Búsqueda exhaustiva utilizando el coprocesador Intel Xeon Phi

Carlos M. Toledo
Departamento de Ingeniería de Sistemas,
Universidad de La Frontera, Chile.
Email: c.toledo01@ufromail.cl

Ricardo J. Barrientos
Centro de Modelación y Computación Científica (CMCC),
Universidad de La Frontera, Chile.
Email: ricardo.barrientos@ufrontera.cl

***Resumen*—En los últimos años, el uso de coprocesadores ha sido bastante estudiado en el campo de Computación Paralela, sobre todo para acelerar procesos secuenciales. Tal es el caso de la GPU (Graphic Process Unit), que es un coprocesador capaz de procesar determinadas funciones de un programa con una gran cantidad de hilos sobre una gran cantidad de núcleos.**

Recientemente, Intel ha lanzado un coprocesador de características similares a la GPU, denominado *Intel Xeon Phi*. El coprocesador Intel Xeon Phi posee hasta 61 núcleos interconectados por un anillo bidireccional de alto rendimiento con capacidad de procesar instrucciones vectoriales. También, reporta hasta 1208 GFLOP/s (miles de millones de operaciones en punto flotante por segundo) sobre datos con precisión doble, que es un rendimiento similar al de la GPU de NVIDIA.

En el presente trabajo, se proponen y analizan diferentes algoritmos exhaustivos para resolver *consultas por rango* utilizando un coprocesador Intel Xeon Phi. El utilizar algoritmos exhaustivos para resolver consultas por rango otorga la ventaja de no estar limitado a un tipo de base de datos, como suele ocurrir al resolver consultas de este tipo, limitándose a espacios métricos. Se comparan los resultados obtenidos con alternativas multi-core indexadas y no indexadas, obteniendo interesantes resultados de speed-up debido al buen uso de memoria caché en la Xeon Phi.

***Index Terms*—Consultas por rango, Arquitectura many-core, Intel MIC, Intel Xeon Phi, Computación Paralela.**

I. Introducción

Últimamente, el uso de coprocesadores orientado a cómputo intensivo ha sido ampliamente estudiado en el área de Computación Paralela, siendo la GPU (Graphics Processing Units) [1], [2] un ejemplo de coprocesador. Estos coprocesadores poseen una *arquitectura many-core*, que es el nombre usualmente utilizado para indicar una arquitectura multi-core masiva, es decir, que posee una cantidad de núcleos muy superior a un procesador multi-core convencional. Además de poseer una gran cantidad de núcleos, están diseñados para maximizar la cantidad de operaciones en punto flotante por segundo, pero todos ellos poseen el problema de la gran latencia al acceder a datos de su memoria principal.

Recientemente Intel también ha lanzado un coprocesador denominado Intel Xeon Phi (basado en arquitectura Intel MIC) [3], que está compuesto de hasta 61 núcleos, y alcanza un rendimiento de hasta 1208 GFLOP/s (miles de millones de operaciones en punto flotante por segundo) sobre punto flotante con precisión doble, que es un rendimiento similar al de una GPU de NVIDIA.

El coprocesador Xeon Phi es compatible con los demás modelos de programación estándar de Intel, como OpenMP[4], [5], POSIX Threads [6] o MPI [7].

En el presente trabajo se proponen distintos algoritmos para resolver consultas por rango utilizando un coprocesador Intel Xeon Phi, y se compara sus resultados contra algoritmos multi-core de búsqueda exhaustiva y también utlizando indexación.

La estructura de este trabajo se describe a continuación. En la Sección II se describe la arquitectura de la Intel Xeon Phi, y también el trabajo relacionado. En la Sección III se describen nuestras propuestas de algoritmos sobre la Intel Xeon Phi. En la Sección IV se muestran los resultados experimentales entre las versiones en la Xeon Phi y algoritmos multi-core con y sin indexación. Finalmente en la Sección V se muestran las principales conclusiones del presente trabajo.

II. Conocimientos Básicos y Trabajo Relacionado

II-A. Coprocesador Intel Xeon Phi

El coprocesador Intel Xeon Phi [3] consta de hasta 61 núcleos conectados en un anillo bidireccional de alto rendimiento en el chip. El coprocesador utiliza un sistema operativo Linux y soporta todas las herramientas de desarrollo más importantes de Intel, como el compilador C/C++, Fortran, MPI, OpenMP y bibliotecas de altas prestaciones como MKL. Las tradicionales herramientas UNIX son soportadas por el coprocesador vía BusyBox, las que combina pequeñas versiones de muchas utilidades comunes de UNIX dentro de un único ejecutable. El coprocesador esta conectado a un procesador Intel Xeon (Host) vía el bus PCI Express (PICe). La implementación

de una virtualización TCP/IP apilada permite el acceso al coprocesador como un nodo de red. Más información sobre la arquitectura del hardware puede ser encontrada en [8]. En los siguientes puntos se citan las más importantes propiedades de la arquitectura MIC (que pueden ser vistas en la Figure 1(a)).

Core: El núcleo de la Intel Xeon Phi (*Scalar Unit* en la Figura 1(a)) obedece a una arquitectura en-orden (basada en la familia de procesadores Intel Pentium). Implementa instrucciones de recuperación y decodificación para 4 hilos (hardware) por núcleo. Las instrucciones vectoriales que posee el coprocesador Intel Xeon Phi, utilizan una unidad en punto flotante (*VPU*) dedicada, con ancho de vector 512-bit, la que está disponible en cada núcleo. Es posible ejecutar dos instrucciones por ciclo de reloj, uno en el U-pipe y otro en el V-pipe (no todos los tipos de instrucciones pueden ser ejecutados por el V-pipe). Cada núcleo compone una interconexión en anillo mediante la CRI (Core Ring Interface).

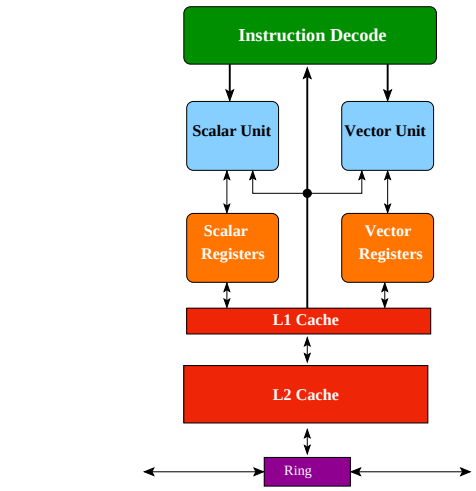
Vector Processing Unit (VPU): La VPU incluye la EMU (Extended Math Unit), y es capaz de realizar 16 operaciones en punto flotante con precisión simple por ciclo, 16 operaciones de enteros de 32-bit por ciclo u 8 operaciones en punto flotante con precisión doble por ciclo.

L1 Cache: Tiene una caché L1 de 32KB para instrucciones y de 32KB para datos, asociativa de 8-vías, con tamaño de línea de caché de 64 bytes. Tiene una latencia de cargado de 1 ciclo, que significa que un valor entero cargado desde L1 puede ser usado en el siguiente ciclo por una instrucción entera (instrucciones vectoriales tienen diferente latencia que las enteras).

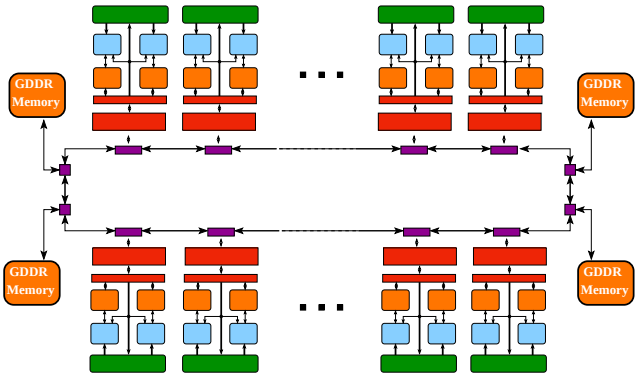
L2 Cache: Cada núcleo posee 512KB de caché L2. Si ningún núcleo comparte algún dato o código, entonces el tamaño total de la caché L2 es de 31 MB. Por otro lado, si todos los núcleos comparten exactamente el mismo código y datos en perfecta sincronía, el tamaño total de la caché L2 sería solo de 512KB. Su latencia es de 11 ciclos de reloj.

Ring: Incluye interfaces y componentes, ring stops, ring turns, direccionamiento y control de flujo. Un coprocesador Xeon Phi tiene 2 de estos anillos, uno viajando en cada dirección (Figure 1(b)).

La Figura 1(b) ilustra una visión general de la arquitectura, donde se muestra como los núcleos están conectados con cachés coherentes mediante un anillo bidireccional de 115GB/sec. La figura también muestra una memoria GDDR5 con 16 canales de memoria, que alcanzan una velocidad de hasta 5.5GB/sec.



(a) Núcleo del coprocesador Intel Xeon Phi.



(b) Interconexión de lo núcleos en una Intel Xeon Phi.

Figura 1. Diagrama de la Arquitectura MIC.

II-B. Consultas por Rango

Las consultas por rango [9] y también las consultas kNN [10] son ampliamente utilizadas en diferentes áreas, tales como, estadística, geometría computacional, inteligencia artificial, bases de datos, biología computacional, reconocimiento de patrones, minería de datos, la Web.

Una consulta por rango se denomina como (q, r), donde q es el elemento consulta y r el rango de búsqueda. Así como indica la Figura 2, el resultado son todos los objetos cuya distancia es menor o igual a r del objeto consulta. Debido a que la solución de una consulta por rango se utiliza como base para resolver consultas kNN, el presente trabajo se enfoca a resolver solamente consultas por rango.

II-C. Trabajo Relacionado

El trabajo relacionado para resolver consultas por rango es amplio, pero destacan los métodos que utilizan indexación, tales como, la Lista de Clusters (LC) [11], EGNAT [12], DSACL$^+$-tree [13], M-Index [14] o Polyphasic Metric Index [15].

Todos ellos intentan descartar elementos de la base de datos para evitar comparar cada elemento contra la con-

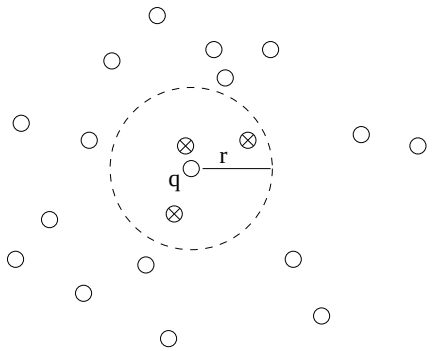


Figura 2. Ejemplo de *consulta por rango* (q,r).

sulta. Para ello restringen la aplicación de sus algoritmos a espacios métricos. Un *espacio métrico* [16] (X, d) está compuesto de una colección de datos $\mathbb{X}$ y una *función de distancia* $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+$ definida sobre los datos. La distancia determina la similitud entre dos objetos dados y debe mantener las siguientes propiedades:

- Positividad: $d(x, y) \geq 0,\ x \neq y \Rightarrow d(x, y) > 0$
- Simetría: $d(x, y) = d(y, x)$
- Desigualdad Triangular: $d(x, y) + d(y, z) \geq d(x, z)$

Sin embargo, en el presente trabajo se proponen algoritmos exhaustivos, lo que implica que no estamos restringidos a utilizar espacios métricos, por lo tanto, las soluciones pueden ser aplicadas a un número mayor de bases de datos, y de distinto tipo.

III. Propuestas de Búsqueda Exhaustiva sobre una arquitectura Intel MIC

A continuación se describen dos diferentes algoritmos exhaustivos utilizando una Intel Xeon Phi para resolver consultas por rango, denominadas MIC v1 y MIC v2. Para ambas estrategias se utilizó el modo offload vía OpenMP para procesar los datos en la Xeon Phi. Los datos fueron fueron mapeados a arreglos de 1 dimensión para ser transferidos a la Xeon Phi.

Para asegurarnos del uso de instrucciones vectoriales, se utilizó `#pragma simd` en la función de distancia, que asegura su uso, y que además puede ser verificado en tiempo de compilación.

III-A. MIC v1

Este método distribuye de manera circular los elementos de la base de datos entre los hilos, es decir cada hilo procesará una porción de la base de datos. Se define una cola global de consultas, y dado que cada hilo sólo tiene acceso a una porción de la base de datos, todas las consultas deben ser procesadas por todos los hilos. Cabe destacar que los datos y consultas son cargados una única vez en la Xeon Phi para evitar la carga innecesaria de datos por cada consulta que se desea resolver.

Siguiendo los objetivos del presente trabajo, sólo se recupera por consulta el número de elementos resultado.

Para evitar barreras de sincronización, cada hilo escribe sus resultados parciales de cada consulta procesada en una variable local en la memoria principal de la Xeon Phi. Al finalizar de procesar el lote completo de consultas, se ejecuta un segundo paso, en donde todos los hilos colaboran para sumar los resultados parciales y encontrar los resultados finales de cada consulta. Dichos resultados son posteriormente copiados a memoria de la CPU.

III-B. MIC v2

Este método distribuye las consultas entre los hilos, y luego cada hilo debe acceder a todos los elementos de la base de datos para resolver sus consultas asignadas. Cada hilo es el responsable de los resultados de su consulta asignada, y es el que escribe su resultado en memoria principal de la Xeon Phi, desde donde son posteriormente copiados a CPU.

Este método tiene por objetivo mejorar el uso de caché, y por eso es que hilos sucesivos acceden a los mismos elementos de la base de datos, aunque comparan sus elementos con una consulta diferente.

III-C. Algoritmos multi-core

Se utilizan dos algoritmos multi-core exhaustivos para tomarlos como referencia, denominados *OpenMP v1* y *OpenMP v2*. Ambos utilizan OpenMP para la programación multithreading, y además se utiliza un núcleo exclusivo por hilo para evitar conflictos de recursos.

OpenMP v1 distribuye los datos entre los hilos. Luego, las consultas son enviadas a todos los hilos, y cada hilo procesará todas las consultas con sus elementos asignados. Luego de cada consulta existe una barrera de sincronización.

OpenMP v2 distribuye las consultas entre los hilos, pero no los elementos de la base de datos. Por lo tanto, cada hilo debe procesar toda la base de datos por cada consulta asignada.

IV. Resultados Experimentales

Los experimentos fueron ejecutados con un coprocesador Intel Xeon Phi 5110P, con 60 núcleos. Los detalles del coprocesador y de la máquina host están descritos en la Tabla I.

Se usó como bases de datos, la colección *CoPhIR* [17] (Content-based Photo Image Retrieval), la que consiste de metadatos extraídos de Flickr. Es una colección de 106 millones de imágenes, y por cada imagen existen cinco descriptores MPEG-7. Para el propósito del presente trabajo sólo se utilizó por cada imagen el descriptor *Color Structure*, el que representa un vector de dimensión

Cuadro I
CARACTERÍSTICAS GENERALES

Procesador	2xIntel Xeon CPU E5-2670, 16-cores (en total) 20M Cache, 2.60 GHz, Sandy Bridge
Coprocesador	Intel Xeon Phi 5110P 60 núcleos de 1.053 GHz 8GB de memoria (ancho de banda 320 GB/s) 30MB L2
Memoria	32 GBytes
Sistema Operativo	GNU Debian System Linux kernel 2.6.32-431 for 64 bits
Compilador	icc version 14.0.3, flags: -03

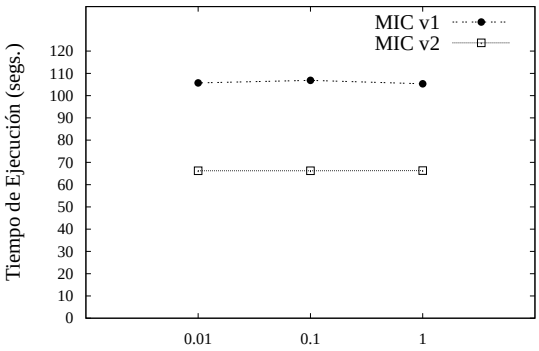
64. Se usó la *distancia euclidiana* como función de distancia. Así como en previos trabajos ([18], [12]), los radios utilizados fueron aquellos que recuperan en promedio el 0.01 %, 0.1 % y 1 % de elementos de la base de datos por consulta.

Como archivo de consultas, se ha utilizado uno recientemente publicado en [19], el que está compuesto de consultas reales, obtenidas desde el sitio web [20]. El archivo de consultas está compuesto de 30,000 imágenes, representadas por su descriptor *Color Structure* de dimensión 64.

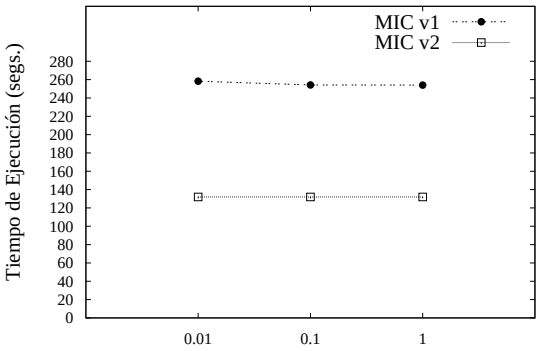
En las versiones que utilizan la Intel Xeon Phi, se utilizan 4 hilos por núcleo, que es la cantidad máxima soportada, y la de mejor rendimiento ([8]). Cabe destacar que el tiempo de copiar los resultados desde la Xeon Phi a la CPU también son tomados en cuenta en los resultados.

Las figuras 3(a), 3(b) y 3(c) presentan el tiempo de ejecución de nuestras dos diferentes versiones sobre la Intel Xeon Phi, sobre la base de datos CoPhIR usando distinto número de elementos. Se aprecia que la versión MIC v2 obtiene una ventaja considerable sobre la base de datos con los tres distintos tamaños. Esta ventaja está dada principalmente porque si más de 1 núcleo accede al mismo dato en la memoria principal de la Xeon Phi, entonces una línea de caché es compartida por los núcleos, tal como se explica en [8]. Recordemos que cada núcleo (capaz de procesar hasta 4 hilos) comparte una caché L1 y una L2 entre los hilos presentes en el núcleo. En la versión MIC v2 cada hilo accede al mismo elemento de la base de datos para compararlo con su consulta, lo que implica que un dato es leído sólo 1 vez desde memoria principal, y esta instrucción de lectura abastece a todos los hilos presentes en el núcleo. En cambio, en la versión MIC v1 cada hilo accede a una posición muy distante de la que accede el siguiente hilo (sizeof(double)*64 bytes), realizando una instrucción de lectura por cada hilo.

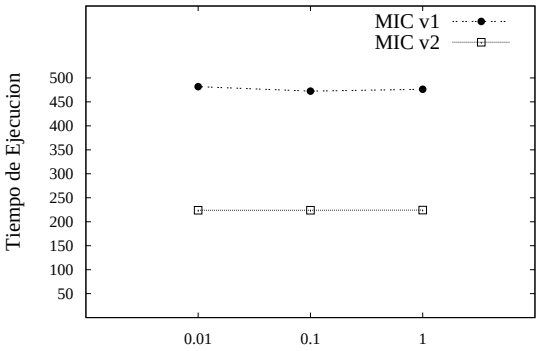
La Figura 4 muestra el speed-up de ambas versiones del algoritmo multi-core utilizando 16 núcleos, de la versión MIC v2 y también del índice *LC* utilizando 16 núcleos. Recordemos que el índice *LC* agrupa los elementos en clusters, que posteriormente intenta descartar. Este índice ha mostrado buenos resultados sobre diferente tipo de plataformas paralelas ([18], [21]). Esta figura muestra una gran eficiencia con radio pequeño por parte del índice *LC*, debido principalmente a su buena capacidad de descarte de elementos. Pero, al incrementarse el radio, disminuye su capacidad de descarte, suficiente como para que la versión MIC v2 se equipare en rendimiento. Esto



(a) BD de 500,000 elementos.



(b) BD de 1,000,000 elementos.



(c) BD de 1,700,000 elementos.

Figura 3. Tiempo de ejecución sobre distintas Bases de Datos.

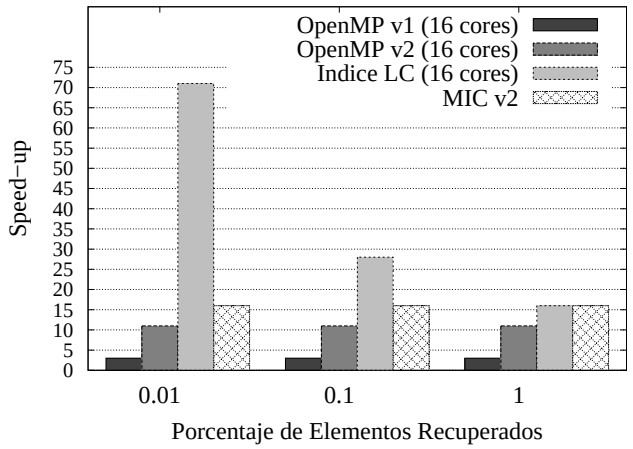


Figura 4. Speed-up sobre el algoritmo secuencial de fuerza bruta.

muestra el buen uso de la capacidad de cómputo y caché en la Xeon Phi por parte de la versión MIC v2, que accede a cada elemento de la base de datos.

V. Conclusiones

En el presente trabajo se han propuesto y comparado diferentes alternativas para procesar consultas por rango utilizando un coprocesador Intel Xeon Phi con algoritmos exhaustivos. El utilizar algoritmos exhaustivos para resolver consultas por rango otorga la ventaja de no estar limitado a un tipo de base de datos, como suele ocurrir al resolver consultas de este tipo, limitándose a espacios métricos, por lo tanto, nuestras soluciones pueden ser aplicadas a un número mayor de bases de datos, y de distinto tipo.

El algoritmo MIC v2 es el que mejor rendimiento mostró, debido principalmente a su buen uso de caché. Esto último ocurre porque hilos que se ejecutan en un mismo núcleo acceden a las mismas posiciones de memoria de la Xeon Phi, por lo que se minimiza la cantidad de accesos a memoria principal del coprocesador, que es una operación muy costosa en ciclos de reloj. También, con un rango de búsqueda grande, se consigue un rendimiento similar a la versión multi-core del índice *LC* (con 16 núcleos), que ha mostrado buen rendimiento sobre distintas plataformas paralelas.

Referencias

[1] "GPU Computing. *http://www.nvidia.com/object/what-is-gpu-computing.html*."

[2] W. mei Hwu, *Programming Massively Parallel Processors, Second Edition: A Hands-on Approach*. Morgan Kaufmann, 2012.

[3] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, *High-Performance Computing on the Intel® Xeon Phi(TM): How to Fully Exploit MIC Architectures*. Springer, 2014.

[4] "OpenMP Architecture Review Board, "OpenMP Application Program Interface, Version 4.0", July 2013."

[5] B. Chapman, G. Jost, and R. V. D. Pas, *Using OpenMP: portable shared memory parallel programming*. The MIT Press, 2008.

[6] U. Drepper and I. Molnar, "The native posix thread library for linux," Tech. Rep., February 2003.

[7] "MPI Forum, "MPI: A message-passing interface standard", version 3.0. september 2012."

[8] "PRACE (Partnership for Advanced Computing in Europe). Best Practice Guide - Intel Xeon Phi: http://www.prace-ri.eu/best-practice-guide-intel-xeon-phi-html?lang=en."

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

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[11] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," *Pattern Recognition Letters*, vol. 26, no. 9, pp. 1363–1376, 2005.

[12] G. Navarro and R. Uribe-Paredes, "Fully dynamic metric access methods based on hyperplane partitioning," *Information Systems*, vol. 36, no. 4, pp. 734 – 747, 2011.

[13] L. Britos, A. Printista, and N. Reyes, "Dsacl+-tree: A dynamic data structure for similarity search in secondary memory," in *5th International Conference on Similarity Search and Applications (SISAP 2012)*, vol. 7404. Canada: Springer, LNCS, 2012, pp. 116–131.

[14] D. Novak, M. Batko, and P. Zezula, "Metric index: An efficient and scalable solution for precise and approximate similarity search," *Information Systems*, vol. 36, no. 4, pp. 721–733, 2011.

[15] E. Tellez, E. Chávez, and K. Figueroa, "Polyphasic metric index: Reaching the practical limits of proximity searching," in *5th International Conference on Similarity Search and Applications (SISAP 2012)*, vol. 7404. Canada: Springer, LNCS, 2012, pp. 54–69.

[16] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[17] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti, "Cophir: a test collection for content-based image retrieval," *CoRR*, vol. abs/0905.4627, 2009. [Online]. Available: http://cophir.isti.cnr.it

[18] R. J. Barrientos, J. I. Gómez, C. Tenllado, M. P. Matias, and M. Marin, "Range query processing on single and multi GPU environments," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2656 – 2668, 2013.

[19] R. Barrientos, J. Gómez, C. Tenllado, M. Prieto, and P. Zezula, "Multi-level clustering on metric spaces using a multi-gpu platform," in *19th International European Conference on Parallel and Distributed Computing (Euro-Par 2013)*, ser. LNCS, vol. 8097. Aachen, Germany: Springer, August 2013, pp. 216–228.

[20] "MUFIN Web Site: Multi-feature Indexing Network. http://mufin.fi.muni.cz/imgsearch/similar."

[21] V. Gil-Costa, R. J. Barrientos, M. Marin, and C. Bonacic, "Scheduling metric-space queries processing on multi-core processors," in *18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010)*. Pisa, Italy: IEEE Computer Society, February 2010, pp. 187–194.