Operaciones en Base de Datos Métricas y Modelo CPU-GPU

Mariela Lopresti, Fabiana Piccoli, Nora Reyes
LIDIC- Universidad Nacional de San Luis
Ejército de los Andes 950
Tel: 02652 420823, San Luis, Argentina
{omlopres,mpiccoli,nreyes}@unsl.edu.ar

1. Resumen

En la actualidad, debido a la evolución de las tecnologías de información y comunicación, tenemos la posibilidad de almacenar distintos tipos de información. Podemos encontrar, por ejemplo en la Web, docenas de billones de documentos y cientos de millones de imágenes y otros tipos de datos tales como fotografías, audio y video. Esta nueva colección de datos no estructurados admite como modelo a las bases de datos métricas. Ante la gran cantidad de información, para poder recuperar estos datos en forma rápida y eficiente han surgido distintas alternativas de optimización. Una de ellas son estructuras de indexación y algoritmos de búsquedas y otra es la optimización a través de la aplicación de técnicas de computación de alto desempeño.

Los sistemas diseñados para resolver problemas específicos como los procesadores gráficos (GPU), han comenzado a ser de gran interés para desarrollar problemas de computación de próposito general ya que proveen un bandwith de memoria extremadamente alto y gran poder computacional necesarios para computación de alta performance y a un bajo costo.

La línea de investigación que se propone seguir pretende evaluar la factibilidad de utilizar la GPU como computadora masivamente paralela para obtener soluciones de alto desempeño en base de datos métricas. Entre las operaciones de interés se encuentran las consultas.

2. Contexto

Esta propuesta de trabajo se lleva a cabo dentro de la líneas de Investigación " Computación de Alto Desempeño" y " Recuperación de Datos e Información Multimedia" del proyecto "Nuevas Tecnologías para el Tratamiento Integral de Datos Multimedia". Este proyecto es desarrollado en el ámbito del Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC) de la Universidad Nacional de San Luis y participa de Programa de Incentivos.

3. Introducción

Muchas aplicaciones computacionales necesitan buscar eficientemente información en bases de datos. Actualmente ha surgido la necesidad de crear bases de datos de información no estructurada, como por ejemplo imágenes, texto, sonido y video. Para realizar consultas a estas bases de datos se han creado nuevos modelos y algoritmos de búsqueda más generales que aquéllos que buscan sólo a partir de una clave en una base de datos tradicional con información estructurada [4].

Las consultas que pueden ser de interés en bases de datos no estructuradas se denominan búsquedas por similitud o búsquedas por proximidad. La búsqueda por proximidad es la búsqueda en Bases de Datos de elementos que sean similares o cercanos a un elemento consultado. La similitud es modelada con la función de distancia, la cual satisface entre otras propiedades la desigualdad triangular y el conjunto de elementos u objetos sobre los que se define es llamado *espacio métrico* [4].

Introducimos ahora la notación básica para el problema de satisfacer búsquedas por proximidad. El conjunto X denota el universo de objetos válidos. Un subconjunto finito de él U, de tamaño n, es el conjunto de objetos o base de datos en donde se realizan las búsquedas. La función $d : X * X \longrightarrow \mathbb{R}^+$ denota la medida de distancia entre los objetos. Esta función de distancia debe cumplir las propiedades de positividad $\forall x, y \in X, d(x, y) \geq 0$, simetría $(\forall x, y \in X, d(x, y) = d(y, x))$, reflexividad $(\forall x \in X, d(x, x) = 0)$, desigualdad triangular $\forall x, y, z \in X, d(x, y) \leq d(x, y) + (d(y, z)$ y en la mayoría de los casos la positividad es estricta $(\forall x, y \in X, x \neq y \Rightarrow d(x, y) > 0)$.

Dada una base de datos finita $U \subseteq X$, la cual es un subconjunto del universo de objetos que pueden ser procesados y una consulta q, el objetivo es recuperar todos los objetos similares que se encuentren en la base de datos [4, 21, 20]. Mientras más chica sea la distancia entre los objetos más similares son dichos objetos.

Existen tres tipos básicos de consultas por proximidad en espacios métricos:

- Consulta por rango$(q, r)_d$: recupera los elementos

que están a lo más a distancia r de q.

- Consulta por vecino más cercano $NN(q)$: recupera el vecino más cercano a q en U.
- Consulta por k vecinos más cercanos $NN_k(q)$: recupera los k vecinos más cercanos a q en U.

Si tenemos un base de datos de cardinalidad n, todas estas consultas pueden ser resueltas ejecutando n evaluaciones de distancia. Generalmente el número de evaluaciones de distancias ejecutadas es la medida de complejidad de los algoritmos. El desafío es diseñar un algortimo de indexación eficiente que reduzca el número de las evaluaciones de distancia [4, 21, 20, 15, 19, 3, 14, 6].

Los algoritmos de indexación permiten construir a priori un índice, una estructura de datos capaz de ahorrar computaciones de distancias al responder consultas por proximidad.

En general todos los algoritmos de indexación particionan el conjunto X en subconjuntos X_i. Se construye un índice para permitir determinar una lista de subconjuntos X_i de candidatos potenciales a contener objetos relevantes para la consulta. Durante la consulta, se busca en el índice para encontrar los subconjuntos relevantes y luego se inspeccionan exhaustivamente todos estos conjuntos. Todas estas estructuras trabajan sobre la base de descartar elementos usando la desigualdad triangular [4]. Existen distintos métodos de indexación, entre ellos se encuentran el método por *pivotes* [21, 20, 4] y el de los *permutantes* [15].

Dadas las ventajas de poder computacional, bajo costo, continua evolución, bandwidth de memoria, flexibilidad y programabilidad de la GPU, y a pesar de la limitación y dificultad para resolver cualquier tipo de aplicaciones siguiendo un modelo de programación no usual [1, 13, 11], se intenta aprovechar la gran potencia de cálculo de las GPU para aplicaciones no relacionadas con los gráficos, en lo que se conoce como GPGPU (General Purpose GPU) [16]. En nuestro caso, considerando que las búsquedas sobre grandes espacios métricos son de gran complejidad hemos considerado implementar estas búsquedas en forma paralela.

Por muchos años la GPU fue utilizada exclusivamente para acelerar el cálculo de ciertas aplicaciones relacionadas directamente con el procesamiento de imágenes, en aplicaciones como videojuegos o 3D interactivas, por ejemplo. Su buen desempeño en este ámbito, junto a su constante y rápida evolución (comparada con los microprocesadores de propósito general) y un número de instrucciones menor [10, 13], ha permitido desarrollar un modelo de supercómputo casero en donde, con menos recursos económicos de los requeridos para comprar una PC, es posible resolver cierto tipo de problemas aplicando un modelo de paralelismo masivo sobre una arquitectura de procesadores con varios núcleos, memoria compartida y soporte multihilos.

Existen alternativas para procesamiento en GPU, la más ampliamente utilizada es la tarjeta Nvidia, para la cual se ha desarrollado un kit de programación en C, con un modelo de comunicación de datos y de control de hilos proporcionado por un driver, el cual provee una interfaz GPU-CPU [9]. Este ambiente de desarrollo llamado Compute Unified Device Architecture (CUDA) ha sido diseñado para simplificar el trabajo de sincronización de hilos y la comunicación con la GPU [5, 12] propone un modelo de programación SIMD (Simple Instrucción, Múltiples Datos) con funcionalidades de procesamiento de vector.

4. Líneas de Investigación y Desarrollo

Utilizar arquitecturas dedicadas, como la GPU, para resolver computacionalmente problemas de naturaleza distinta a la de ellas, implica plantear soluciones paralelas al problema enunciado teniendo en cuenta el modelo de programación propio de la interface.

Las líneas de investigación y desarrollo que actualmente se siguen se pueden dividir en dos áreas, la relacionada a las técnicas de indexación y resolución de consultas en espacios métricos, y la referida a otras operaciones para bases de datos sobre espacios métricos. A continuación se describen cada una de ellas.

4.1. Técnicas de Indexación

Como se mencionó previamente, para resolver las consultas a bases de datos métricas de manera eficiente se debe construir un índice sobre la base de datos. La idea en este caso es acelerar las búsquedas a través de almacenar algunas distancias y utilizar la desigualdad triangular para ahorrarnos, durante las consultas, algunos cálculos de distancia y también a través del uso de técnicas de computación de alto desempeño sobre el proceso.

Consultas: Existen distintos tipos de consultas para bases de datos, que han sido adaptadas para espacios métricos. Las más conocidas son:

- $(q, r)_d$: recupera los elementos que están dentro de una distancia r de q ($(q, r)_d = \{x \in U, d(q, x) \leq r\}$). Una consulta por rango puede ser resuelta seleccionando cada elemento de la base de datos, uno por uno y calcular su distancia a la consulta.
- Consulta por vecinos más cercanos $NN(q)$: recupera los vecinos más cercanos a q en U. Las consultas para encontrar los vecinos más cercanos pueden ser solucionadas con los algoritmos que resuelven las consultas por rango. Esto se logra mediante distintas técnicas:
  - *De Radio Incremental*: El algoritmo para la búsqueda de los vecinos más cercanos con

esta técnica está basado en usar un algoritmo de rango como sigue: La búsqueda de q con radio fijo $r = a^i\varepsilon, (a > 1)$, inicia con $i = 0$ y se incrementa i hasta que al menos el número deseado de elementos (1 o *(k)*) estén dentro del radio de consulta $r = a^i\varepsilon$. Después hay un refinamiento del radio entre $r = a^i\varepsilon$ y $r = a^{i-1}\varepsilon$. Dado que la complejidad de las búsquedas por rango crece abruptamente con el radio de búsqueda, el costo de este método es semejante al costo de una búsqueda por rango con el radio apropiado (cuando se conoce originalmente). El incremento de a puede ser muy pequeño ($a \to 1$) para evitar que el radio de consulta sea mucho más grande de lo necesario.

- *Backtracking con Reducción del Radio*: Una búsqueda más elaborada de los vecinos más cercanos con esta técnica (en principio para $k = 1$) consiste en iniciar la búsqueda con $r^* = \infty$. Cada vez que la consulta q es comparada contra un elemento p en la base de datos, se actualiza el radio de búsqueda $r^* \leftarrow \text{mín}(r^*, d(q, p))$ y continúa la búsqueda con el radio reducido. A medida que se encuentran elementos más cercanos a la consulta, el radio se reduce y se refleja en una búsqueda menos difícil. Por esta razón es importante tratar de encontrar rápidamente los elementos más cercanos.

- Consulta por k vecinos más cercanos $NN_k(q)$: Esta consulta recupera los k vecinos más cercanos a q en U. Resolver búsquedas de los k vecinos más cercanos es una extensión de alguno de los algoritmos descriptos anteriormente. Se mantienen los k elementos más cercanos conocidos en cada momento y el radio se reduce a la distancia máxima entre aquellos k elementos. Cada vez que se compara un elemento contra la consulta, éste podría reemplazar a algún elemento de los candidatos y por lo tanto debería actualizarse el radio de consulta.

Indices: En [4, 21, 20] aparecen numerosos métodos de indexación que pueden ser utilizados para resolver los tipos de consultas mencionados. En particular en [15] aparece una nueva técnica que permite resolver muy eficientemente las consultas, siempre y cuando se admita una respuesta no exacta. La búsqueda por similitud no exacta es razonable en muchas aplicaciones debido a que la extracción de características ya implica una aproximación a la realidad y por lo tanto una segunda aproximación durante las búsquedas es, en general, aceptable. Describimos a continuación brevemente la misma:

Permutantes: Los índices métricos existentes hacen uso de la desigualdad triangular para responder consultas de proximidad, ya sea partiendo el espacio en regiones compactas o utilizando distancias precalculadas a un conjunto distinguido de elementos.

En [15, 2] se presentó una nueva alternativa para resolver el problema, representando los elementos como permutaciones. La idea principal es seleccionar un conjunto de elementos $P \subset U$, donde $\mid P \mid = k$; a este conjunto se lo llama *permutantes*. En la indexación, cada elemento $u \in U$ ordena los elementos de P por distancia creciente a u. Nótese que cada elemento de la base de datos, una vez que ordena el conjunto P, podría distinguirse respecto a otro u' usando sólo ambos órdenes, lo que nos es claramente útil para acotar la búsqueda.

Al momento de responder una consulta q, también se genera un orden en el conjunto de permutantes P de acuerdo a su cercanía a q. De la diferencia entre el orden producido en P por q y por u, se puede inferir si u es menos relevante para q, y al menos estimar la semejanza entre u y q (sin posibilidad de descartarlo).

De la comparación experimental de esta técnica contra el estado del arte, en diversos espacios reales y sintéticos, se concluye que las permutaciones son mucho mejores predictores de proximidad que las técnicas hasta ahora usadas, sobre todo en dimensiones altas. Generalmente basta revisar una pequeña fracción de la base de datos para tener un alto porcentaje de la respuesta correcta.

4.2. Otras Operaciones

Algunas operaciones que podrían ser de interés en bases de datos métricas y que pueden ser optimizadas mediante la inclusión técnicas de programación de alto desempeño son: join por similitud y join por rango. Cada una

- Join por similitud Es el complemento necesario más importante para las búsquedas de vecinos más cercanos. No es muy utilizado ya que tiene una complejidad cuadrática para grandes colecciones de datos [8, 7]. Dado dos conjuntos de objetos, se debe encontrar pares de objetos (uno de cada conjunto) que satisfagan el mismo predicado de similitud [17, 18]. La comunidad de recuperación de información siempre ha considerado que los resultados de una búsqueda se muestran como un ranking en una lista de objetos. Dado una consulta algunos objetos son más importantes para la consulta que otros y los usuarios están interesados en los objetos más relevantes, es decir los que están en la parte más alta del ranking [4, 15, 2]. En [8] se propone extender la estructura de índice métrico D-index [6] y comparar dos algoritmos construídos para join por similitud para búsquedas en espacios métricos. Las implementaciones de los algoritmos estudiados, consulta por rango y overloading join (auto-join por similitud) [8], no logran eliminar la complejidad cuadrática de

los joins por similitud pero si logran una mejora significativa en la performance.

- Join por rango Es una variante del join por similitud. Dadas dos bases de datos $A, B \subset X$ y una distancia límite $r > 0$, se deben encontrar todos los pares de objetos a una distancia no mayor que r. Este join básicamente resuelve varias consultas por rango, donde las consultas provienen de un conjunto y los objetos relevantes para cada consulta provienen del otro conjunto. Por lo tanto, calcular el join por rango consiste en indexar un conjunto y luego solucionar consultas por rango para cada elemento del otro conjunto. Siguiendo este enfoque también se puede indexar ambos conjuntos para acelerar todo el proceso. En [17, 18], en cambio se ha propuesto indexar el join de ambos conjuntos, presentando una nueva propuesta para índice métrico que han denominado *list of twin clusters* (LTC). LTC ha sido utilizada como un índice para solucionar joins por similitud y también como un índice para solucionar otras primitivas básicas como *k-closest pair join*. Los experimentos realizados para la evaluación de LTC para verificar los join por similitud y sus variantes han mejorado sobres las alternativas básica de tiempo cuadrático y que LTC es competitiva respecto de LC clásicas (*list of clusters*) [3, 14] para resolver consultas por rango. Además se demostró que esta técnica tiene gran potencial para mejoras.

El común denominador de los líneas planteadas es el rápido crecimiento del volumen de datos y la necesidad de resolverlos en forma rápida, eficiente y precisa. Analizar las ventajas y desventajas de utilizar la tecnología de GPU y su modelo de programación asociado para implementar en forma paralela las estructuras planteadas para base de datos métricas constituye una innovadora línea de investigación.

Actualmente se está desarrollando la técnica de permutantes considerando las características del modelo CPU-GPU, lo cual implica la aplicación del modelo STMD (Single Threads Multiple Data), el costo de las transferancias entre CPU y GPU, y la jerarquía de memorias propia de la GPU .

5. Resultados y Objetivos

El principal aporte de esta línea de investigación es analizar la factibilidad de utilizar la arquitectura GPU y su modelo de programación en el desarrollo de soluciones de alto desempeño a problemas de propósito general, particularmente sobre bases de datos métricas, estableciendo los límites del modelo CPU-GPU para soluciones eficientes y eficaces de consultas y operaciones en ellos.

Los actuales desarrollos nos permitirán evaluar y comparar el rendimientos de las soluciones secuenciales y paralelas, no sólo respecto a la velocidad y consecuente aceleración de las soluciones, sino también respecto a la eficiencia de ambas implementaciones.

6. Formación de Recursos Humanos

Los resultados esperados respecto a la formación de recursos humanos son hasta el momento una tesis de maestría en desarrollo y dos trabajos de fin de carrera de la Licenciatura en Ciencias de la Computación.

Referencias

[1] Ian Buck. Gpu computing with nvidia cuda. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[2] E. Chávez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(9):1647–1658, 2008.

[3] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[5] Wei-Nien Chen and Hsueh-Ming Hang. H.264/avc motion estimation implmentation on compute unified device architecture (cuda). In *Multimedia and Expo, 2008 IEEE International Conference on*, 23 2008.

[6] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications (MTAP)*, 21(1):9–33, 2003.

[7] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. Similarity join in metric spaces. In *Proc. 25th European Conf. on IR Research (ECIR'03)*, LNCS 2633, pages 452–467, 2003.

[8] V. Dohnal, C. Gennaro, and P. Zezula. Similarity join in metric spaces using eD-index. In *Proc. 14th Intl. Conf. on Database and Expert Systems Applications (DEXA'03)*, LNCS 2736, pages 484–493, 2003.

[9] Mark Joselli, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, Aura Conci, Regina Leal-Toledo, Luis Valente, Bruno Feijó, Marcos d' Ornellas, and Cesar Pozzer. Automatic dynamic task distribution between cpu and gpu for real-time

systems. In *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*, pages 48–55, Washington, DC, USA, 2008. IEEE Computer Society.

[10] Michael D. Lieberman, Jagan Sankaranarayanan, and Hanan Samet. A fast similarity join algorithm using graphics processing units. In *ICDE*, pages 1111–1120. IEEE, 2008.

[11] Brandon Lloyd, Chas Boyd, and Naga K. Govindaraju. Fast computation of general fourier transforms on gpus. In *ICME*, pages 5–8. IEEE, 2008.

[12] D. Luebke. Cuda: Scalable parallel programming for high-performance scientific computing. In *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pages 836 –838, May 2008.

[13] David Luebke and Greg Humphreys. How gpus work. *Computer*, 40:96–100, February 2007.

[14] M. Mamede. Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. In *Proc. 20th Intl. Symp. on Computer and Information Sciences (ISCIS'05)*, LNCS 3733, pages 843–853, 2005.

[15] K. Figueroa Mora. *Indexación Efectiva de Espacios Métricos usando Permutaciones*. PhD thesis, Universidad de Chile, Santiago, Chile, 2007. Directores: Dr. G. Navarro y Dr. E. Chávez.

[16] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[17] Rodrigo Paredes and Nora Reyes. List of twin clusters: A data structure for similarity joins in metric spaces. In *Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008)*, pages 131–138, Washington, DC, USA, 2008. IEEE Computer Society.

[18] Rodrigo Paredes and Nora Reyes. Solving similarity joins and range queries in metric spaces with the list of twin clusters. *J. of Discrete Algorithms*, 7:18–35, March 2009.

[19] N. Reyes. Indices dinámicos para espacios métricos de alta dimensionalidad. Master's thesis, Universidad Nacional de San Luis, San Luis, Argentina, 2002. Director: Dr. G. Navarro.

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

[21] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. XVIII, 220 p., Hardcover ISBN: 0-387-29146-6.