Tesis Doctoral: Cálculo en Tiempo Real de Identificadores Robustos para Objetos Multimedia Mediante una Arquitectura Paralela CPU-GPU

Dra. Miranda Natalia
Director: Dr. Chavez Edgar, Codirectora: Dra. Piccoli Fabiana
San Luis, Argentina

1. Introducción

El surgimiento de los datos multimedia ha innovado el concepto de las bases de datos. Éstas se han visto siempre como repositorios de información, tanto textual como numérica, regidas por un conjunto de normas que le aportan semántica. Ahora al considerar los objetos multimedia, se necesita hacer un cambio radical en esta concepción a fin de poder representarlos y recuperarlos. Un objeto multimedia puede ser un texto, un video, una imágen, un sonido, entre otros.

Por ser representados en forma diferente, los objetos multimedia no pueden ser tratados de la misma manera que los objetos en una base de datos tradicional. Su representación implica un tratamiento distinto, se debe extraer las características particulares y generales de cada uno de ellos para poder recuperarlos por similitud o proximidad. En el caso de las bases de datos multimediales no es posible hacer búsquedas exactas, la información no siempre se almacena de la misma manera; dos imágenes de la misma escena no necesariamente son iguales, desde el punto de vista perceptual, una persona puede no encontrar diferencias y diría que ambas imágenes son idénticas, pero si se las compara bit a bit se podría determinar que ambas imágenes son totalmente diferentes.

Para poder tratar a los objetos multimedia, en particular a las señales de audio, con la misma eficiencia con las que se tratan los objetos de texto, es necesario obtener una representación de ellos, la cual sea estable y persistente a las posibles degradaciones naturales que pueden sufrir. Esta representación se la conoce como característica de la señal o huella digital de la señal.

Una huella digital es un mecanismo capaz de identificar de manera precisa y única a un objeto o persona. La huella digital humana, es un ejemplo de ello, es el certificado de autenticidad de las personas de manera única e inconfundible.

En el caso de las señales de audio, una huella digital de audio (AFP) es un fragmento compacto de bajo nivel basado en el contenido de la señal. La huella digital proporciona la capacidad de identificar las señales de audio de una manera rápida y confiable [4, 12].

Las AFPs son tecnologías usadas como materia prima de software por un gran número de aplicaciones de importancia económica. Algunas de ellas son: monitoreo broadcast [23]; etiquetado automático de audio [12, 24] y detección de duplicados [3], entre otras.

Ahora si la necesidad es recuperar señales de audio o algún otro objeto multimedia a partir de grandes repositorios, es necesario contar con un mecanismo eficaz de recuperación de datos. Esto se puede llevar a cabo mediante el uso del modelo de espacio métrico.

El modelo de espacio métrico es un paradigma capaz de modelizar los problemas de búsqueda por similitud [6, 20, 25]. Un espacio métrico está compuesto de un conjunto de objetos (un subconjunto de éste se denomina base de datos) y una función de distancia especificada entre los objetos. La función de distancia cumple algunas propiedades establecidas y permite determinar qué objetos son próximos o similares unos de otros. Las bases de datos métricas permiten el almacenamiento de objetos de un espacio métrico (por ejemplo: documentos, strings, imágenes, secuencias de ADN, entre otros) y la realización de consultas por similitud sobre ellos de manera eficiente. En este tipo de base de datos, las consultas de mayor interés son: la búsqueda por rango y los k vecinos más cercanos (k-NN) [6]. Otras consultas a realizar en un espacio métrico son: la búsqueda de los k vecinos más cercanos para todos los objetos de la base de datos (all-k-NN) y la unión por similitud (*join*).

Para poder responder a las consultas en un espacio métrico existen diferentes métodos, en forma general se los puede clasificar en dos grandes grupos: los métodos basados en fuerza bruta y los basados en índices. En el primer caso, el más fácil de resolver, se caracteriza por examinar la base de datos (BD) completa calculando la distancia entre el objeto de consulta y todos los objetos de la BD, evaluar las distancias obtenidas y devolver el

resultado. Si se resuelve usando una CPU en forma secuencial, este método posee un alto costo computacional debido al número de evaluaciones de distancia a realizar. Los métodos de indexación, en cambio, reducen el número de cálculos de consulta, evitando tratar toda la BD. Estos métodos con índices requieren previamente una etapa de preprocesamiento de la BD para establecer el índice. En base al índice calculado, estos métodos permiten acelerar las consultas pero no siempre es suficiente, principalmente cuando estamos en un ambiente de tiempo real, donde las respuestas a las consultas deben ser obtenidas en forma rápida. En consecuencia, para lograr mayor velocidad en las respuestas, otras técnicas deben ser incluidas, una de ellas es aplicar computación de alto desempeño en la solución computacional.

Las técnicas de computación de alto desempeño (HPC, sigla en inglés de *High Perfomance Computing*) han permitido acelerar el tiempo de aquellas aplicaciones secuenciales con un alto costo computacional. Tanto la extracción de características de una señal de audio, como la recuperación de objetos en una BD métrica son un ejemplo de este tipo de aplicaciones, es por ello que pensar en soluciones donde se aplican técnicas de HPC es posible. Además, dadas las características de ambas aplicaciones y de las unidades de procesamiento gráfico GPU (sigla en inglés de *Graphics Processing Unit*), aplicar técnicas de HPC usando GPU en las soluciones es una alternativa válida.

Las GPUs son dispositivos masivamente paralelos, procesadores many-core, los cuales permiten ejecutar múltiples unidades de proceso: threads, aplicando un paralelismo de gránulo fino. Tienen características muy diferentes a otras arquitecturas paralelas, como por ejemplo la asignación flexible de los recursos locales (memoria o registros) para los threads, jerarquía de memoria, entre otras.

El modelo completo, denominado *GPU computing* o GPGPU (sigla derivada del inglés *General-Purpose Computing on Graphics Processing Units*), consiste en el uso de un sistema integrado por al menos una CPU y una GPU para acelerar las operaciones de cálculo científico de propósito general. La CPU junto con la GPU constituye una potente combinación: la CPU está formada por varios núcleos optimizados para el procesamiento de datos en serie, mientras que la GPU consta de millares de núcleos más pequeños y eficientes diseñados para el procesamiento en paralelo [7, 18, 14].

Existen varias herramientas para programar en una GPU, una de ellas es CUDA (*Compute Unified Device Architecture*). CUDA permite a los desarrolladores crear componentes aplicando un procesamiento masivo de datos en paralelo. Un programa en CUDA está especificado en lenguaje C/C++ extendido con un conjunto de instrucciones, las cuales permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad [10, 14, 15, 18, 21]. Un programa completo CUDA tiene diferentes fases, las cuales se pueden ejecutar ya sea en la CPU o la GPU. Cuando la fase tiene bajo o nulo paralelismo, su ejecución se realiza en la CPU, en cambio si la fase presenta un paralelismo masivo, se implementa y ejecuta en la GPU.

El trabajo de tesis [17] estuvo dedicado a dar soluciones de alto rendimiento utilizando una arquitectura many-core como la GPU a los problemas de extracción de características de una señal de audio para obtener la huella digital de la misma y su respectiva recuperación en una BD. Ambos problemas, los cuales pertenecen a un sistema de huella digital robusto, tienen un alto costo computacional si los mismos se calculan en la CPU, por lo tanto haciendo uso de los beneficios de la GPU para acelerar este tipo de aplicaciones, las mismas pueden ser obtenidas en tiempo real de forma rápida y eficiente.

En la próxima sección se describe el sistema integral de huella digital de la señal. En la sección 3 se presentan las AFPs desarrolladas en GPU, en la sección 4 se exponen los algoritmos en GPU sobre espacios métricos y por último se muestran las conclusiones obtenidas en el trabajo de tesis.

2. Sistema Integral de Huella Digital

En un sistema integral de huella digital identificamos dos grandes y costosos subproblemas: la extracción de características de una señal de audio (huella digital) para su identificación y la recuperación de objetos multimedia (audio) en una base de datos métrica, con el fin de aplicarla en la recuperación de señales de audio.

La indexación e identificación de señales de audio son dos disciplinas que han sido muy estudiadas en los últimos años, especialmente la última. La identificación de audio consiste en la habilidad de agrupar las señales de audio con la misma naturaleza perceptual, por esta razón se requiere una representación del objeto, la cual debe ser estable y persistente a diferentes degradaciones naturales. Esta representación se denomina huella digital de audio: AFP (AudioFingerPrint). La obtención de la AFP foma parte de un sistema de huella digital.

Un sistema de huella digital, básicamente consiste de dos partes fundamentales: la extracción de características de la señal, AFP, y la aplicación de un algoritmo de búsqueda eficiente para encontrar aquellas AFP similares a una AFP de consulta, en una BD.

Hay varios requerimientos prácticos a cumplir por un sistema de huella digital de audio exitoso, ellos son: capacidad de identificar una señal a pesar de presentar severas degradaciones del audio; rapidez en la identificación de la señal; eficiencia computacional en el cálculo de las huellas digitales y en la búsqueda en una BD de

la señal o de una similar y escabilidad.

Todas estas razones y la naturaleza de cada uno de los subproblemas hacen que una solución computacional aplicando técnicas de GPGPU sea una opción válida. En las siguientes subsecciones se describen las subtareas del sistema que contribuyeron a ser un desafío en el trabajo de tesis.

2.1. Huella Digital de Audio: Extracción de Características

En el diseño de las AFPs, existe cierta tensión entre dos objetivos contrapuestos: robustez y rapidez. Por un lado una característica robusta de la señal implica una representación densa del audio, y por otro lado una AFP densa implica más ciclos de computadora para obtener la representación. Por esta razón sería importante encontrar un punto medio entre estos dos objetivos para que la perfomance del sistema no se vea afectada.

Como la entropía es la información de contenido en una secuencia que el cerebro percibe, esto nos sirvió de motivación para usar la entropía como característica perceptual en sistemas de identificación de audio. En [1, 3, 2] se proponen métodos eficaces, usando la entropía de Shannon [22], para identificar sin error secuencias de audio sujetas a degradaciones severas. Las AFPs propuesta por los autores son TES (*Time-Domain Entropy Signature*) y MBSES (*Multi-Band Spectral Entropy Signature*).

Ambas dividen la señal en segmentos cortos de igual tamaño denominados frames, los cuales son normalizados (la señal estéreo se convierte a monoaural) y se aplica una superposición del 50 % para hacer la AFP robusta a cambios en el volumen y asegurar una baja variación de las características extraídas. Pero difieren en el cálculo de la huella.

La huella obtenida en TES, como se presentó en [2], es compacta, fácil de calcular y es robusta a degradaciones específicas de filtrado, escala y compresión con pérdida. Ésta se obtiene usando el signo de la derivada para construir un string binario donde cada bit esta determinado por el valor de entropía que se obtuvo de cada frame. Este bit tiene el valor 1 si la entropía del frame es mayor a 0, caso contrario el bit posee el valor 0.

Por el contrario, la huella MBSES está basada en las distintas bandas espectrales de entropía, las cuales están determinadas según la escala de Bark [11, 26]. La idea central de MBSES consiste en almacenar para cada banda un indicador si la entropía espectral es creciente o no en el frame actual. El bit correspondiente a la banda b y el frame n está determinado usando las entropías de los frames n y n 1 para la banda b. Solamente 3 bytes (24 bits) son necesarios para cada frame de la señal de audio. Por lo tanto la MBSES de una señal de audio es una matriz binaria donde cada fila representa la AFP binaria de un frame.

Una vez obtenida la AFP se debe encontrar un método eficaz para indexarla y luego poder recuperarla en una BD. En la próxima subsección se describen los métodos para poder llevar a cabo esta tarea.

2.2. Recuperación de Objetos Multimedia: Espacios Métricos

La indexación y recuperación de objetos multimedia también ha captado mucho el interés de los investigadores en los últimos años. Como se mencionó antes, los objetos multimedia no pueden ser tratados del mismo modo que los objetos en una BD tradicional, la representación de los objetos multimedia es totalmente diferente. Por ejemplo dos señales de audio de una misma canción pueden resultar idénticas al oído humano pero computacionalmente son totalmente distintas, pueden no coincidir en todos los bytes. Los objetos multimedia son representados por un conjunto de características relevantes y los podemos recuperar por similitud.

El modelo adecuado para los problemas de búsqueda por similitud es el modelo de espacio métrico [6, 20, 25]. Las bases de datos métricas permiten almacenar objetos de un espacio métrico y ejecutar consultas por similitud sobre ellos de forma eficiente. La organización y obtención de los objetos multimedia se puede realizar mediante la aplicación de algún método, ya sea utilizando fuerza bruta (*scan secuencial*) o mediante la construcción y aplicación de un índice. La selección de uno u otro método está determinado por la clase de objeto y el tipo de aplicación para lo cual se requiere.

Cualquiera sea el método elegido, fuerza bruta o índice, el costo computacional tanto en tiempo como espacio es elevado, debido a las estructuras de datos a administrar y el tiempo de respuesta para una consulta dada. Muchas veces es necesario pensar en aplicar otras técnicas de programación como es la computación de alto desempeño.

3. Huella Digital de Audio en GPU

Mediante la huella digital de audio, AFP, podemos identificar unívocamente a una señal, aunque ésta haya sufrido modificaciones. El proceso computacional para obtener la AFP basada en la entropía de la señal es costoso, por ello resulta necesario emplear paradigmas de computación de alto desempeño como la GPU para acelerarlo.

En esta tesis [17] se desarrollaron las dos AFPs basadas en la entropía: TES_{GPU} y $MBSES_{GPU}$, las cuales se explican a continuación:

3.1. Cálculo de la AFP $MBSES_{GPU}$

El proceso en GPU de $MBSES_{GPU}$ posee tres etapas principales como se muestra en la figura 1, todas ellas aplicadas en secuencia sobre cada frame de la señal. Como puede observarse, el primer paso para la obtención de $MBSES_{GPU}$ es dividir la señal de audio en frames de longitud fija, en este caso los frames son de 16 KB equivalente a 370 ms de duración. Esta longitud es adecuada para el cálculo de la entropía [1]. Todos los frames se superponen un 50 % para asegurar una lenta variación de las características extraídas.

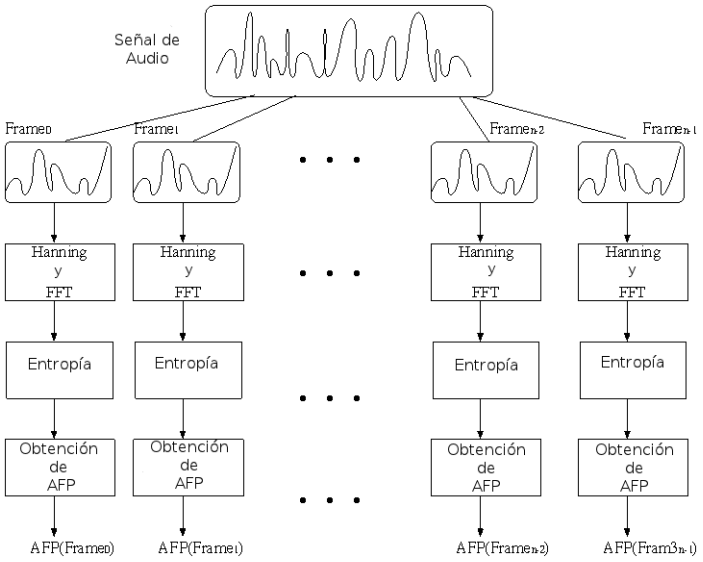

Figura 1: Proceso Paralelo para la obtención de la AFP $MBSES_{GPU}$

El problema ahora se aplica independientemente a cada frame, esta característica hace del problema particularmente adecuado para el procesamiento paralelo masivo.

Una vez dividido en frames, cada uno de ellos es asignado a un grupo de unidades de procesamiento, las cuales aplican secuencialmente las etapas: *Hanning y FFT*, *Entropía* y *Obtención AFP*. A continuación se explican brevemente cada etapa involucrada:

1. *Hanning y FFT*: En esta etapa se calcula la FFT por cada frame de audio, transformándolo desde el dominio del tiempo al dominio de la frecuencia. Previo al cálculo de la FFT se aplica por cada frame la ventana de Hanning. Esto es necesario para reducir los efectos de borde en los extremos del frame, así se ubica al frame en el centro de la ventana. Luego se computa la FFT, cuyo diseño se basó en el algoritmo original de Cooley y Tukey [8]. La muestra se divide a la mitad usando el teorema de Danielson Lanczos [9], obteniendo dos sublistas. Este proceso se repite recursivamente o de forma iterativa hasta que el problema es trivial (el vector sobre el que se aplica la FFT tiene sólo dos componentes). En la solución presente, el algoritmo es iterativo (Cuando fue desarrollado, CUDA no permitía recursión).

   La salida de esta etapa es un vector de números complejos. Las siguientes etapas trabajan sobre ambas componentes, el vector de la parte real y el de la parte imaginaria.

2. *Entropía*: Determinar la entropía implica varias tareas o fases, éstas son: *Traducción a Valores Discretos*, *Histograma Final de la Banda* y *Entropía de la Banda*. Como la entrada a esta etapa es una secuencia de valores complejos, todas las fases se aplican a dos vectores, el vector de las componentes reales y el de las componentes imaginarias. La figura 2 muestra la etapa de entropía y todas sus fases.

   En la subfase *Traducción a Valores Discretos*, Los valores continuos de cada frame son convertidos a valores discretos, necesarios para calcular el histograma. Esto implica obtener el intervalo al que pertenecen todos los valores del frame, dividirlo en tantas partes como valores distintos sean considerados en el histograma y, finalmente, asociar cada elemento del frame a la partición correspondiente. En la subfase *Histograma Final de la Banda* se calcula el histograma del frame así como los histogramas de cada banda. A partir de estos histogramas, del frame y de cada banda, se puede obtener el histograma complemento. La salida de

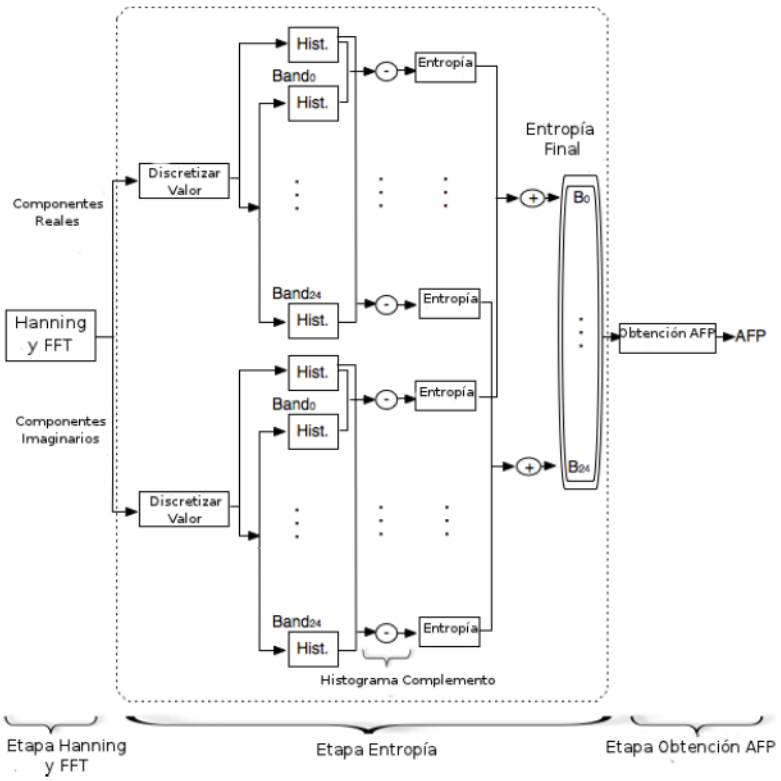

Figura 2: Etapa Entropía para cada Frame.

esta fase son los histogramas complemento de la parte real e imaginaria de cada banda. Estos histogramas son la entrada de la etapa *Entropía de la Banda*. En esta fase, cada componente x_i del histograma complemento representa la frecuencia del i-ésimo elemento ($f(x_i)$), el cual es necesario para obtener la probabilidad según la fórmula de Laplace $\frac{f(x_i)}{N'}$, donde N' es el total de elementos del frame menos la cantidad de elementos en la banda actual. Cada unidad de procesamiento calcula una probabilidad, $p(x_i)$ y la parte interna del cálculo de entropía $p(x_i) \in ln(p(x_i))$. Finalmente los resultados de cada unidad de procesamiento se suman. Esta operación es una reducción en paralelo por suma. Los datos de salida de la etapa de la entropía son N vectores (uno por frame) de 24 componentes. Cada componente se corresponde con la suma de las entropías real e imaginaria de cada banda.

3. *Obtención AFP*: Una vez calculados los valores de entropía para cada frame, la AFP es computada. Por cada frame se obtiene una AFP de 24 digitos binarios, donde cada dígito se corresponde con un valor binario obtenido a partir de la sustración entre la entropía del frame anterior con la entropía del frame actual de una banda.

Al finalizar este proceso, se obtiene la $MBSES_{GPU}$ de la señal de audio, la cual estará compuesta de un conjunto to a AFPs, uno por cada frame, de 24 componentes cada una.

3.2. Cálculo de la AFP TES_{GPU}

La figura 3 muestra el proceso paralelo para la obtención de la AFP de una señal de audio. Por sus características, el problema es altamente paralelizable y adecuado para ser resuelto en la GPU.

Al inicio del proceso, la señal de audio es normalizada, la señal estéreo es convertida a monoaural promediando ambos canales. Posteriormente, es dividida en frames de longitud fija, en este caso la longitud es igual a 64 KB, teniendo en cuenta una superposición del 50 %. Cada frame es procesado en paralelo por un grupo de unidades de procesamiento. El resultado final es una AFP, TES_{GPU}, la cual es una secuencia de N bits, si la señal de audio posee N frames.

Para obtener la TES_{GPU}, el procesamiento en paralelo de cada frame implica resolver en secuencia las siguientes tareas: *Cálculo del valor mínimo y máximo*, *Cálculo de Entropía*, y *Obtención de la huella digital binaria*. Estas tareas se explican a continuación:

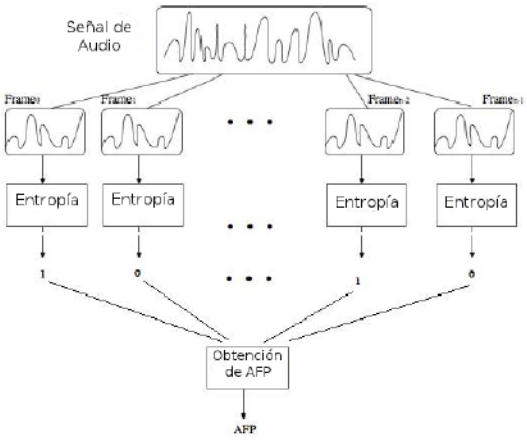

Figura 3: Proceso Paralelo para la obtención de la AFP TES

1. *Cálculo del valor mínimo y máximo*: Para obtener esta AFP debemos computar el histograma total de la señal, la cual está dividida en N frames trabajados en paralelo. Calcular el histograma implica convertir los valores continuos de la señal a valores discretos. Esto implica obtener el intervalo al que pertenecen todos los valores de la señal, dividirlos en tantas partes como valores distintos serán considerados en el histograma y, finalmente, asociar cada elemento de la señal a la partición correspondiente. Para llevar a cabo cada una de estas tareas es necesario primero determinar los límites del intervalo (valor máximo y mínimo) para la señal completa, luego dividir el intervalo en 256 subintervalos (el proceso considera un histograma de 256 valores distintos), y finalmente instanciar cada elemento de la señal a uno de los 256 intervalos.

   Con ambos valores (valor máximo y mínimo), cada unidad de procesamiento realizará la discretización de sus respectivos valores, instanciándolos con uno de los 256 posibles valores discretos.

2. *Cálculo de Entropía*: Su cómputo se basa en la entropía de Shannon mediante histograma. Cada grupo de unidades de procesamiento trabaja sobre un frame distinto. Primero los valores continuos del frame son convertidos a valores discretos teniendo en cuenta los mínimos y máximos globales determinados en la etapa anterior, necesarios para calcular el histograma.

   Cada componente x_i del histograma representa la frecuencia del i-ésimo elemento, el cual es necesario para obtener la probabilidad del elemento x_i según la fórmula de Laplace ($p(x_i) = \frac{f(x_i)}{N}$ donde $f(x_i)$ es la frecuencia de x_i, y N es el tamaño de la señal). Cada unidad de procesamiento es responsable del cómputo de la parte interna de la fórmula de entropía ($p(x_i) \in Ln(p(x_i))$). Los resultados son sumados en paralelo.

3. *Obtención de la huella digital binaria*: Para obtener la AFP final se utiliza un único grupo de unidades de procesamiento, donde la cantidad de unidades depende del número de frames. Cada unidad de procesamiento aporta el valor binario de la huella de un frame.

Todas estas etapas son aplicadas en secuencia, sin poder alterar el orden. El resultado final es una secuencia de bits, los cuales constituyen la AFP de la señal de audio.

Uno de los resultados empíricos que se obtuvieron fue la aceleración (speedup) de $MBSES_{GPU}$ y TES_{GPU} para las señales de audio cuyo tamaños oscilan entre 6 MB y 218 MB. Se pudo observar que $MBSES_{GPU}$ mostró un buen desempeño, el cual es más alto cuando los tamaños de las señales son más grandes. Por ejemplo la señal de 218 MB obtiene una aceleración aproximada a 270X mientras que para la señal de 6 MB es de 19X. Esto obedece a la influencia de los costos de las transferencias entre CPU y GPU. En todos los casos podemos observar buenos desempeños, ya sea en GPUs con arquitecturas antiguas o con menor cantidad de recursos.

En el caso de TES_{GPU}, su comportamiento fue distinto al de $MBSES_{GPU}$. Su desempeño está influenciado por las características del problema (fases de cómputo con menor costo) en la aceleración alcanzada, la cual osciló entre 30X y 40X dependiendo de la GPU. Además es bueno destacar que el tamaño de la señal no incide en el desempeño, éste se mantiene cercano al constante, independiente de las dimensiones de la entrada.

En el trabajo de tesis se mostraron también otros resultados como el througput (cantidad de señales procesadas por segundo) y los tiempos de ejecución de ambas AFPs cuando varía el tamaño del frame (entre 4 KB y 256 KB). Con respecto al througput se pudo observar, en ambas AFPs, que sus comportamientos son dependientes del tamaño de la señal, a medida que las dimensiones del audio se incrementan, menor cantidad

de señales son procesadas por segundo. En cuanto a la variación del tamaño del frame, se mostró que éste no influye significativamente en el tiempo de ejecución de ambas huellas, razón por la cual se tomaron los estudios realizados por [1] estableciendo los tamaños en 16KB para $MBSES_{GPU}$ y 64KB para TES_{GPU}.

4. Espacios Métricos en GPU

El algoritmo propuesto en esta tesis [17], denominado *Top-K*, aplica el método de fuerza bruta para la resolución del problema de k-NN usando GPU. Por su naturaleza, este método es altamente paralelizable y por lo tanto es adecuado para resolverse en este tipo de arquitectura.

Siguiendo la filosofía del método de fuerza bruta, el proceso se inicia con el cálculo de las distancias entre el objeto de consulta y todos los objetos de la BD. Una vez realizado este paso, los objetos con las distancias más pequeñas tienen que ser identificados, lo cual se considera un desafío a superar. La idea es seleccionar los objetos en forma independiente, evitando interrelaciones entre ellos (*crosstalk*).

Como los espacios de memoria de la CPU y la GPU son disjuntos, previo al inicio de la búsqueda debe transferirse la BD y las consultas a la GPU. Al finalizar el proceso la transferencia de los resultados es a la inversa, desde la GPU a la CPU.

Se han desarrollado y definido 3 versiones para la resolución del problema, las cuales se diferencian en el proceso de determinación de los menores. Las tres alternativas propuestas las denominamos: *Top-K_AA* (Todos contra todos, *All to Al*), *Top-K_QSeP* (*Quickselect*), *Top-K_QSeH* (Híbrido: *Quickselect* + *All to All*). Todas ellas tienen una fase común, el cálculo de las distancias. La diferencia radica en la forma de obtener los k elementos más similares a la consulta. La figura 4 muestra el proceso de obtención de los k-NN en forma genérica, donde la diferencia está en la fase de *Selección_k-NN*.

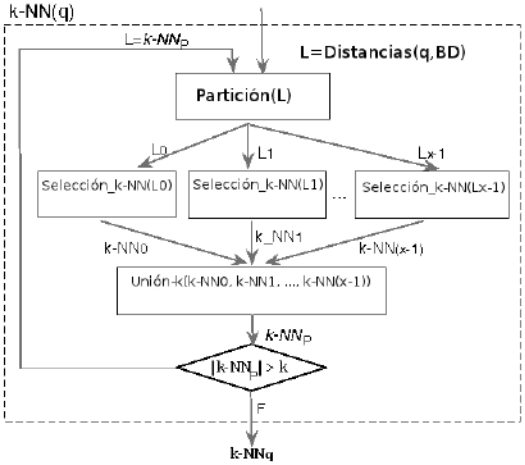

Figura 4: Proceso Genérico para obtener los k-NN de una consulta q

Como puede observarse, el proceso itera mientras el tamaño de la lista de las k-NN$_p$ locales es mayor que k, al inico la lista L está formada por todas las distancias de la consulta a cada objeto de BD. La lista es dividida, etapa de *Partición*, y asignada a un grupo de unidades de procesamiento. Cada unidad de procesamiento determina sobre qué elemento de la lista L_i debe trabajar según la información local provista por el grupo al que pertenece. La etapa *Selección_k-NN* es una unidad de cálculo, su responsabilidad es la elección de los objetos k-NN de la lista L_i, donde $\bigcup L_i = L \mid i = 0 \times\times\times x \quad 1$ y x es la cantidad de grupos trabajando, para luego guardarlos en la lista parcial k-NN_p (etapa *Unión-k*). Después de la primer iteración, L contiene sólo las distancias de los objetos k-NN de cada grupo. A partir de esta nueva lista, se continúa aplicando el proceso.

El algoritmo *Top-K* propuesto en sus tres versiones, se divide en dos etapas bien definidas. En la primera etapa, las distancias entre cada elemento de la BD y el objeto de consulta son procesadas en paralelo. Esto significa que existen n unidades de procesamiento calculando al mismo tiempo una distancia correspondiente a la distancia entre un objeto de la BD y el elemento de consulta.

La segunda etapa se encarga de la iteración mostrada en la figura 4, en la cual se han desarrollado las tres posibles soluciones. Cada una de las soluciones presenta diferentes características, particularmente en la etapa *Selección_k-NN*. Todas las alternativas toman como entrada las distancias de todos los elementos de la BD al objeto de consulta ($L =< d_0, d_1, ..., d_{n-1} >$ donde $d_i = distancia(q, o_i) \mid o_i \ \forall \ BD$). El proceso se inicia,

particionando las distancias en L_i sublistas ($i = 0 \times\times\times x \quad 1$), las cuales son trabajadas en forma independiente y en paralelo. Cada algoritmo desarrollado se caracteriza por:

1. *Top-K_AA*: En la etapa *Selección_k-NN* se eligen los k-NN elementos a partir de la lista local L_i, cada elemento se compara con todos los elementos del grupo para determinar si su distancia está entre los k más pequeños del grupo. Una vez determinados los k elementos menores de cada una de las L_i sublistas, se unifican en una única lista (*Unión-k* de la figura 4). El resultado de la unificación será la entrada de la próxima iteración.

   Este procedimiento se realiza iterativamente hasta obtener los k-NN de la consulta (iteración en la figura 4). La iteración finaliza cuando la secuencia a tratar tiene k elementos, los cuales constituyen los k-NN resultado de la consulta. Después de la primera iteración, la lista parcial tiene $k \in x$ elementos donde x es el número de grupos de trabajo.

2. *Top-K_QSeP*: En esta versión se aplica la metodología propuesta por el algoritmo *Quickselect* [13, 16], el cual encuentra los y elementos más pequeños de una secuencia no ordenada z. En la etapa *Selección_k-NN* de la figura 4 se aplica el algoritmo *Quickselect* en cada una de las L_i listas locales para elegir los k-NN de cada grupo. Este proceso se aplica iterativamente mientras que la posición del elemento distinguido, pivote, no es igual a k. En cada iteración, seleccionamos el pivote, el cual es la mediana de tres valores de la lista local (primero, último y el valor que se encuentra en la posición centro). A partir del pivote se realiza la partición de la lista local. Si la posición de pivote es igual a k, la partición con los elementos más pequeños que el pivote son los k-NN de la lista local. De lo contrario, pueden suceder dos casos posibles: la posición del pivote es mayor que k, o es menor que k. En el primer caso, se ejecuta el *QuickSelect* sobre la partición con los elementos más pequeños. En otro caso, se trabaja sólo sobre la segunda partición de la lista local, la cual tiene los elementos más grandes que el pivote. Al finalizar la iteración, el pivote se encuentra en la posición k, obteniéndose los k-NN de la consulta.

3. *Top-K_QSeH*: Este algoritmo aplica una combinación de los dos anteriores, razón por la cual se llama *Top-K_QSeH*, *H* por híbrido. En esta propuesta, en la primera iteración, mostrada en la figura 4, se aplica el algoritmo *Quickselect* a cada L_i resultante de la partición de la lista con todas las distancias. Como resultado se obtienen los k-NN de cada lista local al grupo. Luego en los pasos siguientes, se determinan los k-NN locales aplicando el algortimo *Top-K_AA* iterativamente hasta obtener los k-NN resultado de la consulta.

Los algoritmos planteados anteriormente han sido diseñados respetando la consigna inicial de formular un algoritmo simple y de bajo costo. Como puede observarse en todas las propuestas se mantiene la independencia entre los datos, de manera tal que la interacción entre las unidades de procesamiento es nula. Al finalizar la ejecución, los k-NN de la consulta quedan almacenados en memoria global. Una vez que se procesaron todas las consultas, se transfieren a la CPU los resultados.

Además de resolver consultas por k-NN como se explicó anteriormente, se han implementado las consultas por rango y los *all-k*-NN. La consulta por rango, tiene la particularidad de que su solución es altamente paralelizable, y su implementación en la GPU es trivial. Una vez transferida la BD a la memoria de la GPU, cada thread calcula la distancia entre el objeto de consulta q y un objeto o_i de la BD. Si la distancia es menor al radio requerido, $r \; \forall \; \mathbb{R}$, entonces o_i forma parte del resultado. Este proceso se realiza por cada objeto de consulta q.

Por el contrario, la consulta por all-k-NN se la considera como una generalidad de una consulta por los k-NN. En este caso, la BD y el conjunto de consultas coinciden, todos los objetos de BD son considerados objetos de consultas y objetos de la BD con los cuales se debe comparar cada consulta. Todos los resultados son transferidos conjuntamente a la CPU, por ello se debe considerar cómo es la representación y el orden para recuperar un resultado particular.

En sistemas de gran escala no es suficiente acelerar una consulta a la vez, también es necesario aprovechar las capacidades de la GPU para responder en paralelo varias consultas. Con el objeto de cumplir con este fin, la GPU recibe un conjunto de consultas y las resuelve todas a la vez. Cada consulta, en paralelo, aplica el proceso de obtención de los k-NN.

El número de recursos necesarios para resolver q consultas en paralelo, es igual a la cantidad de recursos requeridos para calcular una consulta multiplicado por el número de consultas resueltas en paralelo. La cantidad de consultas a resolver, q , en paralelo se debe determinar considerando los recursos disponibles de la GPU, principalmente su memoria. Para llevar a cabo este proceso se deben gestionar cuidadosamente los bloques y sus threads, esto es posible y fácil de lograr estableciendo una relación entre los identificadores del thread, del bloque, de la consulta y del elemento de la BD.

La resolución de múltiples consultas k-NN en paralelo se aplica a la solución de los *all-k*-NN. Si se responden q consultas en paralelo, podemos calcular los all-k-NN en X iteraciones, donde $X = \frac{n}{|q|}$. La misma filosofía se puede aplicar para resolver múltiples consultas por rango.

Para realizar el análisis de los resultados en la tesis, se comparó la solución paralela con el algoritmo secuencial SAT^+ [5, 19]. Las BD que se tuvieron en cuenta fueron string y vectores. Las GPUs sobre las que se ejecutaron los experimentos fueron de distintas arquitecturas: G80 y GF100. En las consultas por rango, k-NN y *all-k*-NN se obtuvieron aceleraciones significantes entre los 40X y los 700X. Además para estos tipos de consultas se analizó el rendimiento donde se pudo observar que el algoritmo secuencial podía resolver de 0 a 4 consultas por segundo mientras que el paralelo en sus tres versiones podía resolver entre 100 y 9000 consultas por segundo dependiendo del tipo de BD que se estaba tratando.

Además se chequeó la escalibilidad del algoritmo *Top-K* a medida que la BD incrementaba de tamaño. El resultado empírico que se observó fue el aumento de la aceleración a medida que crecía la BD. Esto es esperable porque cuando la cantidad de objetos es pequeña se sub-utilizan los recursos de la GPU y las transferencias tienen mayor influencia en el tiempo de procesamiento. También se obtuvieron buenos resultados cuando la BD es mucho más grande que el tamaño de memoria de la GPU.

Del análisis de los resultados, vemos que el algoritmo *Top-K* en sus tres versiones, es simple y aprovecha al máximo las características de la GPU: varios bloques de threads, accesos coalescentes a memoria global, uso de memoria compartida, independencia del trabajo en paralelo y mínima interrelación entre la CPU y la GPU; obtiene excelentes resultados a través del paralelismo intra-consulta e inter-consulta.

5. Conclusiones

El enfoque adoptado en esta tesis doctoral se basó fundamentalmente en el uso de técnicas de computación de alto desempeño considerando la GPU como una computadora paralela de propósito general, GPGPU. Los problemas abordados son los involucrados en un sistema de huella digital: la extracción de características de los objetos multimedia (en particular las señales de audio) y la identificación de los mismos en grandes repositorios de información, bases de datos métricas.

Antes de iniciar el desarrollo de este trabajo, se realizó una amplia exploración del estado del arte en los tres aspectos fundamentales involucrados en esta tesis. Primero se consideró el uso de la GPU como arquitectura paralela para resolver problemas generales. Luego nos dedicamos al estudio de la huella digital de audio y el modelo de espacios métricos. Finalmente, se relacionaron estos temas mediante la evaluación de la aplicación de técnicas paralelas en GPU en las soluciones de ambos problemas.

En este trabajo de tesis se desarrollaron las huellas digitales $MBSES$ y TES propuestas en [1], basadas en la entropía de la señal. Ambas fueron desarrolladas usando la GPU como medio para acelerar el proceso de extracción de características. Los excelentes resultados de ambos han mostrado que el empleo de estas técnicas es una buena desición. En el caso de $MBSES$, la AFP es obtenida a partir del dominio de la frecuencia, implicando un sinnúmero de etapas cada una con un elevado costo computacional (cálculo de FFT, obtención de histogramas, división de la señal en bandas, entre otras). Debido a esto, el tamaño de la señal influye en el tiempo de ejecución del proceso, en consecuencia y mediante el empleo de la GPU hemos logrado acelerar el proceso en un factor de 200x respecto al proceso secuencial cuando la señal es grande (Superior a los 110MB). En el caso de TES, como la AFP se obtiene en el dominio del tiempo, el proceso es más simple. Los resultados alcanzados son casi constante, independiente del tamaño de la señal. Además las ganancias se ven afectadas por las transferencias entre la CPU y GPU. Aún así, y a pesar de esta dificultad, el proceso en la GPU es más rápido que el respectivo secuencial, obteniendo aceleraciones entre 30x y 40x. A partir de los resultados observados y de su análisis es posible afirmar que el uso de la GPU en este tipo de aplicaciones es relevante principalmente si se quiere aplicar en ambientes de tiempo real.

Respecto a la recuperación de datos, el modelo de espacios métricos es adecuado para aplicar en la resolución de consultas por señales de audio cuando éstas son representadas por sus características relevantes. Las bases de datos métricas permiten almacenar las representaciones de objetos multimedia para luego poder recuperarlos mediante búsquedas por similitud. Dichas búsquedas tienen un alto costo computacional, encontrar los objetos similares a un objeto dado en grandes repositorios de datos implica realizar muchas evaluaciones de distancia. Si bien se han propuesto diversos algoritmos secuenciales para reducir los costos tanto de tiempo como de espacio, esto no siempre es suficiente. Por ello pensar en la aplicación de técnicas de computación de alto desempeño, especialmente el uso de GPU, es una alternativa viable.

En esta tesis se propuso un algoritmo, llamado $Top \quad K$ y tres versiones distintas del mismo. $Top \quad K$ es una versión paralela del método de fuerza bruta, el cual es altamente paralelizable y adecuado para resolverse aplicando el modelo de programación de CUDA, permitiendo tomar ventaja de los beneficios de la GPU. Del análisis de los resultados experimentales realizados podemos establecer que sin importar la versión de

Top K utilizada, los resultados obtenidos son muy buenos. Se puede ver claramente que sin importar la naturaleza de la base de datos y las características de cada objeto de la misma, se logró acelerar el tiempo de resolución de las búsquedas, aún cuando la GPU es limitada tanto en espacio de memoria como capacidad de computación y generación de la arquitectura. Además observamos la influencia del tamaño de la base de datos en la aceleración, esto nos permitió determinar que no sólo se obtienen buenos resultados en los casos de bases de datos con tamaños adecuados para alojarse en la memoria principal de la CPU o de la GPU, sino también para aquellas cuya cardinalidad es mucha mayor a las respectivas capacidades de memoria. En el caso de la CPU, se resuelve el problema utilizando la memoria secundaria, hecho que no es posible en la GPU donde el problema se resolvió haciendo múltiples transferencias entre la CPU y la GPU. Aunque las transferencias afectan el desempeño de la aplicación, mediante los resultados experimentales hemos mostrado que siguen siendo eficientes las soluciones en la GPU, hasta cuando se trata de bases de datos de gran tamaño.

En ambas líneas de desarrollos hemos obtenido acelerelaciones muy buenas, algunas de ellas sobre el límite lineal. Ésto se debe al aprovechamiento de todas las capacidades propias de la GPU, las cuales permitieron estos logros, entre ellas destacamos el uso de las memorias de acceso más rápido como la memoria shared, de constante y de registro, como así también el acceso controlado a la memoria global.

Por todo lo expuesto se pudo concluir que el objetivo general de la tesis fue cumplido, los resultados alcanzados nos permiten validar el uso de la GPU para dar solución a los problemas derivados de un sistema de huella digital robusta y eficiente. Con los desarrollos realizados se logró reducir significativamente el tiempo de las soluciones, permitiendo transferir estos logros a la solución de otros problemas.

Referencias

[1] A. Camarena-Ibarrola. *Análisis Digital de la Señal de Voz - PhD thesis Borrador*. PhD thesis, Universidad Michoacana de San Nicolás de Hidalgo, México, Agosto 2007.

[2] A. Camarena-Ibarrola and E. Chávez. A robust entropy-based audio-fingerprint. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 1729–1732, 2006.

[3] A. Camarena-Ibarrola and E. Chávez. Real time tracking of musical performances. In *Proceedings of the 9th Mexican international conference on Artificial intelligence conference on Advances in soft computing: Part II*, MICAI'10, pages 138–148, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] P. Cano, E. Batlle, T. Kalker, and J. Haitsma. A review of audio fingerprinting. *J. VLSI Signal Process. Syst.*, 41(3):271–284, november 2005.

[5] E. Chávez, V. Ludueña, N. Reyes, and P. Roggero. Reaching near neighbors with far and random proxies. *Proceedings of the 8th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2011)*, pages 574–581, 2011.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[7] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Applications of GPU Computing Series. Morgan Kaufmann, 2013.

[8] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematical Computing*, 19:297–301, 1965.

[9] G.C. Danielson and C. Lanczos. Some improvements in practical fourier analysis and their application to x-ray scattering from liquids. *J. Franklin Institute*, 233:365–380, 435–452, 1942.

[10] R. Farber. *CUDA Application Design and Development*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[11] H. Fujisaki. *Recent Research Towards Advanced Man-Machine Interface Through Spoken Language*. Elsevier Science, 1996.

[12] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *In International Symposium on Music Information Retrieval, ISMIR*, 2002.

[13] C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, jul 1961.

[14] W. Hwu. *GPU Computing Gems Emerald Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[15] D. B. Kirk and W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[16] M. Kuba. On quickselect, partial sorting and multiple quickselect. *Inf. Process. Lett.*, 99(5):181–186, sep 2006.

[17] Miranda Natalia. *Cálculo en Tiempo Real de Identificadores Robustos para Objetos Multimedia Mediante una Arquitectura Paralela CPU-GPU*. PhD thesis, Universidad Nacional de San Luis, Abril 2014. Sobresaliente.

[18] NVIDIA. Nvidia cuda c programming guide. vers. 5.5, July 2013.

[19] N. Reyes. índices dinámicos para espacios métricos de alta dimensionalidad. Master's thesis, Universidad Nacional de San Luis, Diciembre 2002.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[21] E. Sanders, J.and Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[22] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[23] S. Shin, O. Kim, J. Kim, and J. Choil. A robust audio watermarking algorithm using pitch scaling. In *Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on*, volume 2, pages 701–704, 2002.

[24] A.L. Wang. An industrial-strength audio search algorithm. In *Proceedings of the 4 th International Conference on Music Information Retrieval*, 2003.

[25] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.

[26] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer Series in Information Sciences, 1990.