $w_{ij}(n)$ es el coeficiente de la transformada discreta del coseno (DCT por sus siglas en inglés) de la marca de agua y α_n y β_n son valores que dependen de la varianza de los valores DCT del bloque n. Luego la imagen con la marca de agua resulta de aplicar la inversa de la DCT a cada uno de los bloques resultante.

En cambio, Chen et al. [25] propone un nuevo esquema en el cual la marca de agua es insertada en un selecto conjunto de coeficientes de la transformada wavelet, y para asegurar que la marca de agua sea invisible y robusta, una secuencia binaria seudo-aleatoria es insertada en coeficientes significativos de la transformada wavelet usando el algoritmo de codificación Embedding Zerotree Wavelet (EZW).

Megalingam et al. [17], proponen un método de marca de agua en el dominio espacial, donde la marca de agua es insertada en diferentes subseciones de la imagen con diferentes intensidades. Como algoritmo de inserción usan la ecuación $y' = y + \alpha I$, donde $y(i, j)$, es la intensidad de la imagen original en la posicion (i, j), y' es

la imagen con la marca de agua, y αI es la imagen insertada con pequeños cambios en los niveles de intensidad.

Por su parte Cox et al. [8], desarrolló un algoritmo de marca de agua invisible el cual escogía una secuencia de números $x_1, x_2, \ldots, x_n$ de acuerdo a la distribución normal con media 0 y varianza 1. La DCT de la imagen, tomada como un solo bloque, es calculada y el componente perceptual más significativo, determinado por los coeficientes DCT más largos, son reemplazados por $v_i(1 + \alpha x_i)$ donde v_i es un componente de frecuencia de la imagen y α es un factor escalar. Finalmente la imagen con la marca de agua consiste en aplicar la inversa de la DCT al este resultado. Dugad et al. [9] presentaron un esquema aditivo donde la marca de agua es insertada en el dominio generado por la transformada wavelet. La marca de agua es insertada usando $\hat{w_{ij}} = w_{ij} + \alpha|w_{ij}|x_{ij}$, donde w_{ij} es un coeficiente seleccionado de los componentes HL_1 y LH_1, y además debe ser mayor a un umbral t_1. La constante α es el parámetro de escalado, x_{ij} es el valor de la marca de agua, y $\hat{w_{ij}}$ es el coeficiente wavelet con la marca de agua. De igual forma, Yong et al. [31], presentan un algoritmo de marca de agua que se basa en la transformada discreta del coseno, y el método de inserción es de la forma $d^* = d_i(1 + aw_i)$ donde d_i es el primer coeficiente con mayor valor en el dominio generado por la DCT y w_i son valores de la marca de agua con secuencias aleatorias gaussianas.

Pereira y Pun [21], propusieron también un esquema en el cual la marca de agua es invisible pero esta es insertada utilizando la transformada de Fourier y el algoritmo de inserción se base en mapas polares lo que permite que la imagen con la marca de agua sea robusta bajo algunas operaciones de procesamiento de imágenes tales como rotación, escalado y compresión.

Muchos esquemas de marcas de agua visibles siguen la idea desarrollada por Cox, [30], [4], que consiste en segmentar la imagen original y la marca de agua en bloques 8×8 que no se superponen, aplicar una transformada discreta ya sea la del

coseno o wavelet, y obtener una marca de agua visible, modificando los valores de la transformada tanto de la imagen como la marca de agua por combinación lineal de ellos. Chen y Ng [5] trabajó con marca de agua visible pero la inserción de la marca de agua se hace en el dominio espacial usando también el esquema aditivo pero el factor de inserción donde este valor se obtiene hallando el umbral de la imagen original mediante la técnica de Otsu.

Por otra parte, Huang et al. [14] argumenta que para obtener una marca de agua invisible más robusta la marca de agua se debe insertar en los componentes "dc" de la transformada discreta del coseno, ya que estos tienen mayor capacidad perceptual que los componentes "ac".

Shiuh et al [26] desarrollan un método en el que un algoritmo genético es utilizado para insertar bloques de 8×8 de la marca de agua binaria en bloques de la DCT de la imagen original. La imagen con la marca de marca de agua, consiste en la concatenación de la IDCT de cada bloque resultante. Usando este algoritmo, la marca de agua se puede extraer sin la necesidad de la imagen original.

García-Cano et al [10] implementan el algoritmo Shieh en una GPU y comparan los resultados de rendimiento con los de una versión secuencial. Otros trabajos en las que las GPU han sido utilizados para marcas de agua son, por ejemplo, los de Lin, Zhoa, y Yang [15] Y Vihari y Mishra [28]. Lin et al extraen las características del dominio de frecuencia baja y media de la DCT e insertarn la marca de agua en el dominio frecuencia alta. Vihari y Mishra utilizan codificación Huffman para codificar los datos de derechos de autor que se insertan utilizando el método de "Modified Auxiliary Carry Watermarking"por su nombre en inglés.

Capítulo 3
MARCAS DE AGUA DE IMÁGENES DIGITALES

En el siguiente capítulo estudiaremos a nivel general conceptos de marcas de agua. También, discutiremos características de las marcas de agua, importancia que implica insertar una marca de agua en el dominio de frecuencia y el método $\alpha - \beta$ como algoritmo de inserción de la marca de agua. Y por último describiremos los requisitos que deben cumplir las marcas de agua.

3.1. CONCEPTOS GENERALES

Definición 3.1.1. *(Imagen digital). Una imagen digital $a(m, n)$ es dividida en N filas y M columnas. La intersección de una fila y una columna es denominada píxel. El valor asignado a la coordenada entera $[m, n]$ con $\{m = 0, 1, 2, \ldots, M - 1\}$ y $\{n = 0, 1, 2, \ldots, N - 1\}$ es $a[m, n]$.*

Definición 3.1.2. *(Robusto). Robusto significa que el propietario es capaz de recuperar el mensaje oculto incluso si el contenido con la marca de agua ha sido alterado después de la inserción [6].*

Definición 3.1.3. *(Imperceptible). Imperceptibilidad representa la similitud de la imagen con la marca de agua y la imagen original. La Relación Señal a Ruido de Pico o PSNR es comúnmente utilizada para evaluar la degradación de la imagen o la fidelidad de reconstrucción. Se define para dos imágenes I y K de tamaño $M \times N$ como:*

$$PSNR = 10log_{10}\frac{255^2}{\sqrt{MSE(I, K)}}$$

donde I es la imagen original, K es la imagen reconstruida, 255^2 es el píxel de mayor valor en la imagen I, y MSE es el error cuadrático medio entre I y K.

$$MSE(I,K) = \frac{1}{M}\frac{1}{N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|I(i,j) - K(i,j)\|^2$$

En la reconstrucción de imágenes los valores de PSNR varían entre $[30, 50]$. Un valor de PSNR de 50 o mayor indica que las imágenes son prácticamente idénticas.

Definición 3.1.4. *(Medida de robustez). Definimos la medida de similitud entre la marca de agua de referencia W y la marca de agua extraída W' como*

$$NC = \frac{\sum_i \sum_j W(i,j)W'(i,j)}{\sum_i \sum_j [W(i,j)]^2}$$

Un valor igual a 1 del NC indica que la marca de agua con la extraida son prácticamente las mismas.

3.2. MARCAS DE AGUA EN EL DOMINIO DE FRECUENCIA

Marcas de agua en imágenes pueden ser insertadas en el dominio espacial o en el dominio de frecuencias. Técnicas en el dominio espacial envuelven manipulación directa del valor de los píxeles, es decir, los valores de los píxeles de la imagen original se modifican directamente por los valores de los píxeles de la marca de agua. Por ejemplo, cambiar el color de ciertos píxeles.

Un método de dominio de frecuencia consiste en la incorporación de la marca de agua en el "dominio de la frecuencia". Dicho método consiste típicamente en tres etapas: (1) convertir la imagen I desde el dominio espacial al dominio de frecuencia, i.e. calcular la transformada discreta T, tal como la transformada de Fourier, coseno, o wavelet, de I; (2) aplicar el algoritmo de inserción E a $T(I)$ y $S(W)$, donde S es alguna función definida en la marca de agua W, para obtener una nueva imagen $E = E(T(I), S(W))$; (3) convertir E nuevamente al dominio espacial para obtener

la imagen E' con la marca de agua. Un método de extracción consiste en un procedimiento X el cual toma la imagen E' con la marca de agua y posiblemente la imagen original I y produce la marca de agua original $W = X(E', I)$.

Un procedimiento típico en el dominio de frecuencia usa la DCT y particiona la imagen I y la marca de agua W en bloques de 8×8 píxeles. La DCT de cada bloque I_{ij} de la imagen I es reemplazada por $\alpha_{ij} I_{ij} + \beta_{ij} W_{ij}$ donde α_{ij} y β_{ij} son valores que son escogidos acordes con las propiedades del bloque I_{ij}. La imagen con la marca de agua $I(W)$ resulta luego de aplicar la inversa de la DCT y concatenar cada bloque resultante. En símbolos

$$I(W) = \bigcup_{ij} IDCT(\alpha_{ij} DCT(I_{ij}) + \beta_{ij} DCT(W_{ij}))$$

Podemos denominar este proceso de marca de agua, un método "$\alpha - \beta$".

En un método $\alpha - \beta$ cada par de bloques (I_{ij}, W_{ij}) puede ser procesado independientemente, i.e., tal método se le conoce como "embarazosamente paralelo".

Robustez, fidelidad, costo computacional son algunas de las características que debe tener una marca de agua. En la práctica, es casi imposible diseñar un sistema de marca de agua que se sobresalga en todas ellas. Por lo tanto, es necesario hacer balance entre ellas.

Para que una marca de agua sea robusta, esta debe ser insertada en regiones perceptualmente significativas. Ya que si esta es insertada en componentes perceptualmente insignificativos, estos componentes son afectados por distorsiones geométricas (rotación, traslación, escalado y recorte) o distorsiones de señal. Por ejemplo, una marca de agua colocada en el espectro de alta frecuencia de una imagen puede ser fácilmente eliminada con una pequeña degradación de la imagen con cualquier proceso que use un filtro paso bajo.

El problema entonces es cómo insertar una marca de agua en las regiones perceptualmente más significativas del espectro de una imagen sin que tales alteraciones

sean notables. Para resolver este problema [8], el dominio de frecuencia de la imagen es visto como un canal de comunicación, y, la marca de agua se ve como una señal que se transmite a través de él. Los ataques y distorsiones no intencionales de la señal son tratados como el ruido a los que la señal sumergida (marca de agua) debe ser inmune.

Esparcir la marca de agua en todo el espectro de una imagen asegura seguridad contra ataques intencionales o no intencionales: En primer lugar, la ubicación espacial de la marca de agua no es obvia. Por otra parte, las regiones de frecuencia se deben seleccionar de forma que garantice la degradación severa de los datos originales después de cualquier ataque a la marca de agua.

3.3. REQUISITOS EN MARCAS DE AGUA DE IMÁGENES DIGITALES

A continuación se mencionan algunos de los requisitos que deben cumplir las marcas de agua en imágenes digitales:

Robustez como se ha mencionado anteriormente en la capacidad de detectar la marca de agua luego de que esta sufra modificaciones tales como filtros espaciales, escaneo, compresión con perdida, traslación, escalado y rotación, y otros procesamientos como cortado y conversiones. No todos los algoritmos de marcas de agua tiene el mismo nivel de robustez, algunas son robustas contra manipulaciones, sin embargo fallan contra ataques más fuertes. Por lo tanto, robustez se puede clasificar en: (1) Robusto, cuando la marca de agua es diseñada para sobrevivir luego de ataques accidentales o intencionales. (2) Frágil, la marca de agua está diseñada para ser destruida con cualquier tipo de modificación con el fin de detectar manipulaciones ilegales. (3) Semi-frágil, la marca de agua de este tipo es robusta contra modificaciones accidentales, pero frágil contra ataques maliciosos.

Imperceptibilidad se refiere a la similitud perceptual entre la imagen original antes de de insertar la marca de agua y la imagen con la marca de agua. Esta

característica es conocida como fidelidad e invisibilidad. Aunque en aplicaciones como verificación de propiedad se desea que la marca de agua sea visible.

La capacidad se refiere al número de bits insertados en una imagen. Y esta puede variar de acuerdo a la aplicación.

Complejidad es otro requerimiento de una marca de agua, esta se refiere al costo de inserción de la marca de agua y la detección de la misma.

3.4. ATAQUES A MARCAS DE AGUA

Existen varias tipos de ataques a las marcas de agua, pero estos se han categorizados en las siguientes clases: ataques de eliminación, ataques geométricos y ataques criptográficos [29].

Los ataques de eliminación apuntan a eliminar la información de la marca de agua sin necesidad de romper la seguridad del algoritmo de la marca de agua, por ejemplo, sin usar la clave utilizada para la inserción de la marca de agua. En esta categoría se incluyen ataques como eliminación de ruido, cuantificación. Algunos autores clasifican estos ataques como involuntarios, por ejemplo, compresión busca reducir el tamaño de la imagen sin perder demasiada información visual. Estos métodos no siempre cumplen su objetivo de eliminar la marca de agua, pero sin embargo ocasionan danõ significativo a la marca de agua. Los ataques geométricos también llamados ataques intencionales, a diferencia de los de eliminación, no busca remover la marca de agua, sino, distorsionarla modificando los valore de los píxeles en el dominio espacial. En esta categoría se incluyen ataques como rotaciones, escalado, transformaciones afín. Los ataques criptográficos apuntan a descifrar la seguridad de las marcas de agua para eliminar la marca de agua o incrustar marcas de agua engañosas. Una de estas técnicas es por fuerza bruta.

Capítulo 4
PRELIMINARES

En el siguiente capítulo se discuten tres modelos de programación en paralelo y se presentaran sus características mas importantes, que serán utilizadas para las implementaciones de un algoritmo de marca de agua.

4.1. ¿QUÉ ES OPENMP?

OpenMP ("Open Multi-Processing") es una interfaz de programación de aplicaciones (API por sus siglas en inglés) de memoria compartida cuyas características se basan en facilitar la programación en paralelo de memoria compartida. OpenMP intenta ser adecuado para implementaciones en distintos tipos de arquitecturas SMPs (shared-memory parallel computer) como lo son máquinas multicore y procesadores multihilos.

OpenMP no es un lenguaje de programación sino consiste de directivas de compilador o "pragmas" que se pueden agregar a un programa secuencial en Fortran, C o C++ para que el trabajo pueda ser compartido a través de hilos que son ejecutados en diferentes procesos o cores.

La inserción adecuada de directivas OpenMP en un programa secuencial permitirá a la aplicación beneficiarse de arquitecturas paralelas de memoria compartida con una mínima modificación del código. La forma más común y fácil de paralelizar código en OpenMP es paralelizando bucles **for**.

El constructor paralelo es la directiva más importante en OpenMP, ya que un programa sin un constructor paralelo será ejecutado secuencialmente.

```
#pragma omp parallel [clause[[,] clause]. . . ]
{
  //codigo...
}
```

Esta construcción se utiliza para especificar los cálculos que se deben ejecutar en paralelo.

La directiva de construcción de bucle **for** hace que las iteraciones se ejecuten en paralelo. En tiempo de ejecución, las iteraciones del bucle se distribuyen a través de los hilos. Su sintaxis se muestra a continuación

```
#pragma omp for [clause[[,] clause]. . . ]
   for ...
```

Cuando se tienen bucles anidados la directiva *collapse*, permite paralelizarlos perfectamente sin necesidad de usar paralelismo anidado. Con esta directiva el compilador forma un solo bucle y luego lo paraleliza. Su sintaxis se muestra a continuación

```
#pragma omp parallel for collapse (2)
for(i=0;i< N; i++)
{
  for(j=0;j< M; j++)
  {
    foo(A,i,j);
  }
}
```

4.2. ¿QUE ES MPI?

MPI ("Message Passing Interface") es una interfaz de programación de aplicaciones (API por sus siglas en inglés) designada para soportar computación distribuida para aplicaciones en C y FORTRAN. Desde 1992 se han creado métodos para la

mayoria de los lenguajes, incluyendo perl, python, ruby, java, etc. Los métodos en C usan el formato MPI_Xxxx en el cual “Xxxx” especifica la operación.

Las funciones MPI_Scatter y MPI_Gather se han utilizado para la implementaciones de MPI. Estas son del tipo de comunicación colectiva y se caracterizan por involucrar la participación de todos los procesos de un comunicador (se encarga de agrupar los procesos implicados en una ejecución paralela).

MPI_Scatter funciona designando al proceso raíz para que envíe la información a todos los procesos de un comunicador). Lo que caracteriza a MPI_Scatter es que envía ”pedazos“ de un arreglo a los diferentes procesos.

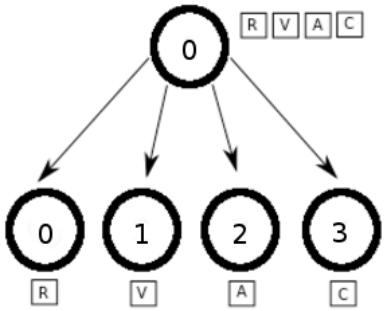

Figura 4–1: MPI_Scatter

En la ilustración, MPI_Scatter toma un arreglo de elementos y distribuye los elementos en orden del rango de los procesos. El primer elemento (R) va al proceso cero, el segundo elemento (V)) va al proceso uno, y así sucesivamente. El proceso raíz (proceso cero) contiene todo el arreglo de elementos y MPI_Scatter copiará el elemento apropiado en el búfer de recepción de los procesos.

El prototipo de la función MPI_Scatter es de la siguiente forma

```
1 MPI_Scatter(
2     void* send_data,
3     int send_count,
4     MPI_Datatype send_datatype,
```

```
5       void* recv_data,
6       int recv_count,
7       MPI_Datatype recv_datatype,
8       int root,
9       MPI_Comm communicator)
```

El primer parámetro, send_data, es un arreglo de datos que reside en el proceso raíz. El segundo y tercer parámetro, send_count y send_datatype, dictan cuantos elementos de un tipo de dato MPI específico se enviarán a cada proceso. En la práctica, send_count es a menudo igual al número de elementos de un arreglo dividido por el número de procesos.

Los parámetros de recepción de la función son casi idénticos a los parámetros de envío. El parámetro recv_data es un búfer de datos que puede almacenar el número de elementos que dice recv_count y que que son del tipo de dato recv_datatype. Los últimos parámetros, raíz y comunicador, indican el proceso raíz que se encarga de la distribución del arreglo y el comunicador en el cual residen los procesos.

MPI_Gather es lo inverso de MPI_Scatter. En lugar de la difusión de elementos de un proceso a muchos procesos, MPI_Gather toma elementos de muchos procesos y los reúne a un solo proceso. A continuación se muestra un ejemplo sencillo del funcionamiento de esta función

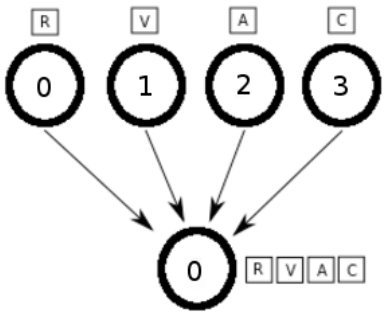

Figura 4–2: MPI_Gather

De manera similar a MPI_Scatter, MPI_Gather toma los elementos de cada proceso y los reúne todos en el proceso raíz. Los elementos son ordenados por el rango de los procesos de los cuales fueron recibidos. El prototipo de la función MPI_Gather es idéntica a la de MPI_Scatter.

```
1 MPI_Gather(
2     void* send_data,
3     int send_count,
4     MPI_Datatype send_datatype,
5     void* recv_data,
6     int recv_count,
7     MPI_Datatype recv_datatype,
8     int root,
9     MPI_Comm communicator)
```

En MPI_Gather, sólo el proceso raíz necesita tener un válido búfer de recepción. Todos los demás procesos que llaman a la función pueden pasar NULL para recv_data.

4.3. ¿QUE ES CUDA?

El modelo de programación CUDA ("Compute Unified Device Architecture"), creado por Nvidia, es un modelo heterogéneo en el cual se usan CPU y GPUs. En CUDA, el *host* refiere a la CPU y su memoria, mientras que el *device* se refiere a la GPU y su memoria.

Códigos ejecutados en el host pueden gestionar la memoria tanto en el host y device, y también ejecutar kernels que son funciones ejecutadas en el device. Estos kernels son ejecutados por muchos hilos de la GPU en paralelo. Los hilos son agrupados en bloques de hasta 512 hilos, que a su vez se organizan en grids. Todos los hilos en un grid ejecutan el mismo kernel. Un grid puede ser uni-, bi-, o tridimensional. El más conveniente para procesar imágenes es un grid bidimensional.

Dada la naturaleza heterogénea del modelo de programación CUDA, una secuencia típica de operaciones para un programa en CUDA C es:

- Declarar y asignar la memoria host y del device.
- Inicializar los datos del host.
- Transferir los datos desde el host al device.
- Ejecutar uno o más kernel.
- Transferir los resultados desde el device al host.

Al trabajar con arreglos bidimensionales la función cudaMallocPitch() es recomendada para asignar memoria lineal, ya que esta se asegura que la alineación se haga adecuadamente para que cumpla los requisitos del Acceso de Memoria del Dispositivo, y garantizar un mejor rendimiento cuando se acceden a las direcciones de fila o se hagan copias entre arreglos bidimensionales y otras regiones de la memoria del dispositivo. Y para liberar la memoria asignada con cudaMallocPitch() se utiliza cudaFree().

La función cudaMemcpy2D() permite copiar datos entre el host y el device. Su sintaxis en la siguiente

```
1 cudaMemcpy2D (
2       void* dst,
3       size_t dpitch,
4       const void* src,
5       size_t spitch,
6       size_t width,
7       size_t height,
8       cudaMemcpyKind kind )
```

donde, *dst* es la dirección de memoria del destino, *dpitch* tamaño de memoria del destino, *src* es la dirección de memoria de la fuente, *spitch* tamaño de memoria de la fuente, *width* es el ancho de la transferencia de la matriz (columnas en bytes), *height* es el largo de la transferencia de la matriz (filas), *kind* es el tipo de traslado.

Capítulo 5
TRANSFORMADA DISCRETA DEL COSENO

En este capítulo se examinan en detalle la DCT y IDCT, pues son las funciones mas usadas en el algoritmo y es de gran interés su optimización.

La **Transformada discreta del coseno** unidimensional es una función lineal $F : \mathbf{R}^N \to \mathbf{R}^N$ definida como

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(u) cos\left(\frac{(2x+1)u\pi}{2N}\right), \quad u = 0, \ldots, N-1 \tag{5.1}$$

La transformada inversa existe y es definida por

$$f(x) = \sum_{x=0}^{N-1} \alpha(u) C(u) cos\left(\frac{(2x+1)u\pi}{2N}\right), \quad x = 0, \ldots, N-1 \tag{5.2}$$

donde

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{n}} & \text{si } u = 0 \\ \sqrt{\frac{2}{n}} & \text{si } u \neq 0 \end{cases} \tag{5.3}$$

La DCT es una función lineal y puede ser representada como una matriz, i.e.,

$$[C_{ij}] = \left[\alpha(i) cos\left(\frac{(2j+1)\pi i}{2N}\right)\right] \qquad i, j = 0, 1, \ldots, N-1$$

y de manera similar para la inversa de la DCT. La DCT bidimensional es una función $F : \mathbf{R}^{\mathbf{N}^2} \to \mathbf{R}^{\mathbf{N}^2}$ la cual cuando es aplicada a una matriz puede ser calculada primero calculando la DCT unidimensional a las filas y luego usando el resultado para calcular la DCT unidimensional de las columnas.

La DCT permite a una imagen ser separada en diferentes bandas de frecuencias. Para el caso de un bloque 8×8 (ver Fig. 5–1), la región **I** o F_L denota los componentes de baja frecuencia del bloque. La región **II** o F_M denota los componentes de frecuencia media, mientras que la región **III** o F_H que denota los componentes de frecuncia alta.

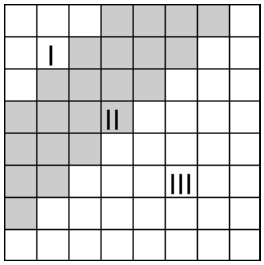

Figura 5–1: Regiones de la DCT

El resultado de aplicar la DCT a un bloque de 8×8 son 1 coeficiente DC y 63 coeficientes AC. El coeficiente DC representa el color promedio del bloque. Los 63 coeficientes AC representan el cambio de color a través del bloque. Coeficientes con baja numeracíon representan cambios de color en bajas frecuencias, o cambios graduales de color a través del bloque. Y coeficientes con numeraciones altas representan el cambio de color de alta frecuencia, o el color que cambia rápidamente de un píxel a otro dentro del bloque.

Dado que un método $\alpha-\beta$ aplica la DCT a bloques 8×8 de una imagen múltiples veces, es de interés minimizar el número de operaciones en sus cálculos. Para esto se usa una idea de Obukhov y Kharlamov [20] la cual es descrita a continuación.

Sea $\gamma(m) = cos\frac{m\pi}{16}$. Y se tiene que

$$\gamma(x+16) = cos\frac{(x+16)}{16}\pi = cos(\frac{x}{16}+\pi) = -cos\frac{x}{16} = -\gamma(x)$$

$$\alpha(x+16i) = \begin{cases} \gamma(x) & \text{si i es par} \\ -\gamma(x) & \text{si i es impar} \end{cases} \tag{5.4}$$

$$\frac{1}{2}\begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \\ \gamma(1) & \gamma(3) & \gamma(5) & \gamma(7) & \gamma(9) & \gamma(11) & \gamma(13) & \gamma(15) \\ \gamma(2) & \gamma(6) & \gamma(10) & \gamma(14) & \gamma(18) & \gamma(22) & \gamma(26) & \gamma(30) \\ \gamma(3) & \gamma(9) & \gamma(15) & \gamma(21) & \gamma(27) & \gamma(33) & \gamma(39) & \gamma(45) \\ \gamma(4) & \gamma(12) & \gamma(20) & \gamma(28) & \gamma(36) & \gamma(44) & \gamma(52) & \gamma(60) \\ \gamma(5) & \gamma(15) & \gamma(25) & \gamma(35) & \gamma(45) & \gamma(55) & \gamma(65) & \gamma(75) \\ \gamma(6) & \gamma(18) & \gamma(30) & \gamma(42) & \gamma(54) & \gamma(66) & \gamma(78) & \gamma(90) \\ \gamma(7) & \gamma(21) & \gamma(35) & \gamma(49) & \gamma(63) & \gamma(77) & \gamma(91) & \gamma(105) \end{bmatrix}$$

usando (5.4) tenemos que la matriz anterior es igual

$$\frac{1}{2}\begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \\ \gamma(1) & \gamma(3) & \gamma(5) & \gamma(7) & \gamma(9) & \gamma(11) & \gamma(13) & \gamma(15) \\ \gamma(2) & \gamma(6) & \gamma(10) & \gamma(14) & -\gamma(2) & -\gamma(6) & -\gamma(10) & -\gamma(14) \\ \gamma(3) & \gamma(9) & \gamma(15) & -\gamma(5) & -\gamma(11) & \gamma(1) & \gamma(7) & \gamma(13) \\ \gamma(4) & \gamma(12) & -\gamma(4) & -\gamma(12) & \gamma(4) & \gamma(12) & -\gamma(4) & -\gamma(12) \\ \gamma(5) & \gamma(15) & -\gamma(9) & \gamma(3) & \gamma(13) & -\gamma(7) & \gamma(1) & \gamma(11) \\ \gamma(6) & -\gamma(2) & -\gamma(14) & \gamma(10) & -\gamma(6) & \gamma(2) & \gamma(14) & -\gamma(10) \\ \gamma(7) & -\gamma(5) & \gamma(3) & -\gamma(1) & -\gamma(15) & \gamma(13) & -\gamma(11) & \gamma(9) \end{bmatrix}$$

Tenemos que:

$\gamma(15) = -\gamma(1)$ $\qquad$ $\gamma(10) = -\gamma(6)$

$\gamma(13) = -\gamma(3)$ $\qquad$ $\gamma(14) = -\gamma(2)$

$\gamma(11) = -\gamma(5)$ $\qquad$ $\gamma(12) = -\gamma(4) = \frac{1}{\sqrt{2}}$

$\gamma(9) = -\gamma(7)$

$$\frac{1}{2}\begin{bmatrix} \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \\ \gamma(1) & \gamma(3) & \gamma(5) & \gamma(7) & -\gamma(7) & -\gamma(5) & -\gamma(3) & -\gamma(1) \\ \gamma(2) & \gamma(6) & -\gamma(6) & -\gamma(2) & -\gamma(2) & -\gamma(6) & \gamma(6) & \gamma(2) \\ \gamma(3) & -\gamma(7) & -\gamma(1) & -\gamma(5) & \gamma(5) & \gamma(1) & \gamma(7) & -\gamma(3) \\ \gamma(4) & -\gamma(4) & -\gamma(4) & \gamma(4) & \gamma(4) & -\gamma(4) & -\gamma(4) & \gamma(4) \\ \gamma(5) & -\gamma(1) & -\gamma(7) & \gamma(3) & -\gamma(3) & -\gamma(7) & \gamma(1) & -\gamma(5) \\ \gamma(6) & -\gamma(2) & \gamma(2) & -\gamma(6) & -\gamma(6) & \gamma(2) & -\gamma(2) & \gamma(6) \\ \gamma(7) & -\gamma(5) & \gamma(3) & -\gamma(1) & \gamma(1) & -\gamma(3) & \gamma(5) & -\gamma(7) \end{bmatrix}$$

$\gamma(1) = cos(\frac{\pi}{16})$

$\gamma(2) = cos(\frac{2\pi}{16}) = cos(\frac{\pi}{8})$

$\gamma(3) = cos(\frac{3\pi}{16})$

$\gamma(4) = cos(\frac{\pi}{4}) = \frac{1}{\sqrt{2}}$

$\gamma(5) = cos(\frac{5\pi}{16})$

$\gamma(6) = cos(\frac{3\pi}{8})$

$\gamma(7) = cos(\frac{7\pi}{16})$

$a = \sqrt{2}\gamma(1)$

$b = \sqrt{2}\gamma(2)$

$c = \sqrt{2}\gamma(3)$

$d = \sqrt{2}\gamma(5)$

$e = \sqrt{2}\gamma(6)$

$f = \sqrt{2}\gamma(7)$

por todo lo anterior y sacando factor común $\frac{1}{\sqrt{8}}$

$$F = \frac{1}{\sqrt{8}} \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a & c & d & f & -f & -d & -c & -a \\ b & e & -e & -b & -b & -e & e & b \\ c & -f & -a & -a & d & a & f & -c \\ \hdashline \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \hdashline 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ d & -a & f & c & -c & -f & a & -d \\ e & -b & b & -e & -e & b & -b & e \\ f & -d & c & -a & a & -c & d & -f \end{array}\right]$$

La forma de la matriz de la DCT unidimensional exhibe varias simetrías que pueden ser aprovechadas. Separando filas pares e impares obtenemos

$$\begin{bmatrix} Y(0) \\ Y(2) \\ Y(4) \\ Y(6) \end{bmatrix} = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ b & e & -e & -b \\ 1 & -1 & -1 & 1 \\ e & -b & b & -e \end{bmatrix} \begin{bmatrix} X(0)+X(7) \\ X(1)+X(6) \\ X(2)+X(5) \\ X(3)+X(4) \end{bmatrix}$$

$$\begin{bmatrix} Y(1) \\ Y(3) \\ Y(5) \\ Y(7) \end{bmatrix} = \frac{1}{\sqrt{8}} \begin{bmatrix} a & -c & d & -f \\ c & f & -a & d \\ d & a & f & -c \\ f & d & c & a \end{bmatrix} \begin{bmatrix} X(0)-X(7) \\ X(2)-X(1) \\ X(4)-X(5) \\ X(5)-X(3) \end{bmatrix}$$

Por lo tanto, la DCT unidimensional aplicada a un vector de longitud 8 puede ser calculada utilizando sólo 28 multiplicaciones y 56 adiciones. El producto habitual de una matriz 8×8 por un vector de longitud 8 requiere 64 multiplicaciones y 56 adiciones.

5.1. TRANSFORMADA INVERSA DEL COSENO

Tenemos que la transformada inversa del coseno es F^t

$$F^t = \frac{1}{\sqrt{8}} \left[\begin{array}{cccc:cccc} 1 & a & b & c & 1 & d & e & f \\ 1 & c & e & -f & -1 & -a & -b & -d \\ 1 & d & -e & -a & -1 & f & b & c \\ 1 & f & -b & -d & 1 & c & -e & -a \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & -f & -b & d & 1 & -c & -e & a \\ 1 & -d & -e & a & -1 & -f & b & -c \\ 1 & -c & e & f & -1 & a & -b & d \\ 1 & -a & b & -c & 1 & -d & e & -f \end{array}\right]$$

si multiplicamos x un vector de longitud 8, por F^t, tenemos que:

$$Y = F^t x$$

$$\begin{aligned} y_0 &= x_0 + ax_1 + bx_2 + cx_3 + x_4 + dx_5 + ex_6 + fx_7 \\ y_1 &= x_0 + cx_1 + ex_2 - fx_3 - x_4 - ax_5 - bx_6 - dx_7 \\ y_2 &= x_0 + dx_1 - ex_2 - ax_3 - x_4 + fx_5 + bx_6 + cx_7 \\ y_3 &= x_0 + fx_1 - bx_2 - dx_3 + x_4 + cx_5 - ex_6 - ax_7 \\ y_4 &= x_0 - fx_1 - bx_2 + dx_3 + x_4 - cx_5 - ex_6 + ax_7 \\ y_5 &= x_0 - dx_1 - ex_2 + ax_3 - x_4 - fx_5 + bx_6 - cx_7 \\ y_6 &= x_0 - cx_1 + ex_2 + fx_3 - x_4 + ax_5 - bx_6 + dx_7 \\ y_7 &= x_0 - ax_1 + bx_2 - cx_3 + x_4 - dx_5 + ex_6 - fx_7 \end{aligned}$$

Haciendo cambio de variables tenemos

$$
\begin{aligned}
y_0 &= x_0 + a_1 + b_2 + c_3 + x_4 + d_5 + e_6 + f_7 \\
y_1 &= x_0 + c_1 + e_2 - f_3 - x_4 - a_5 - b_6 - d_7 \\
y_2 &= x_0 + d_1 - e_2 - a_3 - x_4 + f_5 + b_6 + c_7 \\
y_3 &= x_0 + f_1 - b_2 - d_3 + x_4 + c_5 - e_6 - a_7 \\
y_4 &= x_0 - f_1 - b_2 + d_3 + x_4 - c_5 - e_6 + a_7 \\
y_5 &= x_0 - d_1 - e_2 + a_3 - x_4 - f_5 + b_6 - c_7 \\
y_6 &= x_0 - c_1 + e_2 + f_3 - x_4 + a_5 - b_6 + d_7 \\
y_7 &= x_0 - a_1 + b_2 - c_3 + x_4 - d_5 + e_6 - f_7
\end{aligned}
$$

donde

$a_1 = ax_1$	$b_2 = bx_2$	$a_3 = ax_3$	$a_5 = ax_5$	$b_6 = bx_6$	$a_7 = ax_7$
$c_1 = cx_1$	$e_2 = ex_2$	$c_3 = cx_3$	$c_5 = cx_5$	$e_6 = ex_6$	$c_7 = cx_7$
$d_1 = dx_1$		$d_3 = dx_3$	$d_5 = dx_5$		$d_7 = dx_7$
$f_1 = fx_1$		$f_3 = fx_3$	$f_5 = fx_5$		$f_7 = fx_7$

De igual forma se tiene que la IDCT unidimensional aplicada a un vector de longitud 8 puede ser calculada utilizando sólo 28 multiplicaciones y 56 adiciones. Y el producto habitual de una matriz 8×8 por un vector de longitud 8 requiere 64 multiplicaciones y 56 adiciones.

Capítulo 6
ALGORITMOS

En este capítulo se describen de manera general las diferentes implementaciones del métod $\alpha - \beta$, donde los resultados que presenta son para el caso especial donde los valores de alfa y beta son lo mismo para cada uno de los bloques.

6.1. IMPLEMENTACIÓN PARALELA

Dadas dos imágenes en escala de grises, cuadradas y del mismo tamaño, una imagen I y otra la marca de agua W. Un algoritmo $\alpha - \beta$ inserta W en I (1) particionando tanto I como W en bloques de 8×8 píxeles y se calcula la DCT de cada bloque, I_{ij} y W_{ij}, (2) se calcula $\alpha_{ij} = \alpha(I_{ij}, W_{ij})$ $\beta_{ij} = \beta(I_{ij}, W_{ij})$ (3) se reemplaza cada $DCT(I_{ij})$ en la imagen por $\alpha DCT(I_{ij}) + \beta DCT(W_{ij})$ (4) se calcula la inversa de la DCT de cada bloque $\alpha DCT(I_{ij}) + \beta DCT(W_{ij})$ y se concatenan los resultados.

El siguiente es un ejemplo donde $\alpha = \alpha_{ij} = 0{,}9$ y $\beta = \beta_{ij} = 0{,}14$ para todos los i, j. El grado de visibilidad de la marca de agua es determinado por los valores de α y β.

Figura 6–1: (a) Imagen, (b) Marca de agua, (c) Imagen con la marca de agua

Empecemos con una versión secuencial de este algoritmo para una imagen de tamaño $dim \times dim$:

Algorithm 1 Secuencial

1: **procedure** WATERMARKING$(I, W, dim, \alpha, \beta)$
2: $m = \frac{dim}{8}$
3: **for** $i = 0 : m - 1$ **do**
4: **for** $j = 0 : m - 1$ **do**
5: $X_{ij} = DCT(I_{ij})$
6: $Y_{ij} = DCT(W_{ij})$
7: Compute $\alpha = \alpha(I_{ij}, W_{ij})$ and $\beta = \beta(I_{ij}, W_{ij})$
8: $Z_{ij} = \alpha X_{ij} + \beta Y_{ij}$
9: $Z_{ij} = IDCT(Z_{ij})$
return Z

Cada par (I_{ij}, W_{ij}) de bloques de imágenes en el algoritmo de arriba puede ser procesado independientemente de los otros y por lo tanto en paralelo y una eficiente implementación consiste en determinar en como los recursos disponibles pueden ser usados de la forma más efectiva para alcanzar esto. Vamos cómo se puede hacer esto en OpenMP, MPI y CUDA

6.2. OPENMP

En el procedimiento de WATERMARKING de arriba, cada una de las (i, j) iteraciones es independiente de las otras e idealmente, sería conveniente usar solamente un bucle **for** paralelo. Sin embargo, debido a la doble indexación, se deberá usar dos bucles **for** anidados y en OpenMP es posible paralelizar solamente el bucle **for** exterior. Sin embargo, versiones recientes de OpenMP permiten paralelizar múltiples bucles sin introducir paralelismo anidado.

Algorithm 2 Versión OpenMP

1: **procedure** WATERMARKING(I, W, dim, α, β)
2: $m = \frac{dim}{8}$
3: #**pragma omp parallel for collapse**(2)
4: **for** $i = 0 : m - 1$ **do**
5: **for** $j = 0 : m - 1$ **do**
6: $X_{ij} = DCT(I_{ij})$
7: $Y_{ij} = DCT(W_{ij})$
8: Compute $\alpha = \alpha(I_{ij}, W_{ij})$ and $\beta = \beta(I_{ij}, W_{ij})$
9: $Z_{ij} = \alpha X_{ij} + \beta Y_{ij}$
10: $Z_{ij} = IDCT(Z_{ij})$
return Z

6.3. MPI

En la implementación MPI del algoritmo de marca de agua se particiona tanto la imagen como el logo en n secciones de filas de bloques 8×8, donde n es el número de procesadores. El procesador, dígase p_0, que inicialmente contiene ambas imágenes, I y W, envía a cada uno de los n procesadores un par de secciones (una de la imagen y otra del logo). Cada procesador entonces procesa lo que se le envían de las dos imágenes como en la versión secuencial y luego se envían sus resultados a p_0 el cual ensambla completamente la imagen con la marca de agua.

Algorithm 3 Versión MPI

1: **procedure** WATERMARKING(I, W, dim, α, β)
2: $n =$ number of processors
3: $m1 = \frac{dim}{8}$
4: $m2 = \frac{m1}{n}$
5: send a section $I^{(p)}$ of $m2$ rows of 8×8 blocks of I to each processor p
6: send a section $W^{(p)}$ of $m2$ rows of 8×8 blocks of W to each processor p
7: **for** $i = 0 : m2 - 1$ **do**
8: **for** $j = 0 : m1 - 1$ **do**
9: $X_{ij}^{(p)} = DCT(I_{ij}^{(p)})$
10: $Y_{ij}^{(p)} = DCT(W_{ij}^{(p)})$
11: Compute $\alpha = \alpha(I_{ij}, W_{ij})$ and $\beta = \beta(I_{ij}, W_{ij})$
12: $Z_{ij}^{(p)} = \alpha X_{ij}^{(p)} + \beta Y_{ij}^{(p)}$
13: $Z_{ij}^{(p)} = IDCT(Z_{ij}^{(p)})$
14: receive each $Z^{(p)}$ in Z

6.4. CUDA

Con CUDA se alcanza el objetivo original de asignar un hilo a cada iteración del bucle **for** anidado de la versión secuencial del algoritmo. Para esto se hace uso del código Nvidia [20] para el calculo de la DCT y IDCT en bloques 8×8.

Algorithm 4 Versión CUDA

1: **procedure** WATERMARKING(I, W)
2: **for** each thread-block in *block-grid* **do** in parallel
3: $X = kernel\ DCT(I_{thread})$
4: $Y = kernelDCT(W_{thread})$
5: $\alpha = kernel\ \alpha(I_{ij}, W_{ij})$
6: $\beta = kernel\ \beta(I_{ij}, W_{ij})$
7: $Z_{thread} = kernel\ linearcomb(\alpha, X, \beta, Y)$
8: $Z_{thread} = kernel\ IDCT(Z_{thread})$
return Z

6.5. STAMPEDE

Los experimentos se llevaron a cabo en el superordenador Stampede [3], situada en el Centro de Cómputo Avanzado de Texas (TACC) y patrocinado por la **Ciencia Extrema e Ingeniería del Medio Ambiente (XSEDE por sus siglas en inglés) con financiamiento de la Fundación Nacional para la Ciencia con número de concesión ACI-1053575**. La supercomputadora Stampede es un sistema cluster de 10 PFLOPS (PF) Dell Linux que consiste de 6400+ nodos Dell PowerEdge, cada uno de los cuales tiene 2 procesadores Intel Xeon E5 (Sandy Bridge) y un Intel Xeon Coprocesador Phi (aquitectura MICZ) y un total de 522080 cores de procesamiento.

La siguiente figura Fig. 6–2 muestra como es un nodo en Stampede. Cada nodo en Stampede tiene su propia memoria local, dos sockets por nodo y cada socket tiene 8 cores. Cada nodo puede trabajar con memoria compartida o distribuida.

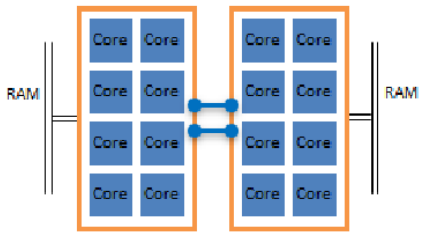

Figura 6–2: Nodo en Stampede

6.6. RESULTADOS EXPERIMENTALES

Para la implementación se escogió un representante de la familia de algoritmos $\alpha - \beta$ en el cual los valores de α y β son el mismo para todos los bloques (como por ejemplo en la Fig. 6.12), sustituyendo así el paso del computo de α y β en los algoritmos anteriores por entradas.

Probamos nuestros programas OpenMP, MPI y CUDA con imágenes en escala de grises de 512, 1024, 2048, 4096, y 8192 píxeles cuadrados. Para OpenMP y MPI se utilizó un máximo de los 16 cores por nodo. Para CUDA usamos un nodo de cómputo con una GPU.

Se observará que el comportamiento de 16 núcleos a veces puede ser errática, especialmente para imágenes pequeñas. Esto se debe a que algunos de los recursos de un nodo son necesariamente dedicado a otros procesos intrínsecos del sistema.

6.7. RESULTADOS OPENMP

Los tiempos para OpenMP se representan en la Fig. 6–3.

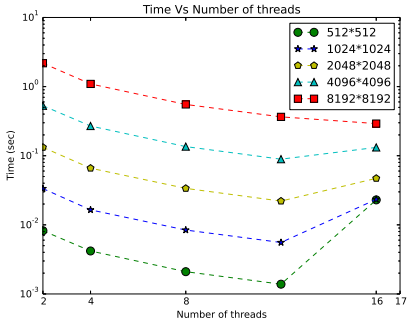

Figura 6–3: openMP times

Speedups para OpenMP se dan en Fig. 6–4.

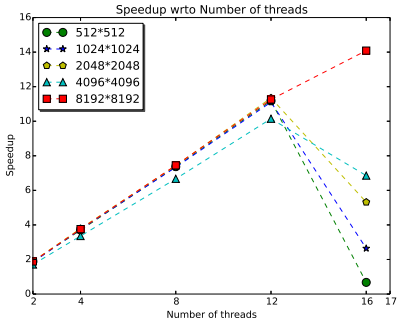

Figura 6–4: openMP Speedup

6.8. RESULTADOS MPI

Los tiempos y los speedups con respecto a las tareas MPI de 1, 2, 4, y 8 nodos se dan en las Fig. 6–5 hasta 6–12.

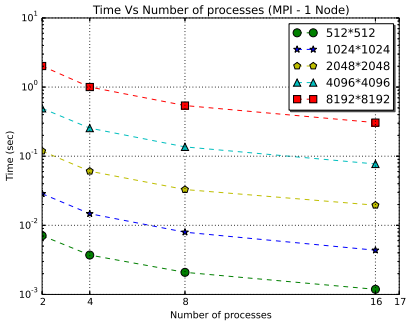

Figura 6–5: MPI 1 Node

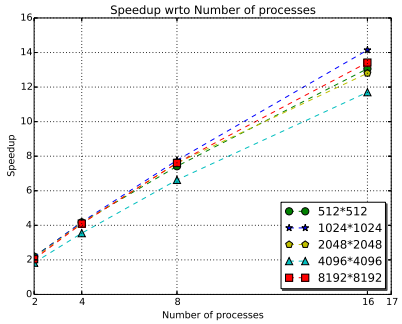

Figura 6–6: Speedup 1 Node

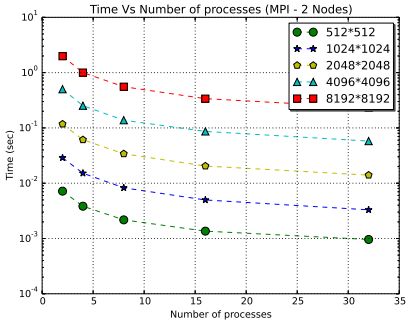

Figura 6–7: MPI 2 Nodes

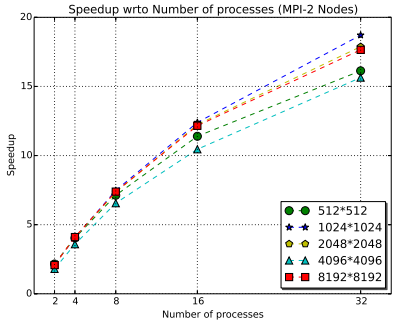

Figura 6–8: Speedup 2 Nodes

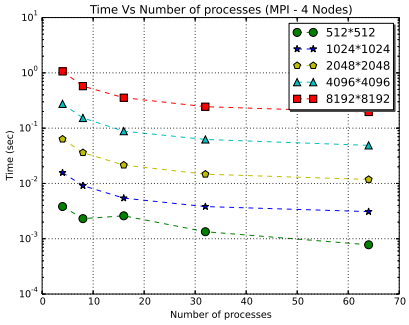

Figura 6–9: MPI 4 Nodes

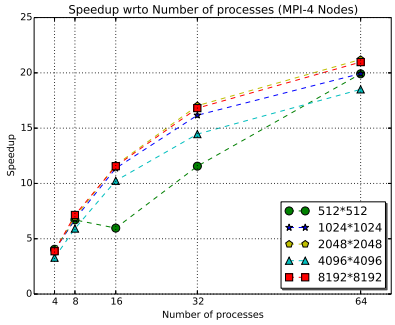

Figura 6–10: Speedup 4 Nodes

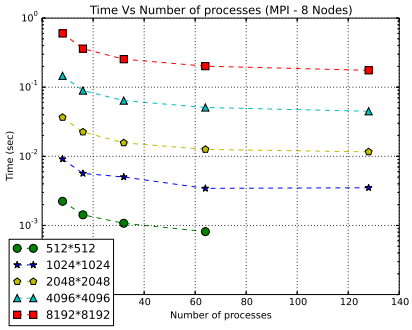

Figura 6–11: MPI 8 Nodes

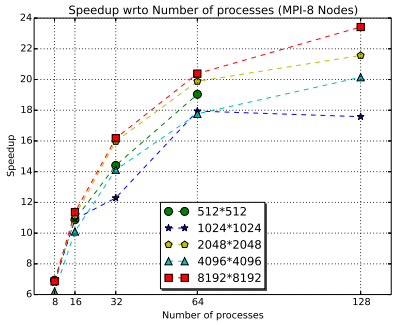

Figura 6–12: Speedup 8 Nodes

Utilizando todos los 16 cores en cada nodo, speedups con respecto al número de nodos se dan en las Fig. 6–13 y 6–14

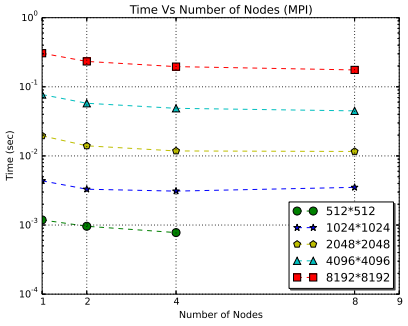

Figura 6–13: MPI times

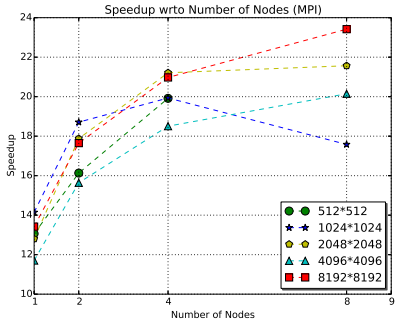

Figura 6–14: MPI Speedups

6.9. OPENMP vs MPI

En Fig. 6–15 y 6–19 se compara el rendimiento de OpenMP para 2, 4, 8 y 16 hilos con MPI en un nodo para 2, 4, 8, 16 tareas, respectivamente, y vemos que el rendimiento de los dos son casi lo mismo hasta 8 hilos/tareas. Para 16 hilos/tareas, MPI gana para tamaños de hasta 4096×4096, pero la brecha se reduce para tamaños

mayores de la imagen y OpenMP gana por un margen muy estrecho para imágenes de tamaño 8192×8192.

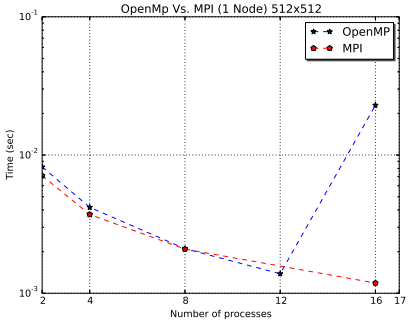

Figura 6–15: MPI vs. openMP (512X512)

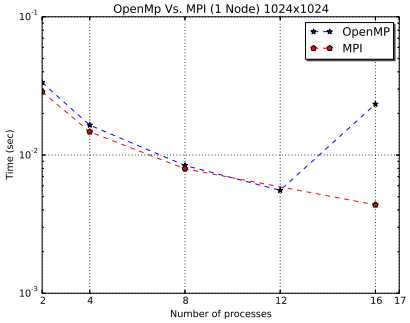

Figura 6–16: MPI vs. openMP (1024X1024)

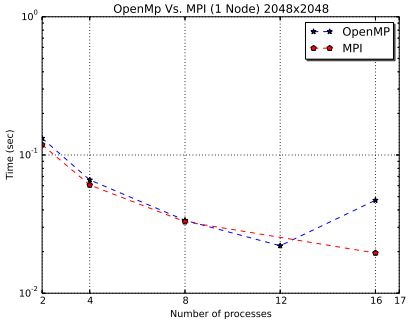

Figura 6–17: MPI vs. openMP (2048X2048)

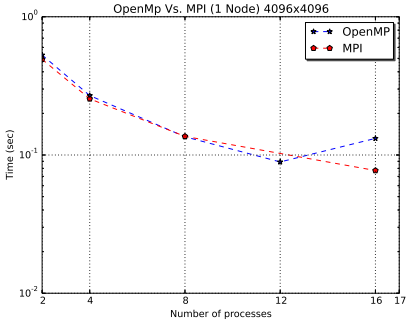

Figura 6–18: MPI vs. openMP (4096X4096)

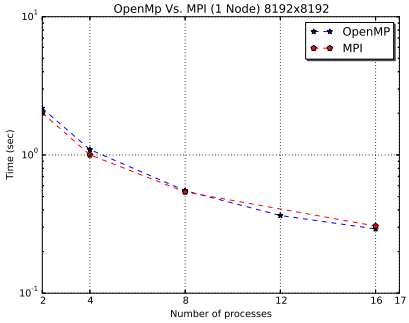

Figura 6–19: MPI vs. openMP (8192X8192)

.

6.10. HÍBRIDO

Se desarrolló un programa OpenMP MPI híbrido en la forma estándar, empezando con la versión MPI y distribuyendo secciones de bloques de 8×8 píxeles de entre los nodos del mismo modo que hicimos en la versión MPI excepto que ahora cada nodo hace su trabajo en paralelo usando OpenMP como en la versión de OpenMP. Desafortunadamente, esta versión bajo el desempeño de las versiones de OpenMP y MPI. Esto no es realmente sorprendente. De hecho, hay casos (ver *e.g.*,[12]) en los que la versión híbrida es más lenta que OpenMP y MPI. La versión híbrida mejora el rendimiento mediante la reducción de las comunicaciones entre los nodos y aumenta las oportunidades de paralelismo y ninguna de estas oportunidades existen en la versión MPI de nuestro algoritmo. Además, resulta que la operaciones **scatter** y **gather** son mucho más lentas en la versión híbrida que en la versión MPI cuando se aplica a los mismos conjuntos de datos.

La siguientes tablas muestran los tiempos para imágenes de tamaño 1024×1024 para el programa híbrido haciendo uso de 2 nodos - 2 procesos y 2,4,8,16 hilos OpenMP en cada nodo

Cuadro 6–1: Tiempos usando directivas de compilación -O2

Tamaño 1024x1024 (100 iteraciones)	
Hilos	**Tiempo (seg)**
2	0.032448
4	0.030236
8	0.028600
16	0.033182

Cuadro 6–2: Tiempos usando directivas de compilación -O0

Tamaño 1024x1024 (100 iteraciones)	
Hilos	**Tiempo (seg)**
2	0.133433
4	0.068759
8	0.052069
16	0.063433

La siguientes tablas muestran los tiempos para imágenes de tamaño 1024 × 1024 para el programa híbrido haciendo uso de 1 node - 4 procesos y 2,4,8,16 hilos OpenMP en cada nodo. Se esperaba que los tiempos se comportaran de manera similar a OpenMP, pero los resultados obtenidos no fueron de esa manera.

Cuadro 6–3: Tiempos usando directivas de compilación -O2

Tamaño 1024x1024 (100 iteraciones)	
Hilos	**Tiempo(seg)**
2	0.010021
4	0.0099597
8	0.043508
16	0.079079

Cuadro 6–4: Tiempos usando directivas de compilación -O0

Tamaño 1024x1024 (100 iteraciones)	
Hilos	**Tiempo(seg)**
2	0.061161
4	0.032894
8	0.062737
16	0.079079

Las opciones de compilación -O2 quiere decir “optimizar”, haciendo referencia a el tamaño del código y tiempo de ejecución. Y la de -O0 quiere decir “sin optimizar”.

6.11. RESULTADOS CUDA

Las comparaciones entre las cuatro implementaciones se representan en Fig. 6–20.

Nuestra implementación CUDA en un solo GPU era mucho más rápida que OpenMP y MPI con cualquier número de nodos hasta 8.

La siguiente tabla muestra los tiempos para imágenes de tamaño 4096×4096 de las cuatro versiones diferentes en un solo nodo, donde OpenMP utiliza 16 hilos y MPI 16 procesos

Cuadro 6–5: Tiempos CUDA (tamaño 4096x4096)

Version	**Time (Seconds)**
Secuencial	0.903651
OpenMP	0.131847
MPI	0.0772164
CUDA	0.0031519

Por lo tanto, en un solo nodo, CUDA es 287 veces más rápido que la versión secuencial, 42 veces más rápido que OpenMP, y 24 veces más rápido que MPI. El

tiempo más rápido, ,0448525 seg, para MPI ocurrió con 8 nodos y utilizando 128 procesos. Así CUDA en un solo nodo es 14 veces más rápido que MPI en 8 nodos.

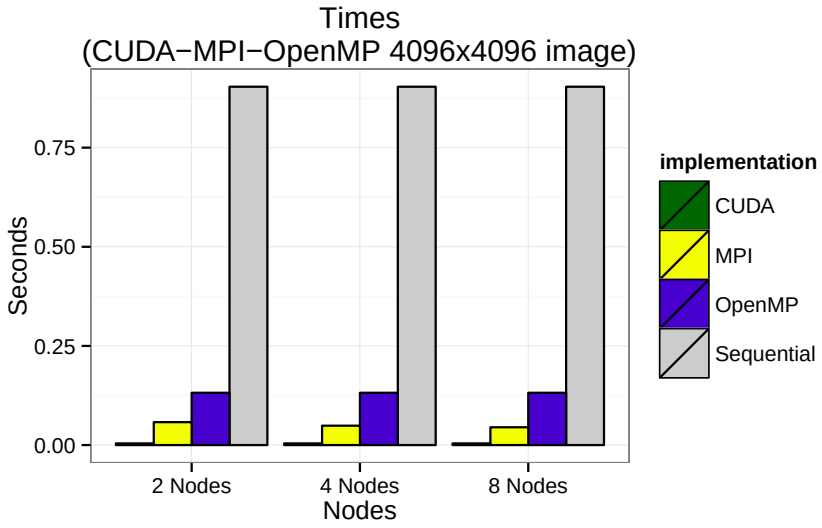

Figura 6–20: Cuda-MPI-openMP 4096X4096 image

6.12. PUNTOS DE REFERENCIA DE CALIDAD (BENCHMARKING)

El uso de puntos de referencia de calidad para evaluar esquemas de marcas de agua podrían ayudar a determinar cuan robusta es una técnica. StirMark [23][22], es un benchmark que divide los ataques en las siguientes 9 categorías: mejora de la señal, compresión, escalado, recorte, corte, rotación, transformaciones lineales, otras transformaciones geométricas, y distorsiones geométricas aleatorias.

Las siguientes imágenes muestran los resultados de aplicar StirMark al algoritmo $\alpha - \beta$ con imágenes de tamanño 1024×1024 y con $\alpha = 0{,}9$ y $\beta = 0{,}2$. Se seleccionarón imágenes que fuerón expuestas a ataques como suavizado por filtro de la mediana utilizando una mascara de 7×7, compresión JPEG manteniendo un 20 % de la información de la imagen original, distorsinamiento aleatorio (0.95), rotación + escalado (0.75), convolución utilizando filtro gaussiano y adición de ruido (20 %). Los ataques de convolución y ruido ocasionarón el daño total de la marca de agua, no permitiendo la extracción de la misma.

Figura 6–21: (a) Imagen, (b) Marca de agua, (c) Imagen con la marca de agua

Figura 6–22: (a) Imagen escalada, (b) imagen suavizada, (c) imagen comprimida (jpeg 20 %) (d) imagen distorsionada

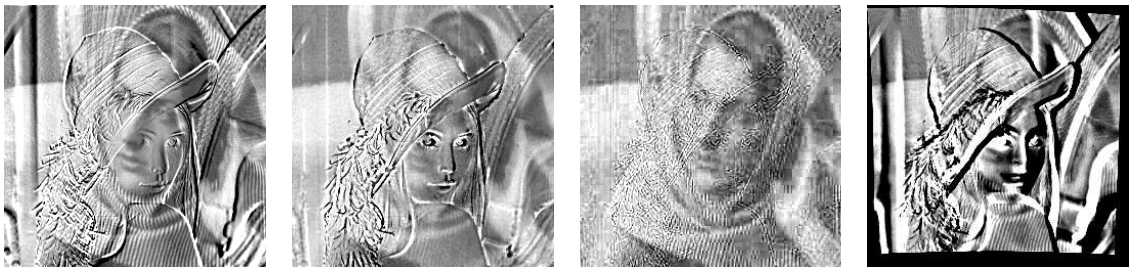

Figura 6–23: (a) (b),(c),(d) Marca de agua extraida

La siguiente tabla muestra los resultados de las metricas **PSNR** y **NC**, de las imágenes de arriba

Imagen	**PSNR**	**NC**
Imagen con marca de agua	36.72	0.97
Imagen comprimida	34.20	1.45
Imagen suavizada	33.86	1.44
Imagen rotada+escalada	33.53	1.04
Imagen distorsionada aleatoriamente	30.21	1.09
Imagen con ruido	28.23	0.56
Imagen + convolución	27.51	0

Por lo tanto, se puede concluir que el método $\alpha - \beta$ para marcas de agua mostŕo resistencia a ataques como compresión, suavizado, rotación + escalado y distorsión ya que los valores para las métricas PSNR y NC se encuentran entre valores razonables. En cambio, este no mostró resistencia a ataques como convolución y adición de ruido, dado que los valores de PSNR y NC no fueron los esperados. El propietario espera reconocer su marca de agua luego de ser extraída y ser sometida a ataques.

Capítulo 7
CONCLUSIONES Y TRABAJOS FUTUROS

Se ha mostrado cómo una familia de algoritmos de marca de aguas comúnmente utilizada en el dominio de frecuencia se puede implementar en paralelo y se han comparado implementaciones en OpenMP, MPI, CUDA de un simple representante de la familia de algoritmos $\alpha - \beta$. Se ha encontrado que la implementación CUDA en un solo GPU corre mucho más rápido que la versión OpenMP y la versión MPI incluso cuando se ejecuta en 8 nodos. Por lo tanto la implementación más rápida se ejecuta en un solo nodo equipado con un GPU. Para un nodo sin gpu, hay ligeras diferencias entre OpenMP y MPI, obteniendo que OpenMP gana sólo para imágenes grandes y MPI exhibiendo una aceleración más regular.

Se pudo observar, que el representante de la familia de algoritmos $\alpha - \beta$ a pesar de su sencillez, es resistente a ataques como compresión, distorsión, rotación mas escalado y suavizado.

Además, los métodos de paralelización que se han implementado para el caso sencillo $\alpha - \beta$ se podría aplicar a cualquier otro método $\alpha - \beta$ y como trabajo futuro seria de interés hacerlo.

APÉNDICES

Apéndice A
TABLAS DE TIEMPOS

OpenMP

Cuadro A–1: Tiempos de OpenMP de 2-16 procesos (tamaño 512x512)

Tamaño 512x512 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.015487		
2	0.00821656	1.8848520549	0.9424260274
4	0.00418677	3.699032906	0.9247582265
8	0.00210072	7.3722342816	0.9215292852
16	0.0229523	0.6747471931	0.0421716996

Cuadro A–2: Tiempos de OpenMP de 2-16 procesos (tamaño 1024x1024)

Tamaño 1024x1024 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.061706		
2	0.0334679	1.8437368344	0.9218684172
4	0.0164662	3.7474341378	0.9368585345
8	0.00840872	7.3383344909	0.9172918114
16	0.0233544	2.6421573665	0.1651348354

Cuadro A–3: Tiempos de OpenMP de 2-16 procesos (tamaño 2048x2048)

Tamaño 2048x2048 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.250255		
2	0.132183	1.893246484	0.946623242
4	0.0660235	3.7903928147	0.9475982037
8	0.0336929	7.4275292421	0.9284411553
16	0.0470662	5.3170852969	0.3323178311

Cuadro A–4: Tiempos de OpenMP de 2-16 procesos (tamaño 4096x4096)

Tamaño 4096x4096 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.903651		
2	0.528831	1.708770855	0.8543854275
4	0.269123	3.3577620642	0.839440516
8	0.135515	6.6682728849	0.8335341106
16	0.131847	6.8537850691	0.4283615668

Cuadro A–5: Tiempos de OpenMP de 2-16 procesos (tamaño 8192x8192)

Tamaño 8192x8192 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	4.10829		
2	2.17845	1.8858775735	0.9429387868
4	1.09327	3.7577999945	0.9394499986
8	0.551706	7.4465204294	0.9308150537
16	0.291638	14.0869502603	0.8804343913

MPI 1 NODO

Cuadro A–6: Tiempos de 1 nodo de MPI y 2-16 procesos (tamaño 512x512)

Tamaño 512x512 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.015487		
2	0.0070819	2.186842514	1.093421257
4	0.00371433	4.1695272095	1.0423818024
8	0.0020878	7.4178561165	0.9272320146
16	0.0011856	13.0625843455	0.8164115216

Cuadro A–7: Tiempos de 1 nodo de MPI y 2-16 procesos (tamaño 1024x1024)

Tamaño 1024x1024 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.061706		
2	0.0286325	2.1551034663	1.0775517332
4	0.0147281	4.1896782341	1.0474195585
8	0.00794955	7.7622003761	0.970275047
16	0.00436301	14.1429884415	0.8839367776

Cuadro A–8: Tiempos de 1 nodo de MPI y 2-16 procesos (tamaño 2048x2048)

Tamaño 2048x2048 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.250255		
2	0.118597	2.1101292613	1.0550646306
4	0.0607032	4.122599797	1.0306499493
8	0.0329241	7.6009670728	0.9501208841
16	0.0195461	12.8033213787	0.8002075862

Cuadro A–9: Tiempos de 1 nodo de MPI y 2-16 procesos (tamaño 4096x4096)

Tamaño 4096x4096 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.903651		
2	0.492212	1.8358979464	0.9179489732
4	0.254868	3.5455647629	0.8863911907
8	0.136352	6.627339533	0.8284174416
16	0.0772164	11.7028377391	0.7314273587

Cuadro A–10: Tiempos de 1 nodo de MPI y 2-16 procesos (tamaño 8192x8192)

Tamaño 8192x8192 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	4.10829		
2	2.01672	2.0371147209	1.0185573605
4	1.00179	4.1009493008	1.0252373252
8	0.540184	7.6053529908	0.9506691239
16	0.306132	13.4199952961	0.838749706

MPI 2 NODO

Cuadro A–11: Tiempos de 2 nodos de MPI y 2-32 procesos (tamaño 512x512)

Tamaño 512x512 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.015487		
2	0.00718253	2.1562040117	1.0781020058
4	0.00383562	4.0376783936	1.0094195984
8	0.00217322	7.1262918618	0.8907864827
16	0.00135912	11.3948731532	0.7121795721
32	0.0009601	16.1306113946	0.5040816061

Cuadro A–12: Tiempos de 2 nodos de MPI y 2-32 procesos (tamaño 1024x1024)

Tamaño 1024x1024 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.061706		
2	0.028834	2.1400430048	1.0700215024
4	0.0151675	4.0683039393	1.0170759848
8	0.00825719	7.4730023168	0.9341252896
16	0.0049781	12.3954922561	0.774718266
32	0.00329807	18.7097302362	0.5846790699

Cuadro A–13: Tiempos de 2 nodos de MPI y 2-32 procesos (tamaño 2048x2048)

Tamaño 2048x2048 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.250255		
2	0.117395	2.1317347417	1.0658673708
4	0.0610817	4.0970536184	1.0242634046
8	0.0340606	7.3473456134	0.9184182017
16	0.0204441	12.2409399289	0.7650587456
32	0.0139985	17.8772725649	0.5586647677

Cuadro A–14: Tiempos de 2 nodos de MPI y 2-32 procesos (tamaño 4096x4096)

Tamaño 4096x4096 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.903651		
2	0.501098	1.8033418613	0.9016709306
4	0.251941	3.5867564231	0.8966891058
8	0.137856	6.5550356894	0.8193794612
16	0.0864331	10.4549183125	0.6534323945
32	0.0577792	15.6397284836	0.4887415151

Cuadro A–15: Tiempos de 2 nodos de MPI y 2-32 procesos (tamaño 8192x8192)

Tamaño 8192x8192 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	4.10829		
2	1.98049	2.0743805826	1.0371902913
4	1.00085	4.1048009192	1.0262002298
8	0.555744	7.392414493	0.9240518116
16	0.337959	12.1561787081	0.7597611693
32	0.232793	17.6478244621	0.5514945144

MPI 4 NODO

Cuadro A–16: Tiempos de 4 nodos de MPI y 4-64 procesos (tamaño 512x512)

Tamaño 512x512 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.015487		
4	0.00382541	4.0484549369	1.0121137342
8	0.00231021	6.7037195753	0.8379649469
16	0.00259775	5.9616976229	0.3726061014
32	0.00133944	11.5622946903	0.3613217091
64	0.00077762	19.9158972249	0.3111858941

Cuadro A–17: Tiempos de 4 nodos de MPI y 4-64 procesos (tamaño 1024x1024)

Tamaño 1024x1024 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.061706		
4	0.0156426	3.9447406441	0.986185161
8	0.00912524	6.7621235168	0.8452654396
16	0.00542634	11.3715690502	0.7107230656
32	0.00381653	16.1680898617	0.5052528082
64	0.00309564	19.9331963665	0.3114561932

Cuadro A–18: Tiempos de 4 nodos de MPI y 4-64 procesos (tamaño 2048x2048)

Tamaño 2048x2048 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.250255		
4	0.0633351	3.9512845168	0.9878211292
8	0.0360708	6.937883274	0.8672354093
16	0.021465	11.6587467971	0.7286716748
32	0.014693	17.03226026	0.5322581331
64	0.0117992	21.2094887789	0.3313982622

Cuadro A–19: Tiempos de 4 nodos de MPI y 4-64 procesos (tamaño 4096x4096)

Tamaño 4096x4096 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.903651		
4	0.274844	3.2878687546	0.8219671887
8	0.15266	5.9193698415	0.7399212302
16	0.0882411	10.2407041617	0.6400440101
32	0.0624851	14.4618637083	0.4519332409
64	0.0488296	18.5062134443	0.2891595851

Cuadro A–20: Tiempos de 4 nodos de MPI y 4-64 procesos (tamaño 8192x8192)

Tamaño 8192x8192 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	4.10829		
4	1.06178	3.8692478668	0.9673119667
8	0.574605	7.1497637508	0.8937204688
16	0.355404	11.5594928588	0.7224683037
32	0.244378	16.8112105018	0.5253503282
64	0.195896	20.9717911545	0.3276842368

MPI 8 NODO

Cuadro A–21: Tiempos de 8 nodos de MPI y 8-128 procesos (tamaño 512x512)

Tamaño 512x512 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.015487		
8	0.00223418	6.9318497167	0.8664812146
16	0.00142424	10.8738695725	0.6796168483
32	0.00107487	14.4082540214	0.4502579382
64	0.00081354	19.0365562849	0.297446192
128	0.00145917	10.6135679873	0.0829184999

Cuadro A–22: Tiempos de 8 nodos de MPI y 8-128 procesos (tamaño 1024x1024)

Tamaño 1024x1024 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.061706		
8	0.00912369	6.7632723164	0.8454090395
16	0.00565647	10.9089237634	0.6818077352
32	0.00502031	12.2912728497	0.3841022766
64	0.00343891	17.9434762759	0.2803668168
128	0.00350978	17.581158933	0.1373528042

Cuadro A–23: Tiempos de 8 nodos de MPI y 8-128 procesos (tamaño 2048x2048)

Tamaño 2048x2048 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.250255		
8	0.0365904	6.8393622371	0.8549202796
16	0.0224872	11.1287754812	0.6955484676
32	0.0156713	15.9690006573	0.4990312705
64	0.0125871	19.8818631774	0.3106541121
128	0.011604	21.5662702516	0.1684864863

Cuadro A–24: Tiempos de 8 nodos de MPI y 8-128 procesos (tamaño 4096x4096)

Tamaño 4096x4096 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	0.903651		
8	0.146051	6.1872291186	0.7734036398
16	0.0894386	10.1035906197	0.6314744137
32	0.0639936	14.1209589709	0.4412799678
64	0.0508606	17.7672107683	0.2776126683
128	0.0448525	20.1471712837	0.1573997757

Cuadro A–25: Tiempos de 8 nodos de MPI y 8-128 procesos (tamaño 8192x8192)

Tamaño 8192x8192 (100 iteraciones)			
Procesos	Tiempo	Speed-up	Eficiencia
Secuencial	4.10829		
8	0.598899	6.8597376185	0.8574672023
16	0.361583	11.3619556229	0.7101222264
32	0.254024	16.1728419362	0.5054013105
64	0.201524	20.3861078581	0.3185329353
128	0.175442	23.4167987141	0.18294374

Bibliografía

[1] M. Abdullatif, A.M. Zeki, J. Chebil, and T.S. Gunawan. Properties of digital image watermarking. In *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*, pages 235–240, March 2013.

[2] Varios Autores. The cimg library. url http://cimg.sourceforge.net/, 2000.

[3] Varios Autores. Stampede user guide. url https://portal.tacc.utexas.edu/user-guides/stampede, 2011-2015.

[4] Biao bing Huang and Shao xian Tang. A Contrast-Sensitive Visible Watermarking Scheme. *IEEE Multimedia*, 13:60–66, 2006.

[5] J. J. Chen, T. M. Ng, A. Lakshminarayanan, and H. K. Garg. Adaptive Visible Watermarking Using Otsu's Thresholding. In *International Conference on Computational Intelligence and Software Engineering*, 2009.

[6] I. Cox, M. Miller, J. Bloom, and Chris Honsinger. Digital Watermarking. *Journal of Electronic Imaging*, 11, 2002.

[7] I. J. Cox, M. Miller, and J. Bloom. Watermarking applications and properties. 2000.

[8] Ingemar J. Cox, Joe Kilian, Frank Thomson Leighton, and Talal Shamoon. *A Secure, Robust Watermark for Multimedia*. 1996.

[9] R. Dugad, K. Ratakonda, and N. Ahuja. A new wavelet-based scheme for watermarking images. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, volume 2, pages 419–423 vol.2, Oct 1998.

[10] E. Garcia-Cano, R. Bassem, and R. Sabourin. A parallel watermarking application on a gpu. *Ingenio Magno*, 3, 2012.

[11] FRANK HARTUNG and MARTIN KUTTER. Multimedia watermarking techniques. *Proceedings of The IEEE*, 87:1079–1107, 1999.

[12] Yun He and Chris H. Q. Ding. MPI and OpenMP paradigms on cluster of SMP architectures: the vacancy tracking algorithm for multi-dimensional array transposition. In *Supercomputing Conference*, pages 1–14, 2002.

[13] Chiou-Ting Hsu and Ja-Ling Wu. Hidden digital watermarks in images. *IEEE Transactions on Image Processing*, 8:58–68, 1999.

[14] Jiwu Huang, Yun Q. Shi, and Yi Shi. Embedding image watermarks in dc components. *IEEE Transactions on Circuits and Systems for Video Technology*, 10:974–979, 2000.

[15] Caiwei Lin, Lei Zhao, and Jiwen Yang. A high performance image authentication algorithm on gpu with cuda. *International Journal of Intelligent Systems and Applications(IJISA)*, 3, 2011.

[16] Mahsa Manoochehri, Hossein Pourghassem, and Ghazanfar Shahgholian. A novel synthetic image watermarking algorithm based on Discrete Wavelet Transform and Fourier-Mellin Transform. In *International Conference on Communication Software and Networks*, 2011.

[17] R.K. Megalingam, M.M. Nair, R. Srikumar, V.K. Balasubramanian, and V.S.V. Sarma. Performance comparison of novel, robust spatial domain digital image watermarking with the conventional frequency domain watermarking techniques. In *Signal Acquisition and Processing, 2010. ICSAP '10. International Conference on*, pages 349–353, Feb 2010.

[18] Han min Tsai and Long wen Chang. A High Secure Reversible Visible Watermarking Scheme. In *International Conference on Multimedia Computing and Systems/International Conference on Multimedia and Expo*, pages 2106–2109, 2007.

[19] Saraju P. Mohanty, K. R. Ramakrishnan, and Mohan S. Kankanhalli. A DCT Domain Visible Watermarking Technique for Images. In *International Conference on Multimedia Computing and Systems/International Conference on Multimedia and Expo*, pages 1029–1032, 2000.

[20] Anton Obukhov. Discrete Cosine Transform for 8x8 Blocks with CUDA. 2008.

[21] Shelby Pereira and Thierry Pun. Robust template matching for affine resistant image watermarks. *IEEE Transactions on Image Processing*, 9:1123–1129, 2000.

[22] Fabien A.P. Petitcolas. Watermarking schemes evaluation. *Signal Processing Magazine, IEEE*, 17(5):58–64, Sep 2000.

[23] FabienA.P. Petitcolas, RossJ. Anderson, and MarkusG. Kuhn. Attacks on copyright marking systems. In *Information Hiding*, volume 1525 of *Lecture Notes in Computer Science*, pages 218–238. Springer Berlin Heidelberg, 1998.

[24] A. D. Poularikas. The transforms and applications handbook. 1996.

[25] Chen Qing, Su Xiang-fang, and Wang Yan-ping. Transparent image watermarking in the Wavelet Domain. *Wuhan University Journal of Natural Sciences*, 5:457–460, 2000.

[26] Chin-Shiuh Shieh, Hsiang-Cheh Huang, Feng-Hsing Wang, and Jeng-Shyang Pan. Genetic watermarking based on transform-domain techniques. *Pattern Recognition*, 37:555–565, 2004.

[27] L.F. Turner. Digital data security system, September 21 1989. WO Patent App. PCT/GB1989/000,293.

[28] P.L.V. Vihari and M. Mishra. Image authentication algorithm on gpu. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 874–878, May 2012.

[29] S. Voloshynovskiy, S. Pereira, T. Pun, J.J. Eggers, and J.K. Su. Attacks on digital watermarks: classification, estimation based attacks, and benchmarks. *Communications Magazine, IEEE*, 39(8):118–126, Aug 2001.

[30] Fu-Hao Yeh, Greg C. Lee, and Y. T. Lin. Removable Visible Watermarking in JPEG Compression Domain. In *Asia-Pacific Services Computing Conference*, pages 1328–1331, 2008.

[31] Jiang Yong Zheng, Dong Hong Ling, Jiang Zeng Liang, and Miao Jin. A dct-based digital watermarking algorithm for image. In *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on*, pages 1217–1220, Aug 2012.