ICT-UNPA-115-2015
ISSN: 1852-4516
Aprobado por Resolución N° 0579/15-R-UNPA

PROCESAMIENTO DE BÚSQUEDAS POR SIMILITUD. TECNOLOGÍAS DE PARALELIZACIÓN E INDEXACIÓN.[1]

DOS SANTOS, EderI. SOFIA, Albert A. OsirisI. URIBE PAREDES, RobertoII.
esantos@unpa.edu.ar, sistemasuarg@gmail.com, roberto.uribeparedes@gmail.com

I. Unidad Académica Río Gallegos – Universidad Nacional de la Patagonia Austral
Lisandro La Torre 1070 – 0054-2966-442313 – Río Gallegos – Santa Cruz – Argentina
II. Departamento de Ingeniería en Computación, Universidad de Magallanes, Chile.

RESUMEN

La búsqueda por similitud consiste en recuperar todos aquellos objetos dentro de una base de datos que sean parecidos o relevantes a una determinada consulta. Actualmente es un tema de gran interés para la comunidad científica debido a sus múltiples campos de aplicación, como la búsqueda de palabras e imágenes en la *World Wide Web*, reconocimiento de patrones, detección de plagio, bases de datos multimedia, entre otros. La búsqueda por similitud o en proximidad se modela matemáticamente a través de un espacio métrico, en el cual los objetos son representados como una caja negra donde la única información disponible es la distancia de este objeto a los otros.

En general, el cálculo de la función de distancia es costoso y los sistemas de búsqueda operan a una gran tasa de consultas por unidad de tiempo. A fin de optimizar este procesamiento se han desarrollado numerosas estructuras métricas, que funcionan como índices y realizan un preprocesamiento de los datos a fin de disminuir las evaluaciones de distancia al momento de la búsqueda.

Por otro lado, la necesidad de procesar grandes volúmenes de datos hace poco factible la utilización de una estructura en aplicaciones reales si ésta no considera la utilización de entornos de procesamiento paralelo. Existen una serie de tecnologías para realizar implementaciones de procesamiento paralelo. Se incluyen entre las más vigentes las tecnologías basadas en arquitecturas multi-CPU (multi-core) y GPU / multi-GPU, que son interesantes debido a las altas prestaciones y los bajos costes involucrados.

En el presente Informe Científico-Técnico se aborda la búsqueda por similitud y la implementación de estructuras métricas sobre entornos paralelos. Se presenta el estado del arte en los temas relacionados a búsqueda por similitud con estructuras métricas y tecnologías de paralelización. También se proponen análisis comparativos sobre experimentos que buscan identificar el comportamiento de un conjunto de espacios métricos y estructuras métricas seleccionados sobre plataformas de procesamiento basadas en *multicore* y GPU.

Palabras claves: Búsquedas por similitud, espacios métricos, estructuras métricas, procesamiento paralelo, GPUs.

[1] Este trabajo fue parcialmente financiado por el proyecto de investigación UNPA-UARG 29/A274-1.

1 INTRODUCCIÓN

La búsqueda de objetos similares sobre un gran conjunto de datos se ha convertido en una línea de investigación de gran interés. Aplicaciones de estas técnicas pueden ser encontradas en reconocimiento de voz e imagen, en problemas de minería de datos, detección de plagios y muchas otras.

En general, se utilizan diversas estructuras de datos con vistas a mejorar la eficiencia en términos de los cálculos de distancia, comparados con la búsqueda secuencial dentro de la base de datos (conocido como algoritmo fuerza bruta). Por otro lado, se busca reducir los costos en términos de tiempo mediante el procesamiento en paralelo de grandes volúmenes de datos [**GKKG03**].

Existen una serie de tecnologías para realizar implementaciones de procesamiento paralelo. Se incluyen entre las más vigentes las tecnologías basadas en arquitecturas multi-CPU, GPU y multi-GPU. En este sentido, la aplicación de estas nuevas tecnologías sobre búsquedas por similitud en espacios métricos [**UVASC11**] [**UACS12**] permiten un alto nivel de paralelismo a un muy bajo coste.

La paralelización de estructuras métricas sobre GPU y en clusters de multicores son áreas de investigación poco exploradas. Existe una serie de características poco comunes en estas estructuras que hacen difícil su implementación directa para aplicaciones reales. La primera tiene relación con capacidades dinámicas, ya que la mayoría de estas estructuras deben ser reconstruidas si existen nuevos objetos que indexar o si hubiese que eliminar objetos de la base de datos. Por otro lado, las estructuras métricas por lo general no consideran la jerarquía de memoria; por ello, resulta relevante considerar este aspecto para poder obtener un mayor rendimiento y eficiencia ante la posibilidad de utilizar nuevas tecnologías, como es el caso de las GPU que poseen distintos niveles de memoria dentro de su sistema. Otra problemática está relacionada con el procesamiento masivo de los datos. En la paralelización de estructuras métricas deben considerarse, entre otros aspectos, la distribución adecuada de la base de datos sobre el entorno, la paralelización de los métodos de búsqueda, la eficiencia en la comunicación entre los procesadores; etc. Finalmente, se ha dado a conocer escenarios en donde la indización puede resultar ineficiente frente a la búsqueda realizada por fuerza bruta. En algunas oportunidades, la indización se comporta peor que fuerza bruta; en ello influyen el rango de búsqueda (puede ser muy grande), la base de datos y el tiempo de procesamiento de una consulta [**UVASC11**].

De acuerdo a los antecedentes disponibles y los trabajos previos realizados, se ha planteado la implementación de distintas soluciones sobre espacios de datos disímiles, tales como histogramas de colores, bases de datos de palabras, vectores de coordenadas, vectores de gauss, imágenes NASA y otros, con el fin de identificar en dichos espacios la forma óptima de distribución de los datos, tanto de la base de datos como de las estructuras, en ambientes de GPU y/o Multi-GPU, en las memorias administradas por las CPUs y las memorias dedicadas de las GPU. Finalmente, se pretende a futuro evaluar la escalabilidad de las implementaciones de acuerdo al tamaño de la Base de Datos, y consecuentemente el almacenamiento y procesamiento de Bases de Datos y consultas en memoria secundaria.

2 MARCO TEÓRICO

2.1 Búsquedas por Similitud en Espacios Métricos

La búsqueda por similitud consiste en la búsqueda de objetos más similares a través de una búsqueda por rango o de vecinos más cercanos en un espacio métrico. Se puede definir un espacio métrico como un conjunto *X* con una función de distancia $d:X2 \rightarrow R$, tal que $\forall x, y, z \in X$, se deben cumplir las propiedades de: positividad ($d(x,y)>=0$ *y* $d(x,y)=0$ *ssi* $x=y$), simetría ($d(x,y)=d(y,x)$) y desigualdad triangular ($d(x,y)+d(y,z)>=d(x,z)$).

Sobre un espacio métrico (X,d) y un conjunto de datos finito $Y \subset X$, se puede realizar una serie de consultas. La consulta básica es la **consulta por rango**. Sea un objeto $x \in X$, y un rango $r \in R$. La consulta de rango alrededor de *x* con rango *r* es el conjunto de puntos $y \in Y$, tal que $d(x,y)<= r$. Un segundo tipo de consulta, que puede construirse usando la consulta por rango, es **los *k* vecinos más cercanos**. Sea un objeto $x \in X$, y un entero *k*. Los *k* vecinos más cercanos a *x* son un subconjunto *A* de objetos de *Y*, donde $|A|=k$ y no existe un objeto $y \in A$ tal que $d(y,x)$ sea menor a la distancia de algún objeto de *A* a *x*.

El objetivo de los algoritmos de búsqueda es minimizar la cantidad de evaluaciones de distancia realizadas para resolver la consulta. Los métodos para buscar en espacios métricos se basan principalmente en dividir el espacio empleando la distancia a uno o más objetos seleccionados. El no trabajar con las características particulares de cada aplicación tiene la ventaja de ser más general, pues los algoritmos funcionan con cualquier tipo de objeto [**CNBM01**].

2.2 Estructuras Métricas para Búsqueda

Existen distintas estructuras para buscar en espacios métricos, las cuales pueden ocupar funciones discretas o continuas de distancia. Su objetivo es reducir las evaluaciones de distancias durante la búsqueda, y con ello disminuir el tiempo de procesamiento.

Algunas de las estructuras basan la búsqueda en pivotes y otras en clustering. En el primer caso se seleccionan pivotes del conjunto de datos y se precalculan las distancias entre los elementos y los pivotes. Cuando se realiza una consulta, se calcula la distancia de la consulta a los pivotes y se usa la desigualdad triangular para descartar candidatos. Algunas estructuras de este tipo son: LAESA [**MOV94**], FQT y sus variantes [**CPM94**], Spaghettis y sus variantes [**CMB99**] [**NN97**], FQA [**FQA01**], SSS-Index [**SODA93**] Yianilos P. *Data structures and algorithms for nearest neighbor search in general metric spaces*. In Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93), pages 311–321, 1993.

[**SOFSEM07**] y otros [**ICCSDE12**].

Los algoritmos basados en *clustering* dividen el espacio en áreas, donde cada área tiene un *centro*. Se almacena alguna información sobre el área que permita descartar toda el área mediante sólo comparar la consulta con su centro. Los algoritmos de *clustering* son los mejores para espacios de alta dimensión, que es el problema más difícil en la práctica. Algunos ejemplos de estructuras son GNAT [**Brin95**], M-Tree [**CPZ97**], SAT [**Nav02**], Slim-Tree [**TTSF00**] , EGNAT [**UN09**] y otros [**CNBM01**].

Existen dos criterios para delimitar las áreas en las estructuras basadas en *clustering*: *hiperplanos* y *radio cobertor* (*covering radius*). El primero divide el espacio en particiones

de *Voronoi* y determina el hiperplano al cual pertenece la consulta según a qué centro corresponde. El criterio de radio cobertor divide el espacio en esferas que pueden intersectarse y una consulta puede pertenecer a más de una esfera.

Un *diagrama de Voronoi* está definido como la subdivisión del plano en *n* áreas, una por cada centro c_i del conjunto $\{c_1, c_2, ..., c_n\}$, tal que *q* al área c_i sí y sólo sí la distancia euclidiana $d(q, c_i) < d(q, c_j)$ para cada c_j, con $j \neq i$.

En los métodos basados en pivotes, se selecciona un conjunto de pivotes y se precalculan las distancias entre los pivotes y todos los elementos de la base de datos. Los pivotes sirven para filtrar objetos en una consulta utilizando la desigualdad triangular, sin medir realmente la distancia entre el objeto consulta y los objetos descartados.

1. Sea $\{p_1, p_2, ..., p_k\} \in X$ un conjunto de pivotes. Para cada elemento *y* de la base de datos Y, se almacena su distancia a los *k* pivotes ($d(x, p_1)$, ..., $d(x, p_k)$). Dada una consulta *q* y un rango *r*, se calcula su distancia a los *k* pivotes ($d(q, p_1)$, ..., $d(q, pk)$).
2. Si para algún pivote p_i se cumple que $| d(q, p_i) – d(x, p_i) | > r$, entonces por desigualdad triangular se tiene que $d(q, x) > r$; por lo tanto, no es necesario evaluar explícitamente $d(x,q)$. Todos los objetos que no se puedan descartar por esta regla deben ser comparados directamente con la consulta *q*.

Las estructuras de tipo árbol utilizan esta técnica en forma indirecta. El árbol se va construyendo tomando el nodo raíz como pivote. Posteriormente el espacio se divide de acuerdo a la distancia de los objetos al pivote. Cada subárbol se construye recursivamente tomando un nuevo pivote de los elementos del subespacio. Las diferencias radican principalmente en la forma y tamaño de los espacios. La búsqueda realiza un *backtrack* sobre el árbol y utiliza la desigualdad triangular para minimizar los subárboles. Algunas estructuras que utilizan esta técnica son el *BKT* y sus variantes [**CNBM01**]. Otros algoritmos, de tipo arreglo, hacen una implementación directa de este concepto, y se diferencian básicamente en su estructura extra para reducir el costo de CPU para encontrar los puntos candidatos, pero no en la cantidad de evaluaciones de distancia. Ejemplos de éstos son: *AESA* [**Vid86**], *LAESA* [**MOV94**], *Spaghettis* y sus variantes [**CMB99**] [**NN97**].

Una estructura métrica genérica (Generic Metric Structure: GMS) puede considerarse como una tabla de distancias entre los pivotes y todos los elementos de la base de datos. Es decir, cada celda almacena la distancia $d(yi, pj)$ siendo yi un elemento de la base de datos. Durante la búsqueda por rango sobre esta estructura dada una consulta *q* y un rango *r*, el algoritmo sería:

1. Para cada consulta *q*, se calcula la distancia entre *q* y todos los pivotes *p1, ... , pk*. Con esto se obtienen *k* intervalos de la forma *[a1, b1], ..., [ak, bk]*, donde $ai = d(pi, q) − r$ y $bi = d(pi, q) + r$.
2. Los objetos, representados en la estructura por sus distancias a los pivotes, son candidatos a la consulta q si todas sus distancias están dentro de todos los intervalos.
3. Para cada candidato y, se calcula la distancia $d(q, y)$, y si $d(q, y) <= r$, entonces el objeto y es solución a la consulta q.

La Figura 1 representa una estructura de datos métrica genérica (GMS) construida con 4 pivotes. Para cada consulta q con distancias a los pivotes d(q,pi) = {8, 7, 4, 6} y un rango de búsqueda r = 2, define los siguientes intervalos {(6, 10), (5, 9), (2, 6), (4, 8)}. Las celdas marcadas en gris oscuro son aquellas que están dentro del intervalo de búsqueda. Las celdas tachadas con líneas son los objetos candidatos (2, 13 y 15), que serán evaluados directamente con la consulta.

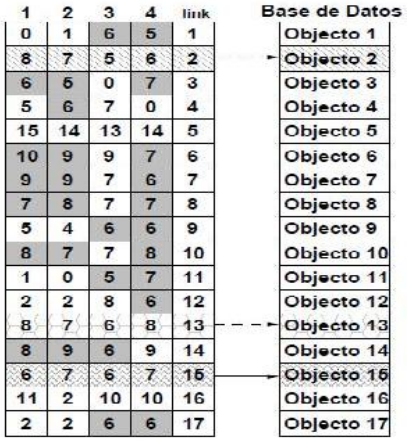


Figura 1. Búsqueda sobre una estructura métrica genérica (GMS).

2.3 Dimensionalidad Intrínseca

Uno de los principales obstáculos para el diseño de técnicas de búsqueda eficientes en espacios métricos es la *dimensionalidad* de éstos. Las técnicas tradicionales de indexación en su mayoría no son eficientes en espacios de alta dimensión. La búsqueda por similitud en espacios métricos se torna intrínsecamente más difícil mientras mayor sea la dimensión intrínseca del espacio [**SODA93**][**Brin95**][**WSP97**][**CNBM01**]; este hecho es conocido como la *maldición de la dimensionalidad* [**BELL57**].

El concepto de dimensionalidad está relacionado a la distribución de probabilidad de las distancias entre los elementos: espacios de alta dimensión poseen una distribución cuyo histograma de distancias es más concentrado y posee una media mayor. Esto dificulta el trabajo a realizar por el algoritmo de búsqueda empleado. A medida que crece la dimensionalidad intrínseca de un espacio, la media del histograma de distancias crece, y su varianza se reduce.

La dimensión intrínseca de un espacio métrico se define cuantitativamente en [**CNBM01**] [**CN00**] [**CN01**] como $3^2 / 2\pm^2$, donde 3 y $\pm$ son respectivamente la media y la varianza del histograma mencionado. Esta definición es coherente con la noción de dimensión en un espacio vectorial con coordenadas uniformemente distribuidas.

Analíticamente la razón de la maldición de la dimensionalidad consiste en que los espacios dimensionales más altos tienen una distribución de probabilidad de distancias entre elementos cuyo histograma es más concentrado y con una media grande. Esto hace que el trabajo de cualquier algoritmo de búsqueda sea más dificultoso (lo que se discute en [**SODA93**][**Brin95**][**WSP97**][**CNBM01**]). A medida que crece la dimensionalidad intrínseca de un espacio, la media del histograma crece y su varianza se reduce. Así, la búsqueda en un

espacio métrico A es más difícil que en otro espacio B cuando el histograma de distancia de A es más concentrado que el histograma de B.

2.4 Plataformas Paralelas de Memoria Compartida

Actualmente, existe una gran variedad de plataformas paralelas sobre las cuales se pueden implementar estructuras métricas. En este contexto, muchos estudios se han enfocado inicialmente a plataformas de memoria distribuida usando bibliotecas de alto nivel como MPI [**MPI94**] o PVM [**PVM94**], y memoria compartida usando el lenguaje o directivas de OpenMP [**GKKG03**]. En este sentido, [**GBMB10**] propone una estrategia para organizar el procesamiento de consultas sobre espacios métricos en nodos multi-core. Finalmente, trabajos recientes han tratado nuevas tecnologías de memoria compartida, entre las cuales se destacan plataformas basadas en GPU, a ejemplo de [**ICCSDE12**][**ISCSCT09**][**GDB08**][**DEXA12**].

A continuación, se presentan las plataformas de memoria compartida utilizadas en los experimentos llevados a cabo para la realización de este Informe.

2.4.1 OpenMP

OpenMP [**OMP07**] es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Consiste en una especificación de un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno que pueden ser usadas para paralelismo de alto nivel en programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución *fork-join*[2]. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows.

OpenMP es por tanto un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Esencialmente, se utilizan los distintos *cores* de los procesadores utilizados actualmente para procesar los hilos establecidos por la aplicación.

Una aplicación construida con un modelo híbrido de programación paralela se puede ejecutar en un *cluster* de computadoras utilizando OpenMP y otras tecnologías como MPI y GPU, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

2.4.2 Unidades de Procesamiento Gráfico

Actualmente las unidades de procesamiento gráfico (GPU o *Graphical Process Unit*) disponen de un alto número de *cores* con un alto ancho de banda. Este tipo de dispositivos permite aumentar la capacidad de procesamiento respecto de las CPU [**FM07**]. Una tendencia denominada *Computación de Propósito General sobre GPU* o GPGPU ha orientado la utilización de GPU sobre nuevos tipos de aplicaciones. Uno de los objetivos de las GPU es

[2] Paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join).

aumentar la capacidad de procesamiento explotando el paralelismo a nivel de datos para problemas paralelos o que se puedan paralelizar.

Actualmente, las empresas fabricantes de dispositivos GPU han propuesto nuevos lenguajes o extensiones para los lenguajes de alto nivel más utilizados. Un ejemplo es la propuesta de NVIDIA con [**CUDA**], la cual consiste en una plataforma software que permite utilizar las GPU para aplicaciones de propósito general, con la capacidad de manejar un gran número de hilos.

2.4.3 Modelo de Programación CUDA

El modelo de Programación CUDA de NVIDIA considera a la GPU como un dispositivo capaz de ejecutar un alto número de hilos en paralelo. Este modelo hace una distinción entre el código ejecutado en la CPU (host) con su propia DRAM (*host memory*) y el ejecutado en GPU (*device*) sobre su DRAM (*device memory*). CUDA incluye herramientas de desarrollo software basado en C/C++, librerías de funciones, y un mecanismo de abstracción que oculta los detalles hardware al desarrollador, todo ello formando una API propia (Application Programming Interface).

En CUDA los hilos son organizados en bloques del mismo tamaño y los cálculos son distribuidos en una malla o *grid* de bloques. Estos hilos ejecutan el código de la GPU, denominado *kernel*. Un *kernel* CUDA ejecuta un código secuencial en un gran número de hilos en paralelo. Los hilos en un bloque pueden trabajar juntos eficientemente, intercambiando datos localmente en una *memoria compartida* y sincronizando a baja latencia en la ejecución mediante barreras de sincronización. Por el contrario, los hilos ubicados en diferentes bloques pueden compartir datos, solamente accediendo a la memoria global de la GPU o *device memory*, que es una memoria de más alta latencia.

2.5 GPU y espacios métricos

Existe una importante cantidad de índices o estructuras métricas para búsquedas sobre espacios métricos. Sin embargo, existe una serie de características poco comunes en estas estructuras que hacen difícil su implementación directa para aplicaciones reales. La primera tiene relación con capacidades dinámicas, la mayoría de estas estructuras deben ser reconstruidas si existen nuevos objetos que indexar o si hubiese que eliminar objetos de la base de datos.

Otra característica, aún menos frecuente, está relacionada con estructuras que permitan la manipulación eficiente de los datos en memoria secundaria, donde deben ser considerados parámetros de costo adicional como son el número de accesos a disco y el tamaño del índice entre otros. En si, dichas estructuras han sido diseñadas como prototipos, por lo que su utilización en aplicaciones reales no ha sido probada, mas aún considerando que en aplicaciones reales los objetos son de gran tamaño y/o la base de datos es considerablemente grande, donde es posible pensar que sólo un bajo porcentaje de datos podría entrar en memoria principal. En general, las estructuras métricas no consideran la jerarquía de memoria, por ello, resulta relevante considerar este aspecto para poder obtener un mayor

rendimiento y eficiencia ante la posibilidad de utilizar nuevas tecnologías, como es el caso de las GPU que poseen distintos niveles de memoria dentro de su sistema.

Otra característica es la relación con el procesamiento masivo de datos. Resulta poco factible la utilización de una estructura en aplicaciones reales si ésta no considera su implementación en entornos paralelos. En la paralelización de estructuras métricas deben considerarse, entre otros aspectos, la distribución adecuada de la base de datos sobre el entorno, por ejemplo en un cluster de PCs; la paralelización de los métodos de búsqueda; eficiencia en la comunicación entre los procesadores; etc.

Desde el punto de vista de las GPU, esencialmente se han abordado soluciones para las consultas kNN sin utilizar estructuras de datos, es decir, se utilizan para paralelizar búsquedas exhaustivas (fuerza bruta) [**ISCSCT09**] [**GDB08**] [**BELL57**] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover

[**BGTPM11**]. Todos estos trabajos abordan el problema desde el supuesto de que los elementos de la base de datos poseen una gran dimensión (con una gran dimensión intrínseca), lo cual implica poder descartar muy pocos objetos usando desigualdad triangular, y en donde la indización es ineficiente frente a la búsqueda secuencial.

La paralelización de estructuras métricas sobre GPU y en clusters de multicores son áreas de investigación poco exploradas. En este contexto, se conocen dos grupos de investigación que están desarrollando trabajados en torno a estructuras métricas sobre plataformas basadas en GPU. El primero de estos grupos, de la Universidad Complutense de Madrid, ha enfocado sus estudios a dos estructura métricas, la Lista de Clusters y SSS-Index, y ha presentado diversos trabajos en propuestas para kNN y consultas por rango [**BELL57**] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover

[**BGTPM11**] [**ISPA12**]. En el segundo grupo, de la Universidad de Castilla La Mancha, participa el profesor Roberto Uribe-Paredes, las líneas de investigación abordadas por este grupo están orientadas a desarrollar y potenciar, sobre un ambiente híbrido, la estructura genérica presentada anteriormente [**UVASC11**] [**ICCSDE12**] [**DEXA12**] [**JP12**].

2.6 Trabajo Relacionado

Algunos estudios se han enfocado a la paralelización de diferentes estructuras sobre plataformas de memoria distribuida y a la distribución de los datos y el balance de carga usando las librerías MPI o BSP, que usualmente trabajan sobre plataformas del tipo *cluster*. Estos estudios igualmente han implicado el estudio de la distribución de los datos y el balance de carga **[MT98] [ADT02] [GMR09]**.

Son más escasos los trabajos orientados hacia aplicaciones de memoria compartida. En general, mientras algunos estudios analizan la distribución de consultas y datos sobre nodos *multicores*, otras investigaciones proponen combinar procesamiento de consultas multihilo totalmente asíncronas con masivamente síncronas en base al nivel de tráfico de consultas [**GBMB10**].

Al día de hoy la mayoría del trabajo realizado se lleva a cabo sobre plataformas clásicas de memoria distribuida y compartida, existiendo pocos estudios en torno a plataformas basadas en GPU. Algunas soluciones generalistas que utilizan GPU sólo abordan el problema de consultas *kNN* sin utilizar estructuras de datos. En general, las GPU básicamente se utilizan para paralelizar búsquedas exhaustivas por lo que no se utilizan estructuras métricas. En general, en los trabajos anteriores la paralelización es aplicada en dos etapas. La primera

consiste en construir la matriz de distancias, y la segunda en el proceso de ordenamiento para obtener los resultados [**KZ09**] [**GDB08**] [**BDHK06**].

En [**BELL57**] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover

[**BGTPM11**] se presenta una variante de lo expresado anteriormente. Este trabajo compara dos estrategias, donde la primera de ellas se realiza al estilo de los trabajos anteriores. Sin embargo, la segunda estrategia, llamada Heap Based Reduction, propone resolver una consulta por cada bloque. Después de haber calculado todas las distancias para una consulta (exhaustivamente), envía en cada lanzamiento de kernel un solo bloque, manteniendo un heap por cada hilo del bloque. Cada heap de tamaño k se utiliza para almacenar los k vecinos más cercanos a partir de las distancias entre los elementos de la base de datos y la consulta.

En [**BELL57**] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover

[**BGTPM11**] y [**UVASC11**] se utilizan estructuras métricas sobre una GPU y comparan sus resultados con versiones secuenciales. En [**UVASC11**] se utiliza una estructura métrica y se comparan los resultados obtenidos para la búsqueda por rango con versiones secuencial y multicore-CPU, mostrando una mejora notable al usar este nuevo tipo de plataforma.

3 ANÁLISIS, DISCUSIÓN Y RESULTADOS

3.1 Casos de Estudio

Para los experimentos llevados a cabo por el grupo de investigación, se seleccionan distintos espacios métricos considerados representativos para este tipo de problemas. Con la finalidad de clasificar dichos espacios, se utiliza la función de distancia utilizada para la búsqueda por similitud. De esta manera, es posible identificar dos grupos de espacios métricos.

El primer conjunto de espacios corresponde a diccionarios de distintos idiomas. La función de distancia utilizada es la distancia de edición, que corresponde al mínimo número de inserciones, eliminaciones o sustituciones necesarias para que una palabra sea igual a otra. Los rangos de similitud aplicados en la búsqueda van de 1 a 4. A continuación, algunos ejemplos de la aplicación de la función de distancia:

d(ala, ala) = 0;
d(ala, ama) = 1;
d(ala, aula) = 1;
d(ala, alas) = 1;
d(ala, hola) = 2;
d(ala, pelo) = 3;
d(ala, oso) = 3;

El segundo grupo de espacios métricos contiene bases de datos de vectores de distintas dimensiones, los cuales representan distintos objetos; son predominantemente histogramas de colores de distintas imágenes (fotos de rostros, imágenes satelitales, diagramas), en donde

dichos histogramas son representados en la forma de vectores de distintas dimensiones. Para este segundo grupo, se ha elegido como función de distancia la distancia Euclidiana, y se usaron rangos que recuperaban el 0.01%, 0.1% y 1% de similitud desde el conjunto de datos.

Para llevar a cabo los experimentos, ambas bases de datos se dividen en dos conjuntos aleatorios; el primero contiene el 90% de los objetos de la base, y se utiliza para construir las estructuras métricas; el 10% restante se utiliza como objetos de consulta sobre el primer conjunto. Los costes de construcción de las estructuras no son considerados en los tiempos medidos. En la *Tabla 1* y *Tabla 2* se presenta un resumen de las bases de datos utilizadas en los distintos experimentos.

ESPACIO MÉTRICO	Dimensión ABSOLUTA	Elementos de BÚSQUEDA	Elementos de la BASE DE DATOS
Diagramas de Gauss	05	10000	90000
Diagramas de Gauss	10	10000	90000
Diagramas de Gauss	15	10000	90000
Diagramas de Gauss	20	10000	90000
Diagramas de Gauss	101	10000	90000
Imágenes NASA	20	11915	107241
Histogramas	112	11268	101414
Rostros	254	11915	107241

Tabla 3-1: Espacios Métricos de Vectores utilizados en los experimentos.

ESPACIO MÉTRICO	Elementos de la BASE DE DATOS	Elementos de BÚSQUEDA	Dimensión MÁXIMA
Español I	77455	8606	21
Español II	46431	5158	20
Alemán I	206396	22932	38
Inglés	62162	6907	21
Francés	124432	13825	25
Alemán II	67578	7508	33
Italiano	105192	11687	24
Noruego	77074	8563	32
Sueco	109149	12127	29
Multi-idiomas	444644	49404	50

Tabla 3-2: Espacios Métricos de Palabras utilizados en los experimentos.

Basados en las experiencias de los integrantes del grupo de investigación, los experimentos realizados tenían el propósito de determinar espacios y dimensiones donde es realmente conveniente utilizar estructuras métricas sobre plataformas basadas en GPU. Así, se espera confirmar o refutar la hipótesis de la conveniencia de utilización de fuerza bruta o a lo sumo estructuras genéricas basadas en pivotes en ambientes GPU, extendiendo las pruebas de laboratorio a un amplio conjunto de ambientes y espacios. Cabe también resaltar que se han elegido estas condiciones para la realización de las pruebas debido a que son las típicamente usadas para este tipo de experimentos.

3.2 Ambiente Experimental

Como plataforma de evaluación experimental, se ha trabajado con los espacios métricos mencionados en un entorno *multicore* y un entorno GPU, utilizando la estructura GMS basada en distintas cantidades de pivotes y el algoritmo de búsqueda por fuerza bruta.

Para cada rutina, se ejecuta la aplicación 4 veces y se obtiene un promedio. Para determinar la cantidad de repeticiones, se ha considerado un factor de variación entre los tiempos máximos y mínimos obtenidos en cada batería de experimentos, con el fin de identificar si la variación podría resultar representativa o si sería necesario generar más experimentos para cada caso. A continuación, se exhiben los máximos factores de variación detectados:

Procesamiento	Rango 1	Rango 2	Rango 3	Rango 4
GPU (Mem. compartida)	0,056751%	0,063488%	0,058146%	0,020388%
GPU (Mem. Global)	0,181196%	0,034801%	0,011006%	0,002801%
Open MP	0,095846%	0,178914%	0,043177%	0,072305%

Tabla 3-3. Máxima variación entre los resultados máximos y mínimos obtenidos en los experimientos para espacios métricos de palabras.

Los experimentos generados para cada espacio métrico consisten en un producto cartesiano de las variables descritas a continuación:

- Índices / Estructuras / Algoritmo: Fuerza Bruta y GMS basada en pivotes.
- Cantidades de Pivotes utilizadas (para GMS): 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 y 1024.
- Procesadores: 1 CPU (secuencial), 4 CPU (multi-core), 8 CPU (multi-core con hyper-threading) y GPU (utilizando memoria global y memoria compartida).

De modo de cuantificar el universo de experimentos, se presenta a continuación un resumen de las variables tenidas en cuenta y el volumen global de experimentos realizados.

Tipo de Base	Espacios Métricos	Tipos de Procesamiento	Estructuras / Algoritmos	Rangos de Búsqueda	Repeticiones por Caso	Total
Diagramas de Gauss	5	5	12	3	4	3600
Otros Vectores	3	5	12	3	4	2160
Diccionarios	10	5	12	4	4	9600
					TOTAL	**15360**

Tabla 3-4. Resumen de los experimentos realizados.

En virtud de lo mencionado anteriormente, se ha llevado a cabo un plan de experimentos con un total de 15360 experimentos distintos.

Finalmente el *hardware* utilizado corresponde a un Intel® Core™ i7-2600 CPU @3.40GHz de 4 núcleos y soporte para Hyper-Threading, 12GB de memoria principal y dos tarjetas Nvidia EVGA DDR5 de 384 cores CUDA y 1 GB de memoria global cada una. La codificación de las estructuras métricas y de los algoritmos de búsqueda se ha realizado utilizando el lenguaje C (gcc 4.3.4), ejecutadas sobre una plataforma Linux Ubuntu 12.04 LTS (Precise Pangolin); para la paralelización, se ha adoptado CUDA SDK v3.2 para el caso

de las aplicaciones GPU, y se ha utilizado la librería OpenMP para la paralelización multi-CPU.

3.3 Resultados Preliminares

A continuación, se detallan los resultados de la evaluación experimental. Cabe resaltar que los experimentos de laboratorio fueron realizados de modo que se pudiera realizar las siguientes comparaciones:

1. Eficiencia de las distintas opciones de procesamiento. Se han realizado experimentos con códigos diseñados para el procesamiento de distintos tipos:

    1. Secuencial: utilización de 1 sólo *core* en su capacidad máxima.
    2. Multi-core: utilización de los 4 *cores* en su máxima capacidad con OpenMP.
    3. Hyper-Threading(TM): utilización de los 4 *cores* en su máxima capacidad en modo Hyper-Threading con OpenMP.
    4. GPU con el procesamiento en memoria global.
    5. GPU con procesamiento distribuido en memoria compartida.

2. Estructura Genérica GMS *vs.* Algoritmo de Fuerza Bruta: por un lado, se han realizado experimentos empleando el algoritmo de Fuerza Bruta. Por otro lado, se emplearon distintas cantidades de pivotes (1, 2, 4, 8, 16, 32, 64, 128, 256, 512 y 1024) con el fin de identificar la cantidad más eficiente en cada experimento.

3. Cálculo de la Dimensionalidad Intrínseca de los Espacios Métricos y su influencia sobre la performance de los algoritmos.

Los experimentos fueron realizados sobre las distintas bases especificadas en el punto anterior. A continuación, se presenta una selección de resultados relevantes que orientan el análisis de los experimentos realizados. El análisis de los resultados obtenidos en los experimentos se basan en el estudio de los tiempos de procesamiento. Se utilizarán gráficos en los cuales los ejes de los gráficos corresponden al tiempo de procesamiento en segundos (Y) en los distintos rangos propuestos anteriormente (X).

3.3.1 Procesamiento Secuencial vs. Procesamiento Paralelo

Este primer bloque de análisis realizados contempla las distintas opciones de procesamiento (secuencial, multicore, GPU con uso de memoria global, GPU con distribución en memoria compartida), sobre el algoritmo Fuerza Bruta; ello nos permite visualizar el rendimiento del procesamiento plano de toda la base de datos sin el uso de índices o estructuras especiales. Se han utilizado como casos de estudio los experimentos sobre un espacio métrico de vectores de dimensión absoluta 112 (Histogramas) y palabras del diccionario español. Se ha adoptado esta selección inicial a fin de exhibir de un modo general el rendimiento de las aplicaciones sobre las distintas funciones de evaluación de distancia descritas en el presente documento.

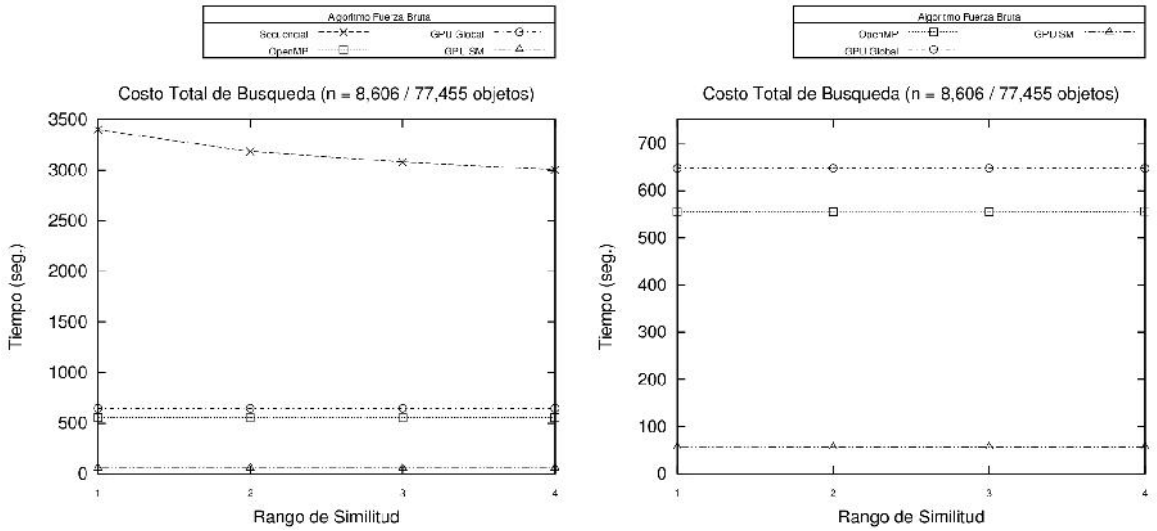


Figura 2. Tiempos de procesamiento sobre un espacio de métrico de palabras con procesamiento paralelo y secuencial.

En la figura anterior se observan dos gráficos con el tiempo de procesamiento para un espacio métrico de palabras (diccionario Español) utilizando el algoritmo de búsqueda conocido como *Fuerza Bruta*. En el gráfico a la izquierda es posible observar la optimización lograda por todas las modalidades de paralelismo comparadas con el procesamiento secuencial. El gráfico a la derecha representa un detalle del gráfico a la izquierda, donde queda de manifiesto el rendimiento de la aplicación en las distintas opciones de paralelismo. Es posible observar finalmente un grado de optimización en el rendimiento cuando se utiliza el procesamiento basado en la memoria compartida de la GPU.

Para cuantificar los resultados, se utilizan como unidad de medida los tiempos de procesamiento en segundos. Luego, se calcula la razón entre el tiempo transcurrido entre dos resultados A y B, dando origen al *speed-up* de latencia. En este primer análisis, se puede constatar una diferencia significativa entre el procesamiento secuencial y las distintas opciones de procesamiento paralelo.

A continuación, se pueden observar los tiempos de procesamiento obtenidos y los valores de *Speedup* correspondientes.

Opción	Tiempo s.	SpeedUp			
		GPU	GPUSM	MC	SEC
GPU	647,30	--	*0,09*	*0,86*	5,25
GPU SM	56,48	11,46	--	9,83	60,20
MC	555,09	1,17	*0,10*	--	6,13
SEC	3400,15	*0,19*	*0,02*	*0,16*	--

Tabla 3-5. Valores de SpeedUp sobre una búsqueda de rango 1.

La mayor diferencia constatada es con el uso de la memoria compartida (GPU SM) respecto del procesamiento secuencial, que resulta en un *Speedup* de 60.2, 56.2, 54.21 y 52.89 para los rangos 1, 2, 3 y 4, respectivamente. Se exhibe a continuación un resumen del *Speedup* obtenido, de acuerdo al siguiente esquema: en el gráfico a la izquierda, se exhiben los valores de *Speedup* obtenidos con procesamiento paralelo sobre el procesamiento secuencial; en el gráfico a la derecha, los valores de *Speedup* obtenidos desde el procesamiento con la memoria compartida de la GPU sobre las demás opciones de procesamiento (GPU con memoria global

y multicore con *Hyper-Threading*). El eje X corresponde a los rangos de búsqueda y el eje Y representa los valores de *Speedup* obtenidos.

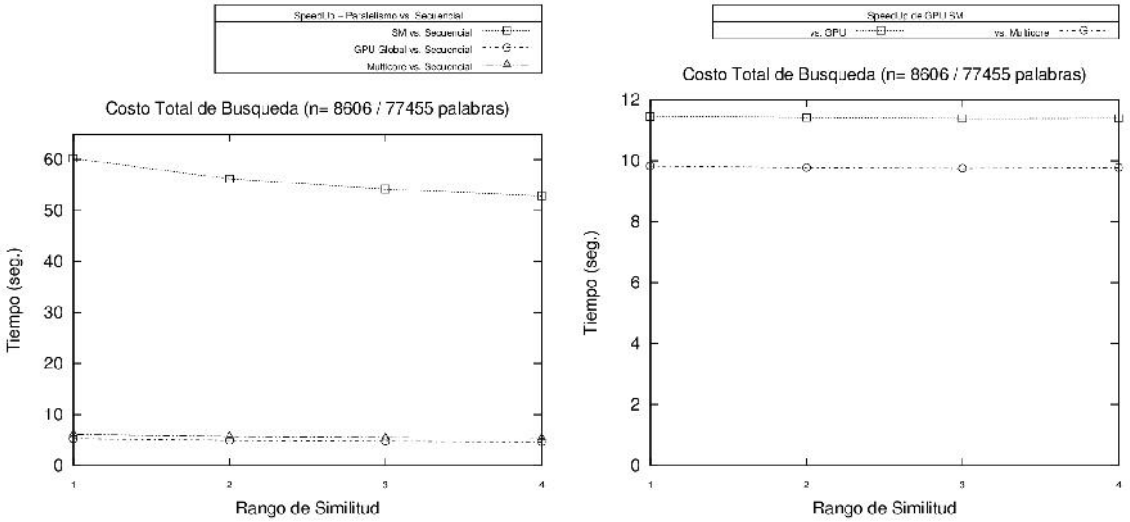


Figura 3. Speedup obtenidos paralelismo vs. secuencial (izq.) y GPU compartida vs. GPU Global y Multicore HT (der.).

Cabe resaltar que los resultados anteriores no son suficientes para corroborar la eficiencia del procesamiento con GPU, por lo que se realizaron experimentos sobre distintos espacios métricos. Se exhiben a continuación los resultados obtenidos en experimentos realizados sobre un espacio métrico de vectores; el esquema de presentación es similar al dispuesto en la Figura 2.

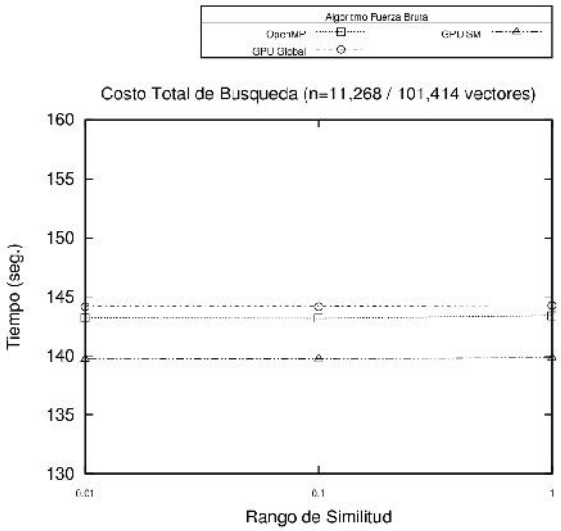


Figura 4. Tiempos de procesamiento sobre un espacio de métrico de vectores con procesamiento paralelo y secuencial.

Los gráficos nos permiten identificar que el procesamiento por GPU con memoria compartida también es más eficiente sobre un espacio métrico de vectores, aunque los resultados no

poseen una mejora tan significativa como en el caso anterior. Los coeficientes de *Speedup* obtenidos en el procesamiento de la memoria compartida en comparación con el procesamiento secuencial fueron de 3.88, 3.86 y 3.83 para los rangos de búsqueda 0.01%, 0.1% y 1%, respectivamente. En este mismo sentido, este segundo experimento presenta una mayor equidad en los tiempos de procesamiento comparado con el espacio métrico utilizado anteriormente, en lo referente al rendimiento de las distintas opciones de paralelismo. En el caso anterior, es posible apreciar valores de *speedup* cercanos a 11.4 (GPU con memoria compartida *vs.* GPU con memoria global) y 9.8 (GPU con memoria compartida *vs.* Multicore con *Hyper-Threading*). En el último ejemplo, los valores de *speedup* obtenidos fueron respectivamente de 1.03 y 1.02. La Tabla 3-6 exhibe los valores de *speedup* obtenidos en la búsqueda sobre el espacio métrico de vectores mencionado anteriormente.

Opción	Tiempo s.	SpeedUp				
		GPU	GPUSM	MC	SEC	
GPU	144,16	--	*0,97*	*0,99*	3,76	Rango 0.01
GPU SM	139,84	1,03	--	1,02	3,88	
MC	143,22	1,01	*0,98*	--	3,79	
SEC	542,26	*0,27*	*0,26*	*0,26*	--	

Opción	Tiempo s.	SpeedUp				
		GPU	GPUSM	MC	SEC	
GPU	144,17	--	*0,97*	*0,99*	3,74	Rango 0.1
GPU SM	139,77	1,03	--	1,02	3,86	
MC	143,16	1,01	*0,98*	--	3,76	
SEC	538,92	*0,27*	*0,26*	*0,27*	--	

Opción	Tiempo s.	SpeedUp				
		GPU	GPUSM	MC	SEC	
GPU	144,26	--	*0,97*	*0,99*	3,72	Rango 1
GPU SM	139,82	1,03	--	1,03	3,83	
MC	143,39	1,01	*0,98*	--	3,74	
SEC	535,99	*0,27*	*0,26*	*0,27*	--	

Tabla 3-6. Speedup obtenidos en la búsqueda sobre un espacio métrico de vectores.

De este modo, si bien las muestras anteriores permiten dar cuenta de la eficiencia de la memoria compartida del GPU sin el uso de estructuras métricas, se realizaron más experimentos con el fin de corroborar esta afirmación; se extendió el experimento a un conjunto más amplio de espacios métricos. Es posible observar a continuación en los gráficos de la Figura 5 el rendimiento obtenido en las plataformas de memoria compartida para experimentos sobre distintos espacios métricos. Se descarta la presentación de los resultados del procesamiento secuencial. Si bien la razón de *speedup* presenta variaciones entre las tres tecnologías de paralelismo adoptadas en los experimentos, en todos los casos el procesamiento sobre memoria compartida (GPU SM) de la GPU se ha mostrado más eficiente que las otras opciones de paralelismo utilizadas.

Finalmente, otro aspecto que se puede observar está relacionado con la influencia del rango de búsqueda sobre el procesamiento: en el algoritmo de Fuerza Bruta, el rango de similitud prácticamente no ejerce influencia significativa en el procesamiento, ya que no se utilizan índices para reducir las evaluaciones de distancia.

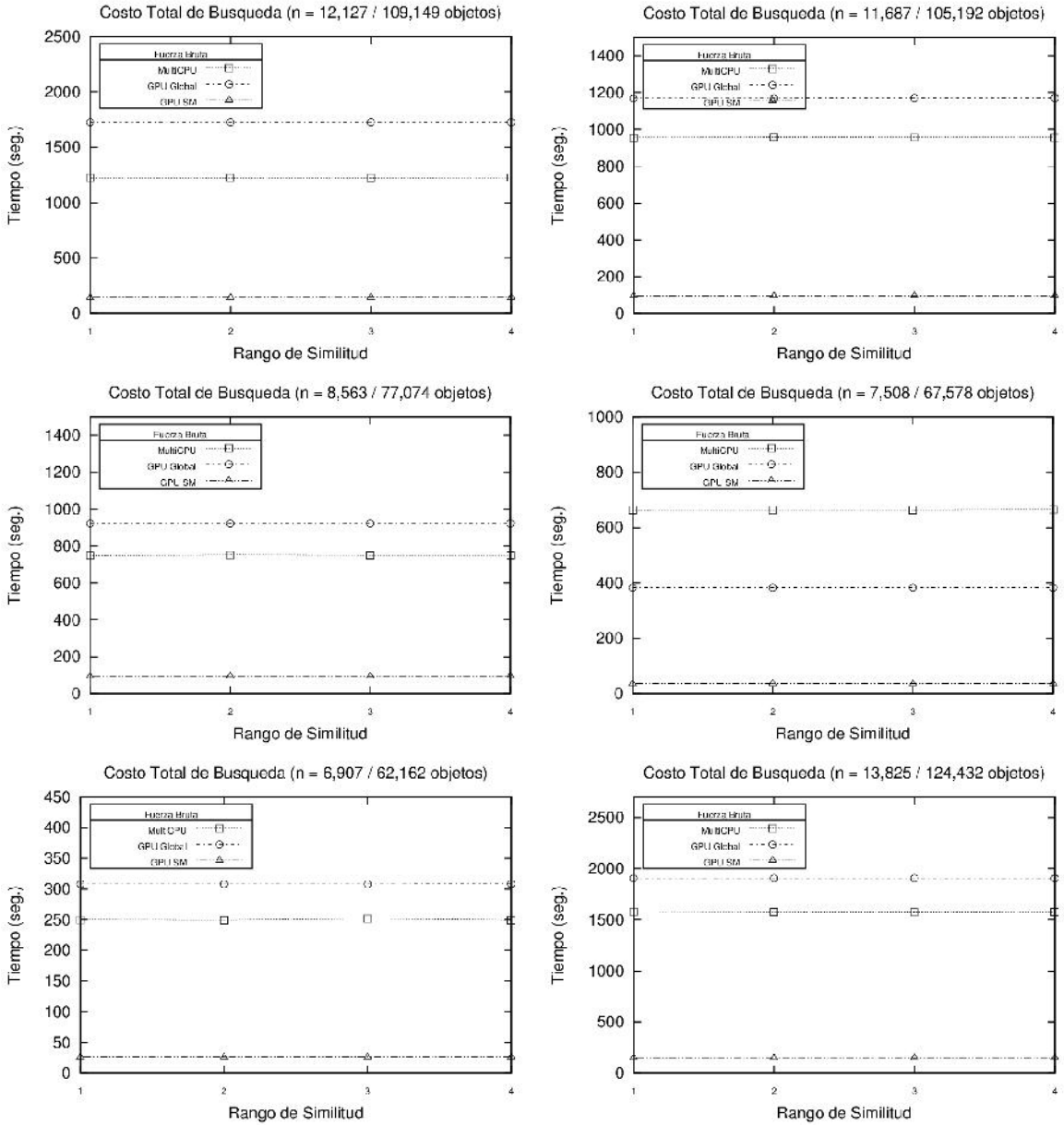


Figura 5. Tiempos de procesamiento en distintos espacios métricos de palabras utilizando el algoritmo Fuerza Bruta.

3.3.2 Estructuras Métricas vs. Fuerza Bruta

El análisis anterior nos ha permitido identificar que, en un entorno de procesamiento por Fuerza Bruta – es decir, sin el uso de estructuras métricas –, el procesamiento en paralelo utilizando la memoria compartida de la GPU presenta un rendimiento sustancialmente superior al procesamiento multi-core y GPU únicamente con el uso de la memoria global.

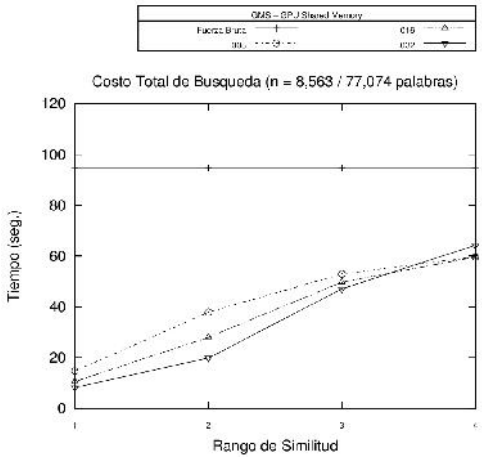
No obstante, el análisis sobre estructuras métricas es más complejo y debe realizarse considerando el coste de procesamiento y almacenamiento de dichas estructuras en la jerarquía de memoria de los dispositivos. Así, se hace necesario realizar distintas evaluaciones en cuanto al tamaño de las bases de datos y sus respectivas estructuras métricas, y su

comportamiento en cuanto a distintas cantidades de pivotes y, por ende, a estructuras de distintos tamaños.

Con vistas a profundizar el estudio, se presentará el resultado de distintos experimentos que permiten dar cuenta del rendimiento de las estructuras métricas bajo las distintas técnicas de paralelismo. Para ello se han realizado experimentos utilizando estructuras con distintas cantidades de pivotes (1, 2, 4, 8, 16, 32, 64, 128, 256, 512 y 1024 pivotes) en OpenMP y GPU (memoria global y luego memoria compartida) sobre espacios métricos de distintas características en cuanto a los siguientes parámetros:

- cantidad de elementos (y el respectivo tamaño de las estructuras GMS creadas)
- tamaño de los objetos: dimensión (vectores) o largo máximo de la palabra con mayor cantidad de caracteres (palabras).
- función de distancia utilizada.

Inicialmente, se exhibirán los resultados del procesamiento sobre tres diferentes espacios métricos en una serie de gráficos en donde se podrá observar el tiempo de procesamiento en fuerza bruta y el procesamiento sobre una estructura métrica genérica con distintas cantidades de pivotes. Cada gráfico corresponde a un espacio métrico distinto en cuanto a la dimensión de sus objetos, la cantidad de objetos y el tipo de objeto (y su respectiva función de evaluación de distancia), y todos los experimentos han sido realizados utilizando la memoria compartida de la GPU. A modo de simplificar su exhibición, se han incluido los resultados del procesamiento utilizando Fuerza Bruta y la estructura métrica genérica con 16 y 32 pivotes.



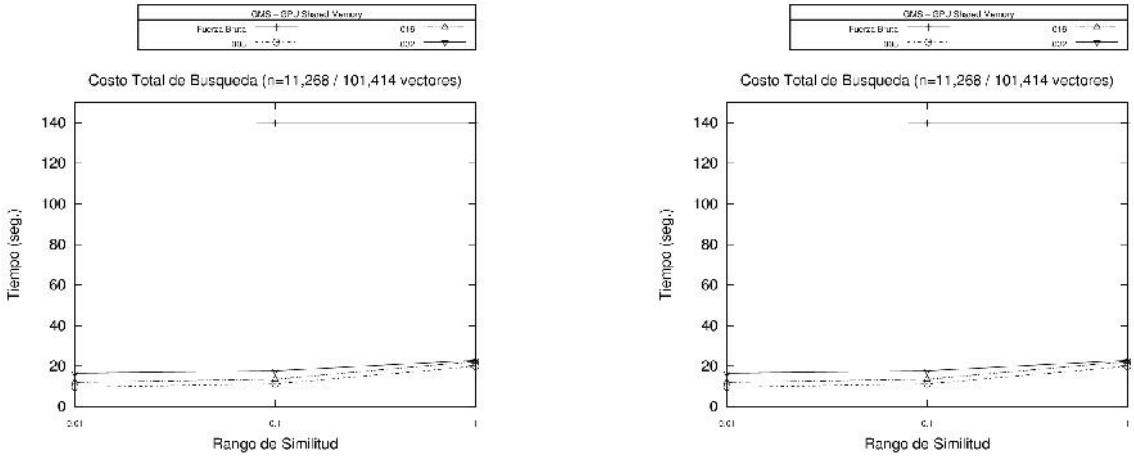


Figura 6. Resultados del Procesamiento - Fuerza Bruta y Estructura GMS con 16 (izq.) y 32 (der.) pivotes en distintos espacios métricos.

Los resultados obtenidos en los experimentos exhibidos en la revelan que los tiempos de búsqueda de la GMS han sido mejores que el procesamiento por Fuerza Bruta para 8, 16 y 32 pivotes en la mayoría de los casos. Cabe resaltar que en el gráfico a la izquierda presenta una excepción: el uso de fuerza bruta ha presentado un resultado ligeramente más eficiente que la estructura métrica de 32 pivotes en el rango de búsqueda más alto (4).

También es interesante observar que existe una variación en los tiempos de procesamiento de acuerdo al rango de búsqueda aplicado en las rutinas que utilizan la estructura GMS, al contrario de lo que ocurre en el algoritmo de Fuerza Bruta, en donde los tiempos de procesamiento son constantes en los distintos rangos de búsqueda. Es posible constatar que el tiempo de procesamiento sobre estructuras métricas tiende a aumentar a medida en que el rango de búsqueda aumenta. Ello se debe principalmente a que se tiende a realizar más evaluaciones de distancia a medida en que el rango de búsqueda aumenta.

Considerando que el comportamiento mencionado se replica en los espacios métricos descritos en este trabajo, se exhibirá a continuación el resultado de una batería de experimentos cuyo objetivo es constatar el método de paralelismo más eficiente sobre estructuras métricas genéricas en distintos espacios métricos. Los resultados son presentados bajo el siguiente esquema: cada columna de figuras representa un espacio métrico en particular. Se despliegan tres gráficos en una misma escala, que representan las distintas modalidades de procesamiento (de izquierda a derecha: OpenMP, GPU con memoria global y GPU con memoria compartida). El eje “x” corresponde a distintos rangos de búsqueda, en tanto el eje “y” refleja los tiempos de procesamiento, en segundos.

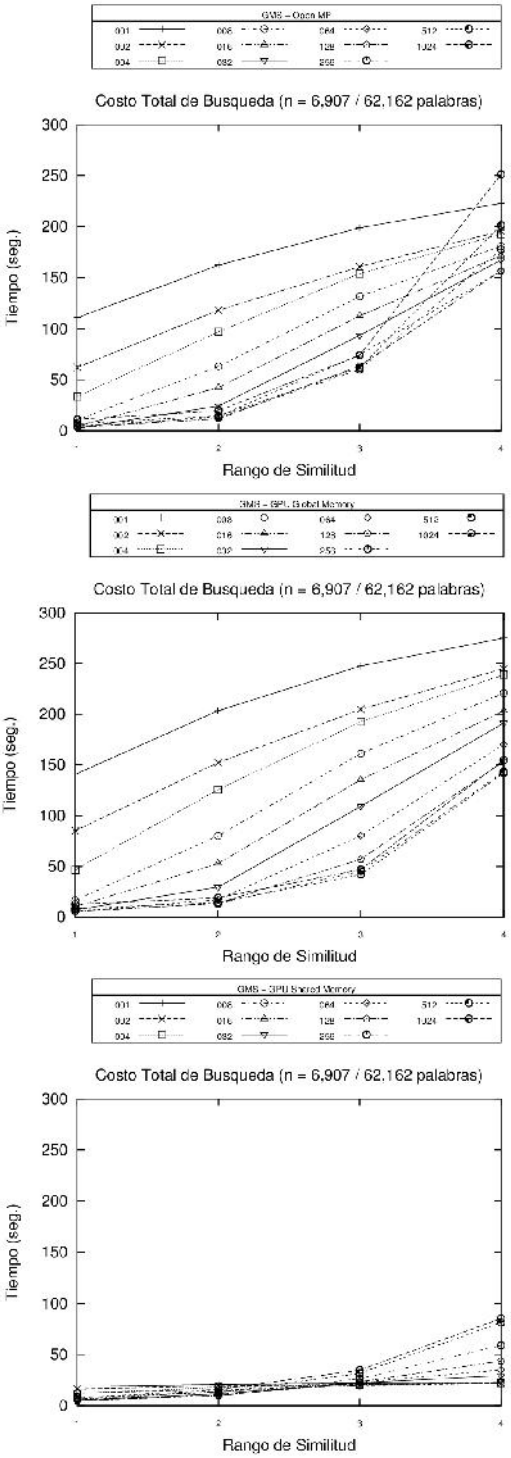


Figura 7. GMS (sup. a inf.) Open MP, GPU Global y GPU memoria compartida. Espacio métrico de Palabras (Inglés).

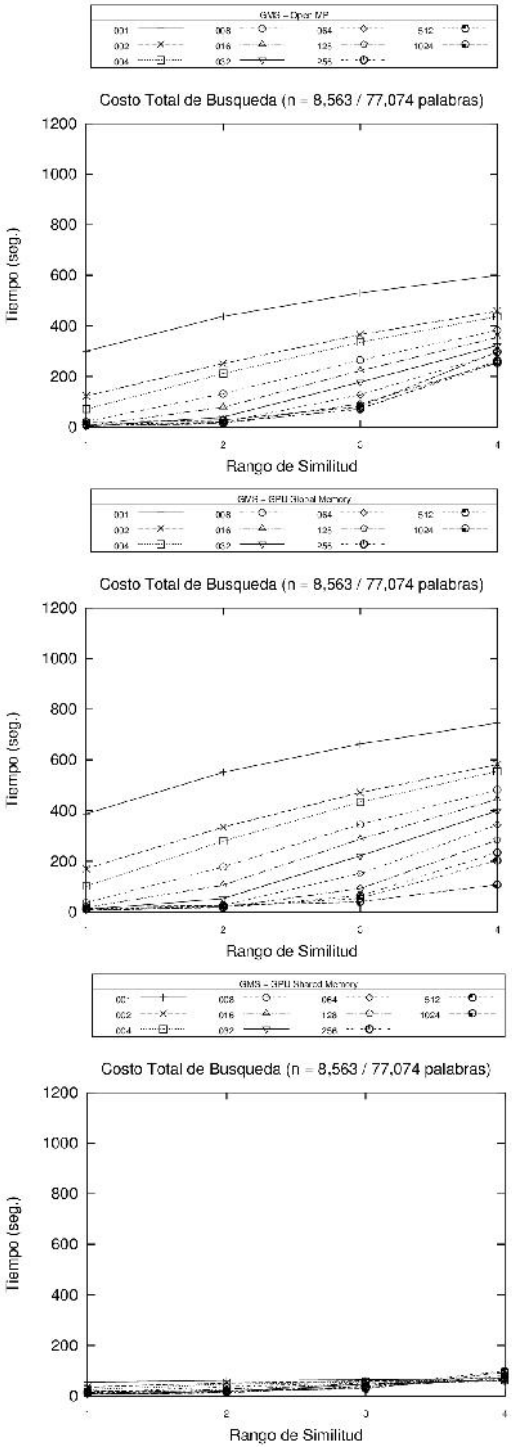


Figura 8. GMS (sup. a inf.) Open MP, GPU Global y GPU memoria compartida. Espacio métrico de Palabras (Noruego).

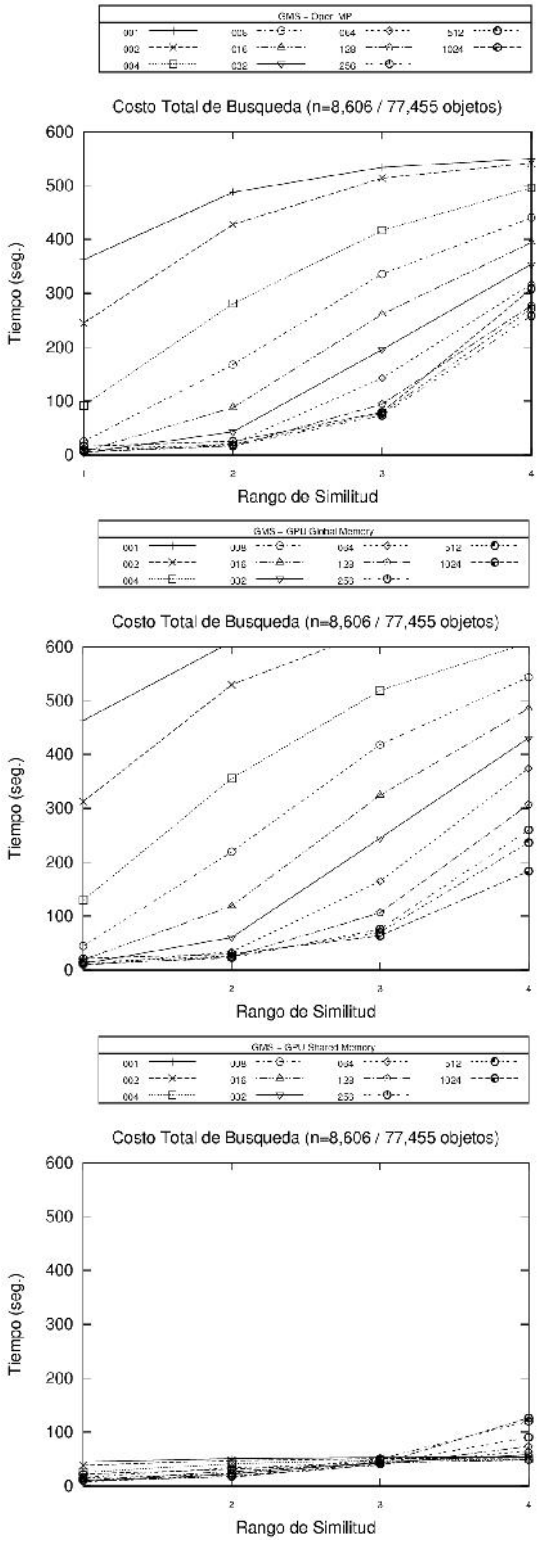


Figura 9. GMS (sup. a inf.) Open MP, GPU Global y GPU memoria compartida. Espacio métrico de Palabras (Español I).

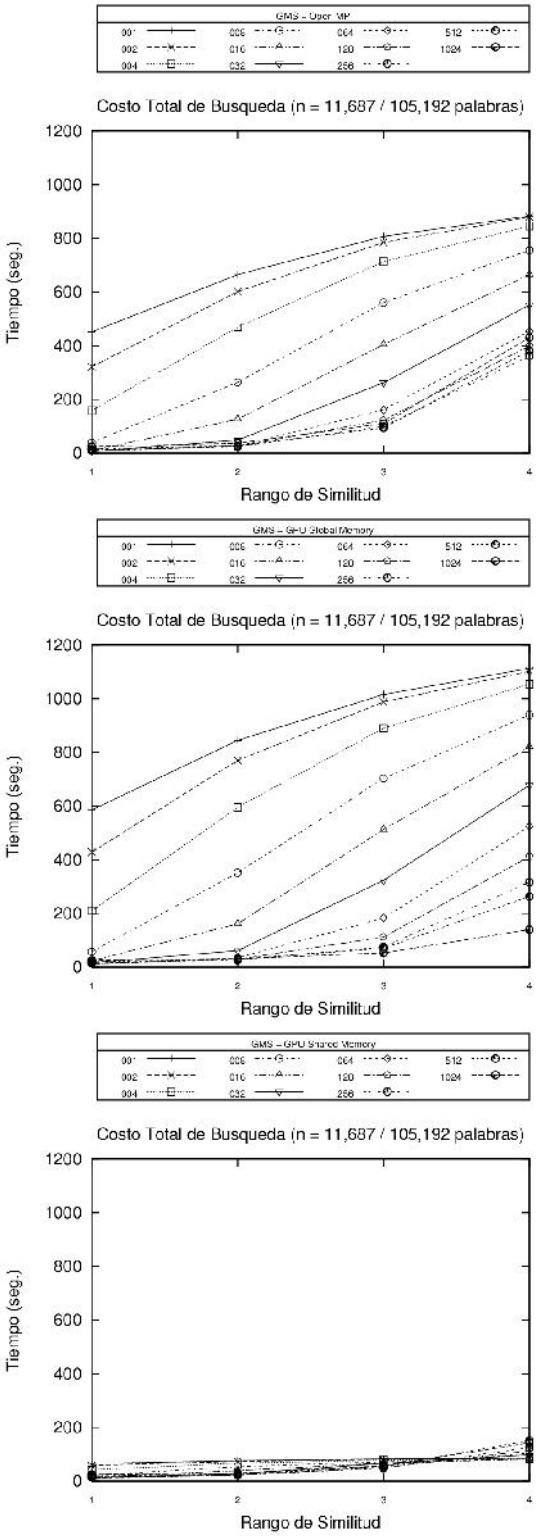


Figura 10. GMS (sup. a inf.) Open MP, GPU Global y GPU memoria compartida. Espacio métrico de Palabras (Italiano).

Las figuras anteriores permiten corroborar que los tiempos de búsqueda con la estructura métrica genérica aumentan a medida que aumentan los rangos de búsqueda en todos los entornos de paralelismo y en cualquier cantidad de pivotes, para todos los casos.

Por otro lado, es posible observar en los experimentos que el rendimiento del procesamiento con la memoria compartida de la GPU resulta conveniente a medida en que aumenta el rango de búsqueda (consecuentemente la cantidad de evaluaciones de distancia realizadas). Efectivamente, se identificaron algunos casos en que el procesamiento con OpenMP ha sido más eficiente que el procesamiento con GPU: en los espacios métricos de palabras, se identificaron algunos casos particulares con el rango de búsqueda más bajo (1), como se puede observar a continuación:

Espacio Métrico	***SpeedUp GPU SM x OpenMP***				***SpeedUp GPU SM x GPU Global***			
	Rango 1	**Rango 2**	**Rango 3**	**Rango 4**	**Rango 1**	**Rango 2**	**Rango 3**	**Rango 4**
Inglés	***0,46***	2,09	4,19	5,69	1,31	2,53	4,90	6,51
Noruego	***0,44***	2,03	3,78	5,02	1,27	2,70	4,70	6,22
Español I	***0,44***	1,97	4,45	6,08	1,32	2,76	5,56	7,39
Italiano	***0,50***	1,77	3,95	5,64	1,23	2,24	4,88	6,91
Francés	***0,50***	2,04	4,23	6,09	1,23	2,46	5,06	7,20

Tabla 3-7. Speedup obtenidos en una estructura GMS basada en 32 pivotes sobre distintos espacios métricos.

3.3.3 Impacto de la selección de pivotes

Los resultados exhibidos en la sección anterior demuestran una mejora significativa en el procesamiento utilizando la memoria compartida de la GPU, constatando así su eficiencia.

Por otro lado, de acuerdo a la literatura vigente, algunos criterios para la eficiencia de las estructuras basadas en pivotes son por un lado la selección de cantidades de pivotes y, por otro lado, la elección de los pivotes. Los experimentos a continuación son un detalle de los resultados obtenidos anteriormente utilizando la memoria compartida de la GPU, y nos permitirán visualizar el comportamiento de las estructuras en distintas cantidades de pivotes. Cabe resaltar que los experimentos realizados han utilizado pivotes seleccionados aleatoriamente. Asimismo, las bases utilizadas no superan la capacidad de procesamiento del hardware utilizado en cuanto al volumen de datos.

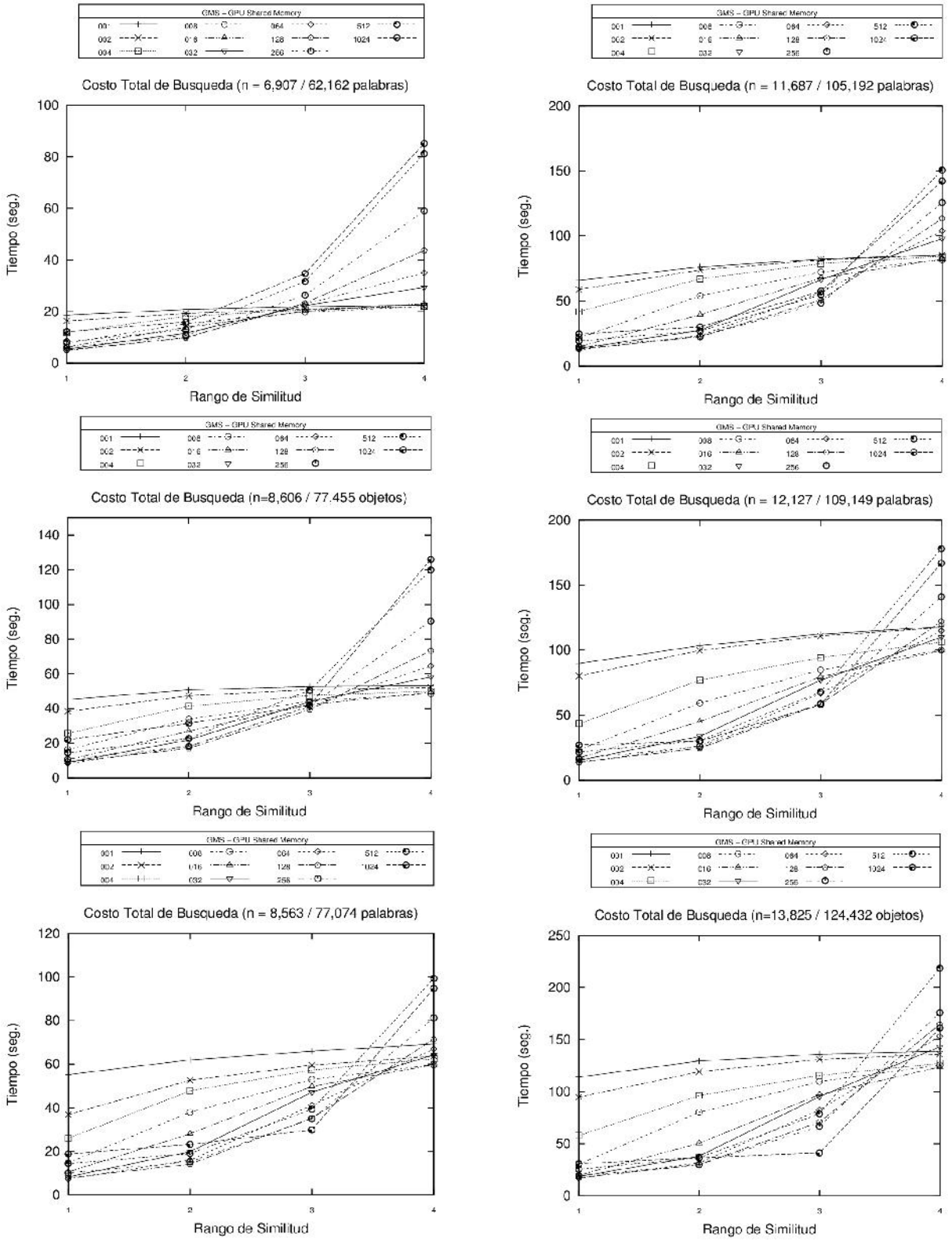


Figura 11. Experimentos con estructura GMS en GPU - memoria compartida.

Es posible destacar algunas conclusiones de los experimentos llevados a cabo, que permiten identificar algunas tendencias en cuanto a la cantidad de pivotes utilizados: por un lado, se registra un decremento en el rendimiento (aumento en los tiempos) a medida que se aumenta

el rango de búsqueda en estructuras con altas cantidades de pivotes. Por otro lado, el comportamiento de las estructuras con pocos pivotes presenta una mejora en la medida en que aumenta el rango de búsqueda.

3.3.4 Efecto de la Maldición de la Dimensionalidad

Tal como fue expresado en el marco teórico, las investigaciones actuales plantean que de un modo general la búsqueda por similitud en espacios métricos se torna intrínsecamente más difícil mientras mayor sea la dimensión del espacio, dando origen al fenómeno conocido como *maldición de la dimensionalidad*.

Para corroborar este fenómeno, se ha propuesto realizar experimentos sobre espacios métricos con la misma cantidad de elementos (10000 objetos de consulta sobre 90000 elementos) de vectores de distintas dimensiones. Se ha utilizado la misma función de evaluación de distancia para todos los experimentos.

En la primera fila, de izquierda a derecha, se observa el resultado del procesamiento de vectores de dimensiones 10 y 15. Luego, vectores de dimensión 20 y 101. Los resultados pueden ser apreciados a continuación:

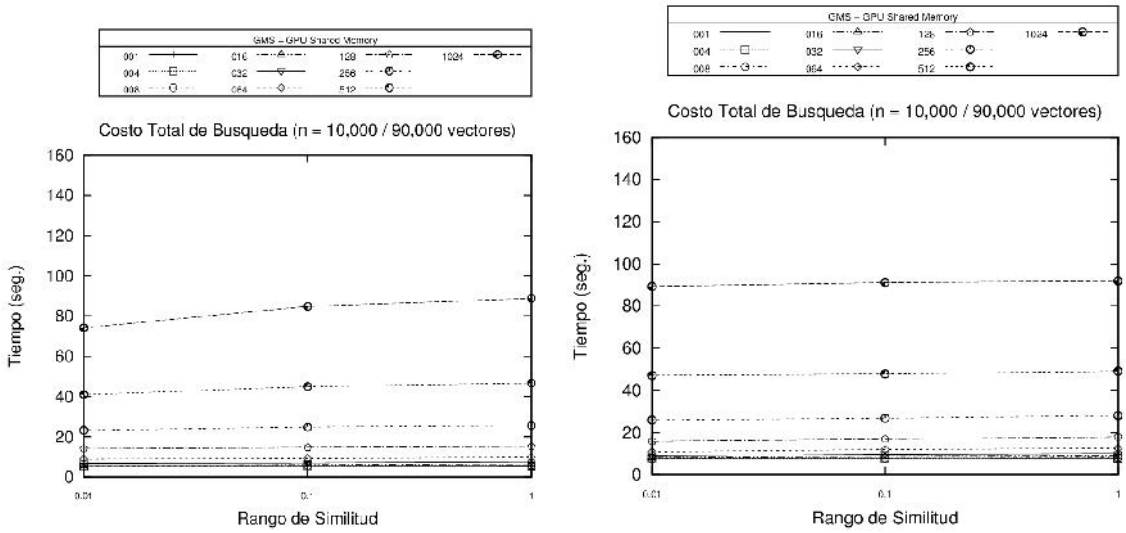


Figura 12. Tiempos de procesamiento sobre vectores de dimensión 10 (izq.) y 15 (der.) utilizando GMS con pivotes.

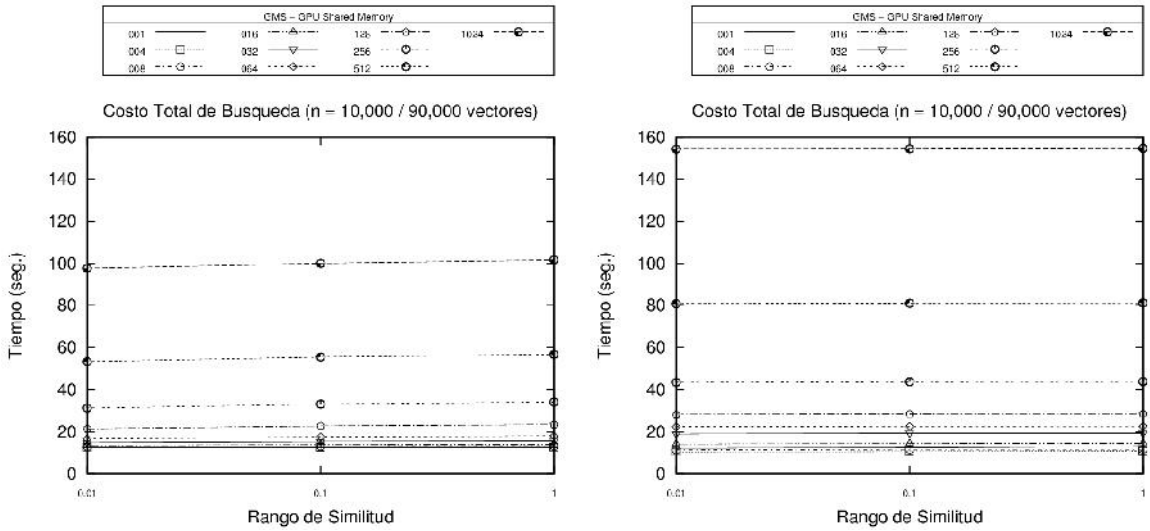


Figura 13. Tiempos de procesamiento sobre vectores de dimensión 20 (izq.) y 101 (der.) utilizando GMS con pivotes.

Para facilitar el análisis de los resultados de los experimentos, los gráficos exhibidos en la Figura 12 y en la Figura 13 son propuestos en la misma escala. Así, la visualización de los gráficos permite dar cuenta de los tiempos de procesamiento. Es posible constatar un incremento en los tiempos de procesamiento a medida en que aumenta la dimensión de los objetos de un espacio métrico.

4 CONCLUSIONES y TRABAJOS FUTUROS

La necesidad de procesar grandes volúmenes de datos requiere de incrementar la capacidad de procesamiento y reducir los tiempos de búsqueda promedio. En este contexto, es relevante el estudio en términos de la paralelización de los algoritmos y de la distribución de la base de datos. Por otro lado, el aumento de tamaño de las bases de datos y la aparición de nuevos tipos de datos sobre los cuales no interesa realizar búsquedas exactas, crean la necesidad de plantear nuevas estructuras para búsqueda por similitud o búsqueda aproximada. Así también, las aplicaciones reales requieren que dichas estructuras permitan ser almacenadas en memoria secundaria eficientemente, como también que posean métodos optimizados para reducir los costos de accesos a disco.

Hemos presentado en este Informe las principales características de los llamados espacios métricos y algunas estructuras métricas utilizadas para la realización de búsquedas por similitud en dichos espacios. Se ha profundizado la discusión sobre aquellos experimentos basados en la utilización de estructuras genéricas basadas en pivotes, comparados con el procesamiento linear o secuencial conocido como procesamiento por fuerza bruta. Por otro lado, hemos presentado los resultados de experimentos utilizando distintas estrategias de procesamiento paralelo, analizando el rendimiento de la búsqueda en ambientes con GPU (CUDA) y multi-core (OpenMP). También se corroboró la hipótesis conocida como maldición de la dimensionalidad.

Con base en el marco teórico vigente y en los experimentos realizados, es posible arribar a las siguientes conclusiones:

- Las distintas tecnologías de procesamiento paralelo utilizadas permiten lograr una optimización en los tiempos de procesamiento si son comparadas con el procesamiento secuencial, registrando distintos coeficientes de speedup. En un análisis más detallado, es posible finalmente observar en líneas generales un grado de optimización en el rendimiento cuando se utiliza el procesamiento basado en la memoria compartida de la GPU en relación a las demás opciones utilizadas, excepto en algunas implementaciones de la estructura métrica basada en pivotes en los que el rango de búsqueda es bajo y por ende no se realizan muchas operaciones de evaluación de distancia. Por otro lado, el rango de búsqueda no es un factor determinante en el rendimiento de las aplicaciones en el algortimo de fuerza bruta. Bajo este algoritmo, la utilización de la memoria compartida de la GPU se ha mostrado más eficiente que las demás modalidades de paralelismo en todos los casos registrados.

- Para la mayoría de los experimentos, la utilización de estructuras métricas basadas en una cantidad moderada de pivotes ha permitido obtener tiempos de procesamiento inferiores al uso del algoritmo de Fuerza Bruta. Se ha corroborado que los tiempos de búsqueda con la estructura métrica genérica aumentan a medida que aumentan los rangos de búsqueda en todos los entornos de paralelismo y en cualquier cantidad de pivotes, para todos los casos. El procesamiento con GPU resulta más eficiente a medida que aumenta el rango de búsqueda -y consecuentemente la cantidad de evaluaciones de distancia realizadas-.

- Por otro lado, se identificaron algunas tendencias en cuanto al impacto de la cantidad de pivotes utilizados sobre la estructura métrica genérica: mientras se registra un aumento en los tiempos a la medida que se aumenta el rango de búsqueda en estructuras con altas cantidades de pivotes, el comportamiento de las estructuras con pocos pivotes presenta mejores tiempos de procesamiento en la medida en que aumenta el rango de búsqueda.

- Finalmente, ha sido posible constatar un incremento en los tiempos de procesamiento a medida que aumenta la dimensión de los objetos de un espacio métrico, validando así la hipótesis del fenómeno conocido como maldición de la dimensionalidad.

Como trabajo futuro se propone discutir la escalabilidad del uso de esas plataformas, ya que actualmente se proponen experimentos en equipos individuales. Así, se busca llegar a soluciones híbridas que permitan la distribución de los espacios métricos y sus correspondientes estructuras, de modo que sea posible realizar aplicaciones sobre volúmenes de datos en escala de producción.

Por otro lado, otra línea de gran interés a ser abordada está relacionada con la eficiencia energética en el uso de distintos dispositivos y tecnologías para el procesamiento paralelo de grandes volúmenes de información en lo referido a la búsqueda por similitud sobre espacios métricos.

5 AGRADECIMIENTOS

El presente trabajo fue posible gracias a los aportes de la Universidad Nacional de la Patagonia Austral a través del proyecto acreditado 29/A274, "***Búsquedas por similitud en espacios métricos sobre plataformas basadas en GPUs***".

6 ANEXOS

6.1 Publicaciones

1. *Roberto Uribe-Paredes, Diego Cazorla, Enrique Arias, José Luis Sánchez. "**Estudio de Diferentes Esquemas de Planificación para el Reparto de Consultas en una Plataforma Heterogénea**". Actas XXIV Jornadas de Paralelismo (JPAR2013). Madrid, España, Sept. 2013*
2. *Roberto Uribe-Paredes, Diego Cazorla, Enrique Arias, José Luis Sénchez. "**An approach to an Efficient Scheduling Scheme for Delivering Queries to Heterogenous Clusters in the Similarity Search Problem**". In 13th International Conference Computational and Mathematical Methods in Science and Engineering. Almería, Spain, June 2013*
3. *Osiris SOFIA, Jacobo SALVADOR, Eder DOS SANTOS, Roberto URIBE PAREDES. **Búsquedas por Rango sobre Plataformas GPU en Espacios Métricos**. pp 658 - 662. XV Workshop de Investigadores en Ciencias de la Computación - WICC 2013. 18 al 18 de Abril de 2013. Universidad Autónoma de Entre Ríos, Paraná, Argentina. Proceeding Publicados en CD por Universidad Autónoma de Entre Ríos y RedUNCI. ISBN13: 978-987-28179-6-1*
4. *Osiris SOFIA, Jacobo SALVADOR, Eder DOS SANTOS, Roberto URIBE PAREDES. **Búsquedas por Similitud en Espacios Métricos sobre Plataformas Basadas en GPUs**. II Encuentro de Investigadores de la Patagonia Austral, Universidad Nacional de la Patagonia Austral, Unidad Académica San Julián, Puerto San Julián, Argentina. 7 de septiembre de 2012. Proceeding Publicados en CD por Universidad Nacional de la Patagonia Austral.ISBN13: 978-987-1242-66-5.*
5. *Roberto Uribe-Paredes, Enrique Arias, Diego Cazorla, José Luis Sánchez. **Una estructura Métrica Genérica para Búsquedas por Rango sobre una Plataforma Multi-GPU**. XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD2012). Sept. 2012, Almería, España. ISBN 978-84-15487-28-9*
6. *Roberto Uribe-Paredes, Diego Cazorla, Enrique Arias, Jossé Luis Sánchez. **Acelerando la Búsqueda por Rango con un Sistema Híbrido de Memoria Compartida**. Actas XXIII Jornadas de Paralelismo (JP2012). Sept. 2012, Elche, España. ISBN 978-84-695-4471-6*

6.2 Presentaciones y otras

7. *Curso de Posgrado de la Maestría en Informática y Sistemas de la UNPA, "**Paradigmas de Computación Paralela, Concurrente y Distribuida**", dictado presencial y virtual.*

7 REFERENCIAS

[**ADT02**] Adil Alpkocak, Taner Danisman, and Ulker Tuba. A parallel similarity search in high dimensional metric space using m-tree. In Advanced Environments, Tools, and Applications for Cluster Computing, volume 2326 of LNCS, pages 247--252. Springer Berlin / Heidelberg, 2002.

[**BDHK06**] Benjamin Bustos, Oliver Deussen, Stefan Hiller, and Daniel Keim. A graphics hardware accelerated algorithm for nearest neighbor search. In Computational Science (ICCS), volume 3994, pages 196--199. Springer, 2006.

[**BELL57**] Bellman, R.E. 1957. Dynamic Programming. Princeton University Press, Princeton, NJ. Republished 2003: Dover

[**BGTPM11**] Ricardo J. Barrientos, José I. Gómez, Christian Tenllado, Manuel Prieto, and Mauricio Marín. *kNN* query processing in metric spaces using *GPUs*. In 17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011), volume 6852 of LNCS, pages 380--392, Berlin, Heidelberg, 2011. Springer-Verlag.

[**Brin95**] Sergei Brin. Near neighbor search in large metric spaces. In the 21st VLDB Conference, pages 574—584. Morgan Kaufmann Publishers, 1995.

[**CMB99**] Edgar Chávez, José L. Marroquín, and Ricardo Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In 6th International Symposium on String Processing and Information Retrieval (SPIRE'99), pages 38--46. IEEE CS Press, 1999.

[**CN00**] E. Chávez and G. Navarro. Measuring the dimensionality of general metric spaces. Technical Report TR/DCC-00-1, Dept. of Computer Science, University of Chile, 2000.

[**CN01**] Chávez E.; Navarro G. Towards measuring the searching complexity of metric spaces. In Proc. Mexican Computing Meeting, volume II, pages 969–978, Aguascalientes, México, 2001. Sociedad Mexicana de Ciencias de la Computación.

[**CNBM01**] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. ACM Computing Surveys, 33(3):273--321, 2001.

[**CPM94**] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. "Proximity matching using fixedqueries trees". In 5th Combinatorial Pattern Matching (CPM'94), 1994, LNCS 807, pp. 198–212.

[**CPZ97**] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-Tree : An efficient access method for similarity search in metric spaces. In the 23st International Conference on VLDB, pages 426--435, 1997.

[**CUDA**] NVIDIA CUDA C Programming Guide, Version 4.0. 2011. http://developer.nvidia.com/object/ gpucomputing.html

[**DEXA12**] Roberto Uribe-Paredes, Enrique Arias, José L. Sánchez, and Diego Cazorla. "Improving the Performance for the Range Search on Metric Spaces using a Multi-GPU Platform". To appear: 23rd International Conference on Database and Expert Systems Applications (DEXA 2012). Vienna, Austria, Sept. 2012.

[**FQA01**] E. Chávez, J. Marroquín, and G. Navarro. "Fixed queries array: A fast and economical data structure for proximity searching". Multimedia Tools and Applications, vol. 14, no. 2, pp. 113–135, 2001.

[**FM07**] Wu-Feng and Dinesh Manocha. High-performance computing using accelerators. Parallel Computing, 33:645--647, 2007.

[**GBMB10**] Veronica Gil-Costa, Ricardo Barrientos, Mauricio Marín, and Carolina Bonacic. Scheduling metric-space queries processing on multi-core processors. Euromicro Conference on Parallel, Distributed, and Network-Based Processing, 0:187--194, 2010.

[**GDB08**] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using GPU. Computer Vision and Pattern Recognition Workshop, 0:1--6, 2008.

[**GKKG03**] Grama, Ananth and Karypis, George and Kumar, Vipin and Gupta, Anshul. Introduction to Parallel Computing (2nd Edition). Addison Wesley. 2003.

[**GMR09**] Veronica Gil-Costa, Mauricio Marín, and Nora Reyes. Parallel query processing on distributed clustering indexes. Journal of Discrete Algorithms, 7(1):3--17, 2009.

[**ICCSDE12**] Roberto Uribe-Paredes, Diego Cazorla, José L. Sánchez, and Enrique Arias. "A comparative study of different metric structures: Thinking on gpu implementations". In International Conference of Computational Statistics and Data Engineering (ICCSDE'12), London, England, July 2012.

[**ISCSCT09**] Quansheng Kuang and Lei Zhao. "A practical GPU based kNN algorithm". International Symposium on Computer Science and Computational Technology (ISCSCT), pp. 151–155, 2009.

[**ISPA12**] R.J. Barrientos, J.I. Gómez, C. Tenllado, M. Prieto, and M. Marin. "Range query processing in a multi-GPU enviroment". In 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2012), Madrid, Spain, July 2012.

[**JP12**] Roberto Uribe-Paredes, Diego Cazorla, Enrique Arias and José L. Sánchez. "Acelerando la Búsqueda por Rango con un Sistema Híbrido de Memoria Compartida". To appear: Actas XXIII Jornadas de Paralelismo (JP2012). Sept. 2012, Elche, España.

[**KZ09**] Quansheng Kuang and Lei Zhao. A practical {GPU} based {kNN} algorithm. International Symposium on Computer Science and Computational Technology (ISCSCT), pages 151--155, 2009.

[**MOV94**] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. Pattern Recognition Letters, 15(1):9--17, January 1994.

[**MPI94**] W. Gropp, E. Lusk, and A. Skelljum. Using MPI: Portable Parallel Programming with the Message Passing Interface. Scientific and Engineering computation Series. MIT Press, Cambridge, MA, 1994.

[**MT98**] Zezula, Pavel and Savino, Pasquale and Rabitti, Fausto and Amato, Giuseppe and Ciaccia, Paolo. Processing M-trees with Parallel Resources. In RIDE '98: Proceedings of the Workshop on Research Issues in Database Engineering. IEEE CS. 1998.

[**Nav02**] Gonzalo Navarro. Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ), 11(1):28--46, 2002.

[**NN97**] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(9):989--1003, 1997.

[**OMP07**] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press, 2007.

[**PVM94**] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. PVM: Parallel Virtual Machine -- A User's Guide and Tutorial for Network Parallel Computing. MIT Press, 1994.

[**SODA93**] Yianilos P. *Data structures and algorithms for nearest neighbor search in general metric spaces*. In Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93), pages 311–321, 1993.

[**SOFSEM07**] Oscar Pedreira and Nieves R. Brisaboa. "Spatial selection of sparse pivots for similarity search in metric spaces". In 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007), Harrachov, Czech Republic, 2007, vol. 4362 of LNCS, pp. 434–445, Springer.

[**TTSF00**] Caetano Traina, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In VII International Conference on Extending Database Technology, pages 51--61, 2000.

[**UN09**] Roberto Uribe-Paredes and Gonzalo Navarro. Egnat: A fully dynamic metric access method for secondary memory. In Proc. 2nd International Workshop on Similarity Search and Applications (SISAP), pages 57--64, Prague, Czech Republic, August 2009. IEEE CS Press.

[UP13] Roberto Uribe-Paredes. Tesis de Doctorado: Estructuras Métricas para Búsquedas por Similitud sobre Arquitecturas Heterogéneas Basadas en GPUs, Universidad de Castilla - La Mancha, Doctorado en Ciencias Informáticas, Julio 2013.

[**UACS12**] Roberto Uribe-Paredes, Enrique Arias, Diego Cazorla, José Luis Sánchez. "Una estructura Métrica Genérica para Búsquedas por Rango sobre una Plataforma Multi-GPU". XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD2012). Sept. 2012, Almería, España.

[**UVASC11**] Roberto Uribe-Paredes, Pedro Valero-Lara, Enrique Arias, José L. Sánchez and Diego Cazorla. "Similarity search implementations for multi-core and many-core processors". In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 656–663 (July 2011). Istanbul, Turkey.

[**Vid86**] E. Vidal. An algorithm for finding nearest neighbor in (approximately) constant average time. Pattern Recognition Letters, 4:145--157, 1986.

[**WSP97**] Chávez E.; Marroquín J. Proximity queries in metric spaces. In R. Baeza-Yates, editor, Proc. 4th South American Workshop on String Processing (WSP'97), pages 21–36. Carleton University Press, 1997.