Perfilado y Optimización de una Aplicación Java para el Modelado de Escenarios de Flujo Vehicular

Monetti, Julio[1]; Tinetti, Fernando[2]; León Oscar[1]
[1]Universidad Tecnológica Nacional, Facultad Regional Mendoza
Mendoza, Argentina
jmonetti,oleon{frm.utn.edu.ar}

[2]Universidad Nacional de La Plata, Facultad de Informática,
[2] Comisión de Investigaciones Científicas de la Prov. de Bs.As.,
Buenos Aires, Argentina
fernando@info.unlp.edu.ar

Resumen
El trabajo expone experiencias en el modelado de datos sobre la dinámica vehicular en zonas urbanas, con el fin de constituir modelos microscópicos de tránsito. A partir de datos obtenidos de dispositivos GPS, se han conformado mapas digitales que describen un grafo dirigido a través de la ciudad de Mendoza. Tales grafos, almacenados en una matriz de adyacencia de grandes dimensiones, representan un insumo de entrada para algoritmos que determinan caminos más cortos entre puntos y otro tipo de relaciones de interés.
El volumen de datos a procesar torna impracticable la aplicación de algoritmos cásicos para realizar operaciones de búsqueda. Se propone un cambio en los algoritmos que procesan las matrices de adyacencia desde dos perspectivas: 1) el procesamiento paralelo del producto matricial, 2) la reestructuración del modelo de datos que contiene el grafo dirigido.
Estos cambios permiten obtener mejoras considerables en el tiempo de ejecución de los algoritmos.

Palabras Clave
Flujo Vehicular, Modelos de Tránsito, Programación Paralela, Producto Matricial

Introducción

Los métodos para encontrar el camino más corto entre dos puntos sobre un grafo dirigido representan uno de los problemas clásicos de las ciencias computacionales [1][2]. Este problema es experimentado en el presente trabajo al intentar establecer el camino más favorable entre dos puntos de la ciudad. Los trabajos se encuadran en el marco de un proyecto para el modelado de flujo vehicular llevado a cabo por docentes y alumnos de la Universidad Tecnológica Nacional, por el cual se pretende crear micromodelos de tránsito para luego proveer simulaciones sobre diferentes escenarios.

La congestión vehicular [3][4] en ambientes urbanos representa un problema crítico hoy en día que no tiene solución a corto plazo en la mayoría de las ciudades. Se utilizan micromodelos de tránsito con el objeto de describir situaciones particulares y luego proponer mejoras en el tránsito y transporte a través de la ciudad. A partir del modelado generalmente surgen dificultades desde un punto de vista computacional, relacionadas con procesos complejos o gran cantidad de datos a procesar, que resultan difíciles para abordar con métodos numéricos clásicos.

Los estudios sobre tránsito y transporte, requieren del modelado y simulación de diferentes escenarios; lo cual requiere una correcta determinación de aquellas variables que permitan describir en forma clara y precisa situaciones actuales y proyecciones de la dinámica vehicular.

El proyecto, apunta a ofrecer soluciones alternativas sobre el procesamiento de datos provenientes de dispositivos GPS, atendiendo a la estructuras de datos y algoritmos aplicados sobre ellas, para el planteo de escenarios particulares de flujo

vehicular, determinación de indicadores de congestión vehicular, etc.
A continuación se mencionan los estudios de campo realizados para el posterior análisis de datos; observando lineamientos metodológicos en la captura masiva de datos a través de la creación de casos de simulación sobre zonas urbanas reducidas.
Se presenta la arquitectura computacional utilizada para la recolección de datos y procesamiento de los mismos.
Se señalan las características de los experimentos llevados a cabo, su preparación y ambiente de ejecución.
Luego, se propone un procesamiento de los datos con el objeto de describir las funciones necesarias para la identificación de caminos más favorables entre dos puntos determinados.
A partir de la caracterización de los algoritmos para la identificación de caminos, se propone un tipo de almacenamiento adecuado para reducir los altos tiempos de cómputo.
Finalmente se plantea la discusión sobre los resultados obtenidos y conclusiones del trabajo.

Estudios de Campo

Se han comenzado los estudios con el análisis sobre distintas metodologías de medición y estudio de la dinámica vehicular [5]. Esto se materializa a través de diferentes pruebas de medición bajo metodologías clásicas, a través del aforo vehicular en base a hojas de datos. Estas mediciones permiten en una primera instancia analizar las posibilidades de automatización de las muestras, el análisis de las características necesarias para los dispositivos de captura, etc.
La experiencia resulta de utilidad para observar tanto las principales variables de medición (por ejemplo: velocidad media por tramo), como así también situaciones de circulación irregulares o anómalas.
También durante esta etapa se realizan estudios acerca de las herramientas de modelado y simulación urbana existentes en el mercado. Este estudio supone observar y documentar el uso de software destinado al modelado vehicular a través de micro y macro modelos para afrontar estudios particulares de congestión vehicular.
Como resultado de esta investigación preliminar, se destaca que los productos relevados no satisfacen las expectativas de investigación propuestas, debido a la difícil manipulación del volumen de datos con el que se cuenta.

Arquitectura Computacional

El diseño de la arquitectura computacional para la simulación de escenarios comprende tanto la infraestructura física (*hardware*) como los componentes de software necesarios para el procesamiento de los datos obtenidos en bruto.
Para la prueba de los algoritmos se utiliza una arquitectura superescalar [6], cuya característica principal es la ejecución de más de una instrucción por ciclo de reloj, y en forma independiente [7]. Esto permite que la arquitectura haga uso de paralelismo de instrucciones y paralelismo de flujo. Este último gracias a la estructura de *pipeline*.
En una estructura superescalar con *pipeline* se deben tener en cuenta diferentes etapas en el tratamiento de las instrucciones que están siendo ejecutadas: lectura, decodificación, lanzamiento, ejecución, escritura y finalización. Estas etapas de ejecución manifiestan el paralelismo implementado por el procesador en un bajo nivel, no perceptible directamente por el programador que hace uso de lenguajes de alto nivel. Este paralelismo transparente, implementado por las capas bajas de la arquitectura, luego es complementado por la aplicación de paralelismo a nivel de hilos de ejecución, administrados conjuntamente por el programador y el sistema operativo.
Se utiliza para el conjunto de experimentos llevados a cabo un procesador *Intel Core i5 750* con una frecuencia de 2.67Ghz, línea *Lynnfield* (Una descripción de este procesador se encuentra en [7]). El conjunto de instrucciones con el que trabaja dicho procesador son MMX, SSE, SSE2,

SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x.

Para la obtención de datos se utiliza un dispositivo GPS *GARMIN Etrex Vista HCx* [8], el cual resulta adecuado para la captura de los datos requeridos, los cuales se ajustan tanto a las variables de medición para el posterior modelado, como así también al tiempo mínimo requerido entre lecturas. El dispositivo permite tomar trazas cada un segundo, lo que resulta conveniente para obtener múltiples muestras en un tramo reducido.

Para la adquisición masiva de datos también se analiza la aplicación de *Smartphones* por su grado de generalidad y acceso, encontrándose que los mismos son extremadamente dependientes del software que poseen, no encontrando estándares que permitan una obtención de datos homogéneamente confiable.

Si bien el sistema de información planteado no cuenta con las características de un sistema de procesamiento en tiempo de real, el mismo está orientado a automatizar la captura del tren de datos provenientes del dispositivo GPS, de tal forma que resulte favorable la generación de información estadística a partir de la estructura de datos propuesta.

Desde el punto de vista del software, se analizan diferentes algoritmos y metodologías de cálculo algebraico de cara a los requerimientos de procesamiento, para asegurar una calidad aceptable en tiempos de ejecución y almacenamiento requerido. Se presenta el inconveniente de un procesamiento masivo de grandes volúmenes de datos, con lo cual se prevé el diseño de algoritmos con características concurrentes para ser ejecutados sobre arquitecturas computacionales paralelas, a fin de poder alcanzar tiempos de respuesta aceptables.

El producto computacional obtenido consiste en un conjunto de algoritmos, bases de datos normalizadas y metodologías de procesamiento, que sirva como insumo para ser utilizado en futuros proyectos de modelado matemático de situaciones reales de tránsito. Se espera que los principales aportes sean: 1) Una metodología de procesamiento masivo de datos de la dinámica vehicular, lo cual permita obtener información útil para crear nuevos modelos urbanos. 2) Un modelo de datos "abierto" que permita ser accedido por diferentes sistemas de información, asegurando así una fácil integración tendiente a resolver problemas particulares, y 3) Información que resulte útil para la posterior confección de modelos matemáticos/estadísticos que describan el comportamiento vehicular en las zonas de estudio.

Adquisición de Datos

El almacenamiento principal se encuentra en una primera instancia sobre un conjunto de ficheros de texto obtenidos del dispositivo GPS. Cada fichero representa una circulación particular del vehículo de prueba; luego, el conjunto de ficheros es introducido en la base de datos con el objeto de generar un único dominio de datos del cual se pueda generar información estadística.

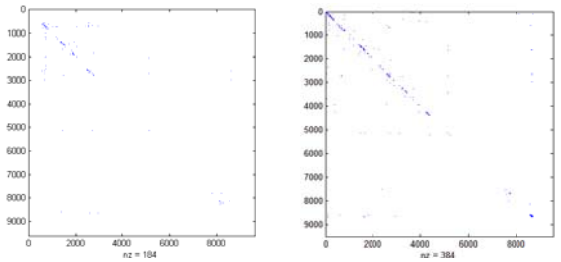
Se ha propuesto una metodología estándar de clasificación y nominación de los ficheros para mantener un orden de acuerdo al vehículo de prueba que toma la muestra, y otras variables que permitan describir el entorno en el cual se tomo la misma, por ej: *Característica del Vehículo Testigo, Día de la Semana / Fecha, Hora, Datos climáticos,* etc.

Dada la necesidad de almacenamiento masivo, y con el objeto de mantener reunidas la totalidad de las muestras obtenidas del conteo vehicular, se requiere el desarrollo de un modelo de datos eficiente, para ser implementado con un motor de base de datos que resulte adecuado para la gestión. La confección de la base de datos consta de dos etapas: La base de datos con información estática comprende la georeferenciación de puntos de interés a lo largo de las zonas de estudio. Estos puntos luego conforman una malla interconectada o grafo dirigido que establece la circulación de cada tramo (de

esquina a esquina). La información estática está conformada por tablas que contienen básicamente la información sobre esquinas, tramos de calles y zonas de estudio; sus nombres y datos principales para identificarlas. La base de datos dinámica contiene información recolectada a través de la circulación del auto flotante. La información obtenida se puede caracterizar en base a:

1. Información sobre el paso de un tramo a otro, lo que permite establecer sentidos de circulación.
2. Información sobre la velocidad puntual en coordenadas específicas de cada tramo.

Se obtienen permanentemente datos a través del dispositivo GPS, los cuales alimentan la base de datos de esquinas y tramos. En la figura 1 se observan dos momentos del estado de la base de datos: (a) y (b).



(a) (b)

Figura 1. *La Matriz de Adyacencia presenta elementos que determinan el paso de un tramo x a otro y. (a) y (b) son dos momentos del estado de la base de datos. La segunda matriz presenta mayor cantidad de entradas.*

A medida que se obtienen más trazas a través de la circulación libre del vehículo a través de la ciudad, se registran más datos sobre la relación de paso entre una esquina y otra. Es decir que un coeficiente en la matriz de adyacencia $A_{x,y}=1$ corresponde a una relación *paso(tramo1=x,tramo2=y)*. A simple vista se observa mayor densidad en la matriz de adyacencia (b). (Se destaca que la futura matriz de adyacencia que contenga la totalidad de relaciones posibles, contará con un gran grado de dispersión también).

Se visualiza en la figura que la densidad de la matriz se centra a lo largo de la diagonal principal; situación ventajosa al proponer mejores métodos de almacenamiento y operar con la matriz a través de métodos numéricos. Para la aplicación de los algoritmos que establecen el camino más favorable se toma una zona reducida, con el objeto de observar el funcionamiento del sistema (figura 2).

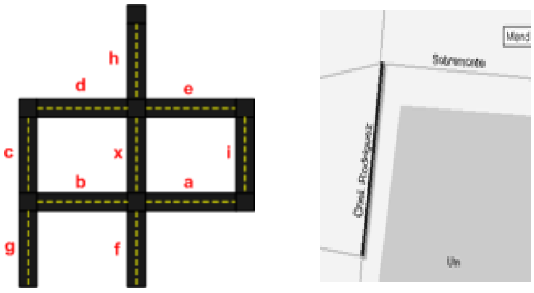
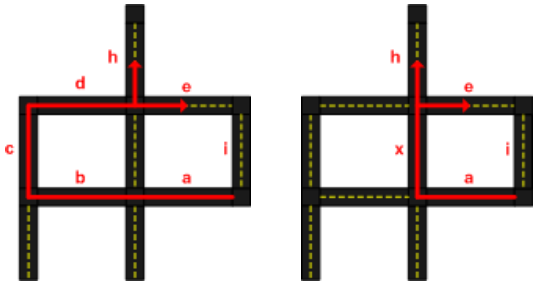


Figura 2*. Zona de estudio con una cantidad de tramos acotada. Corresponde al análisis de un tramo particular: x.*

La figura muestra gráficamente un tramo (*x*), que resulta de particular interés para describir un flujo de vehículos que transita de una zona de la ciudad a otra.
La matriz de adyacencia contiene información sobre el paso entre los diferentes tramos como pares ordenados. Para el ejemplo de la Figura 2: *(a,x),(a,b), (b,c), (c,d), (d,e), (d,h), (x,e),(x,h) (f,x), (f,b), (g,c), (e,i).*



(a) (b)

Figura 3*. Ejemplo de una zona de estudio, donde el tramo x presenta gran densidad vehicular en la hora de estudio.*

Luego, el sistema propuesto debe proveer información sobre el camino más favorable entre dos puntos. La figura 3 muestra un caso típico, en donde en hora de circulación pico se encuentra más favorable el camino más largo frente al más corto. En la figura se muestran dos caminos alternativos para acceder desde el tramo *a* al *e*: *(a,b,c,d,e)* o *(a,x,e)*.

El sistema de información permite llegar a la conclusión sobre el camino más favorable en base a información estadística previamente almacenada, la cual contiene entre otros datos, velocidades puntuales y velocidad promedio de cada tramo.

Esta información es obtenida a través de la aplicación de operaciones algebraicas fundamentales sobre la matriz de adyacencia.

Esto indica que al realizar un análisis y procesamiento global sobre el grafo completo se incurre en tiempos de ejecución que tornan inaplicable los algoritmos clásicos de búsqueda en grafos.

Preparación de Experimentos

El programa propuesto es parte de una librería de funciones provistas en Java, con lo cual es ejecutado en el entorno de una máquina virtual Java [9].

La matriz de adyacencia que describe las zonas de estudio cuenta con una cantidad superior a las 9.000 relaciones. Luego, se configura dicha matriz con un total de 10.000x10.000 elementos; requiriendo un espacio de almacenamiento de 381,469Mb de acuerdo al tipo de datos utilizado (entero *int* del lenguaje Java).

Haciendo un uso razonable de las diferentes estructuras de datos utilizadas en la aplicación, y luego del perfilamiento de la misma se encuentra que el máximo *heap* usado es de 776,727Mb.

Se verifica que la memoria utilizada por la totalidad de las estructuras no sature la memoria disponible en la máquina virtual.

Se corrobora también que aplicaciones y servicios ejecutando en la computadora de prueba no interfieran con los tiempos de medición [10].

Procesamiento de Datos

Para la validación del sistema de información, y para una primera experiencia de calibración se modela un caso de estudio.

Como se especificó anteriormente, el sistema tiene por objeto establecer caminos más favorables entre dos puntos, basados en información estadística sobre la circulación previa del vehículo de prueba sobre los tramos de estudio.

Como ejemplo se presenta un caso de estudio reducido (ver figura 2) para obtener el camino más favorable entre el tramo *a* y el tramo *e*. Esta circulación es a través del tramo crítico *x*. Dicho tramo, junto con los tramos próximos, representa una zona de congestión vehicular típica en la ciudad de Mendoza, por lo tanto resulta de interés establecer el mejor camino *camino(a,e)*: *(a,x,e)* o *(a,b,c,d,e)*.

En primer lugar se registran datos sobre el flujo vehicular en forma manual de acuerdo a lo establecido por pautas preestablecidas por la Ingeniería de Tránsito.

Este aforo manual permite confirmar la relación inversamente proporcional entre la *velocidad promedio* del tramo con la *densidad vehicular* del mismo.

En una segunda instancia se realizan mediciones de acuerdo a la metodología del auto flotante, con el objeto de registrar velocidades medias a lo largo del tramo crítico.

El tramo *x* posee 94 metros de esquina a esquina, y la dirección de circulación es sur-norte. Con el objeto de mejorar la información provista, se recaban datos sobre velocidades puntuales en subtramos de *x*, de aproximadamente 10 metros cada uno. Ya que el dispositivo GPS permite recolectar datos cada 1 segundo, cada tramo es dividido en 10 subtramos para obtener información particular en cada porción.

La circulación del vehículo a través del tramo de estudio describe un tren de datos, donde cada muestra es obtenida cada un segundo, y corresponde a una velocidad puntual (ver figura 4). La coordenada *(latitud, longitud)* obtenida en la medición

es convertida por una función (método de una clase utilitaria Java) en un escalar que establece la distancia en metros desde la esquina hacia dicha coordenada. Esto permite luego establecer información circunscrita a los subtramos de estudio (esta información no es tenida en cuenta para el presente trabajo).
Con el conjunto de datos recolectados es posible luego establecer una clara relación entre la velocidad promedio y el grado de congestión dado por la densidad vehicular en el tramo.
Finalmente, se almacena en la base de datos información resumida sobre las características de circulación a través del tramo *x* en el horario de prueba. Esta información conforma luego el *costo* de transitar por dicho tramo en un horario específico.

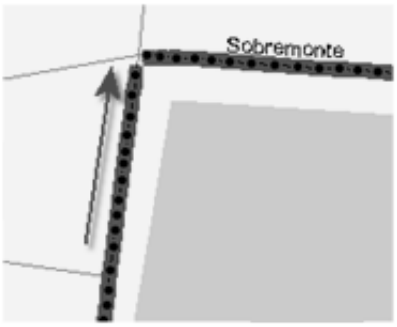


Figura 4*. Tren de datos obtenidos con el dispositivo GPS proyectados sobre el mapa del tramo de estudio. Cada punto negro corresponde a una lectura de datos.*

La información obtenida por la circulación del vehículo es contrastada con las coordenadas del tramo previamente georeferenciado. El paso consecutivo del vehículo de prueba por el mismo tramo genera numerosas muestras para cada subtramo, lo que permite ajustar los valores estadísticamente.

Búsqueda del Camino más Favorable

Para dicho procesamiento se realiza la potenciación de la matriz de adyacencia *A* con el objeto de obtener los caminos de paso 2 en A^2, los de paso 3 en A^3, y así sucesivamente [11]. En la matriz producto se encuentran los costos de circulación de un punto *i* a otro *j*. La ecuación (1) representa en particular un elemento *(i,j)* de la matriz de adyacencia de paso 2

$$A^2_{i,j} = C_{i,j} \qquad (1)$$

donde *Ci,j* constituye el costo de circular de *i* a *j*. La potencia 2 de la matriz *A* especifica que sus coeficientes representan pasos de longitud 2. Como se especificó anteriormente, los costos de tránsito sobre un tramo en particular son establecidos en base a información estadística previa, y se refieren particularmente a la velocidad promedio del tramo durante el horario de estudio.
Al realizar la potenciación de la matriz *A* (como productos matriciales sucesivos) se encuentran tiempos de procesamientos excesivamente altos para la calidad de ejecución requerida.
En primer lugar se optimiza el algoritmo utilizando multiplicación matricial por bloques [12]. Para una cantidad de dos bloques por matriz se tiene

A	B	*	E	F	=	I	J
C	D		G	H		K	L

Figura 5. *Producto matricial por bloques. Cada matriz es descompuesta en 4 submatrices o bloques.*

La matriz producto resulta:

$$\begin{aligned} I &= A*E + B*G \\ J &= A*F + B*H \\ K &= C*E + D*G \\ L &= C*F + D*H \end{aligned} \qquad (2)$$

La figura 5 y relación (2) muestran en forma genérica la multiplicación de dos matrices particionada en dos bloques. Para este trabajo se declara únicamente una matriz, la cual se multiplica por sí misma, puesto que se realiza la potenciación.
Se observan dos razones principales por las cuales aplicar el producto como múltiples subproductos por bloques: 1) Disponer los

subproductos en hilos de ejecución independientes, los cuales puedan ser distribuidos fácilmente bajo un esquema paralelo de procesamiento. 2) Aprovechar las capacidades que presenta la arquitectura del procesador utilizado, al disponer de una memoria caché nivel 3 compartida, de gran capacidad (8Mb) y baja latencia.
Con respecto a esta segunda razón se provee una dimensión de bloques específica, de modo tal que los mismos puedan ser contenidos totalmente en memoria cache L3. Esto permite que cada bloque pueda ser reutilizado sin la necesidad de requerir sus elementos a memoria principal y ahorrar así tiempos de acceso a la misma.
La metodología de multiplicación utilizada para una cantidad de bloques arbitraria es la misma que para (2), generalizando la expresión como:

$$C_{pq} = \sum_{r=1}^{M} A_{pr} * B_{rq} \qquad (3)$$

Donde A_{pr} corresponde a un al bloque de coordenada *(p,r)* en la matriz *A* y B_{rq} un bloque de coordenada *(r,q)* en la matriz *B*. Las condiciones para la multiplicación por bloque se mantienen con respecto a la multiplicación de matrices tradicional (cantidad de columnas de *A* igual a la cantidad de filas de *B*).
La relación (3) sugiere que el producto puede ser calculado en forma recursiva, donde cada recursión resuelve la multiplicación de bloques cada vez más pequeños consiguiendo resultados intermedios, hasta alcanzar una profundidad determinada o finalmente la multiplicación escalar.
En este trabajo no se evalúa esta posibilidad, sino que se propone esta multiplicación con el objeto de paralelizar la multiplicación de diferentes bloques a través de diferentes computadoras (por ej. en un *cluster*) o diferentes núcleos en una arquitectura de procesadores *multicore* [13] [14]. Se toma esta segunda arquitectura para la paralelización del proceso.

Al realizar el perfilado del algoritmo de multiplicación clásico, se observa que la máquina virtual utiliza un único núcleo del procesador (ver figura 6). Se utiliza la aplicación *top* del sistema operativo Linux para verificar el estado de la ejecución del algoritmo.

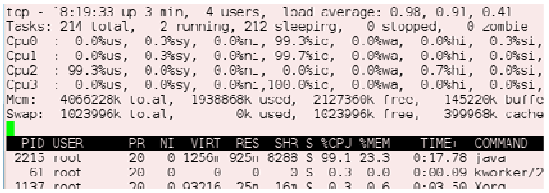


Figura 6. Se encuentra operativo solamente el núcleo Cpu2 durante el cálculo del producto matricial.

Entre otros datos relevados a través de *top* se encuentra la cantidad de memoria de intercambio utilizada (*swap*), y el porcentaje de ocupación de cada núcleo.
Con el objeto de que la máquina virtual ocupe aquellos núcleos ociosos hasta el momento, se dispone cada subproducto sobre un hilo de Java [15] manteniendo así independencia en los cálculos de cada subproducto. Esto permite luego distribuir la carga de trabajo a lo largo de diferentes núcleos (ver figura 7).
Luego, es posible establecer diferentes estrategias de planificación de hilos, teniendo en cuenta tanto aquellas características provistas por el lenguaje, como así también el mapeo de hilos Java sobre hilos del sistema operativo [16] [17].

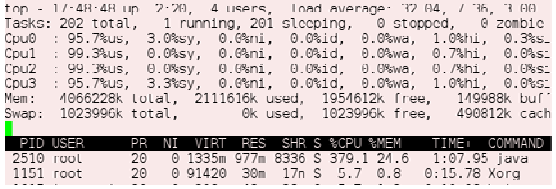


Figura 7. Los cuatro núcleos (Cpu0, Cpu1, Cpu2 y Cpu3) se encuentran operativos en el cálculo del producto matricial.

Una conclusión preliminar establece que las mejoras algorítmicas llevadas a cabo a través del mejor aprovechamiento de la memoria caché [18], y la posterior paralelización del proceso, provee tiempos

sensiblemente mejores. Pero al tratarse este producto matricial de un cálculo repetitivo en el proceso de simulación y búsqueda de caminos más favorables, se concluye que se deben proveer cambios más radicales, y más allá de la mejora algorítmica.
Luego, el modelo de datos es la próxima variable a analizar.
Se realiza la modificación del modelo de datos y se utiliza un modelo de almacenamiento para matrices dispersas [19].

Almacenamiento de la Matriz Dispersa

Para los datos almacenados hasta el momento, la capacidad de ocupación de la matriz de adyacencia representa el $9{,}73*10^{-4}$% del disponible. Luego, el formato coordenado representa una alternativa adecuada para el almacenamiento de dicha matriz. Este formato se basa en la creación de una lista de elementos no nulos. Junto con dichos elementos se almacena la fila y columna correspondiente al mismo.
La figura 8 muestra un ejemplo sobre el almacenamiento coordenado de una matriz de adyacencia con solo tres elementos no nulos:

{ (x=1,y=1,val=1),
(x=10,y=10,val=1),
(x=100,y=1,val=1) }

Fila	Columna	Valor
1	2	1
10	10	1
100	1	1

Figura 8. La lista de registros contiene los datos de tres coeficientes no nulos de la matriz rala: (1,2), (10,10) y (100,1).

El formato coordenado es uno de los más simples de los propuestos para el almacenamiento de una matriz dispersa.

Resultados

Las pruebas realizadas corresponden a un número considerable de ejecuciones, en las cuales no se encontraron desviaciones considerables respecto de los tiempos medios.
En primer lugar se registran tiempos de ejecución para la multiplicación por bloques, considerando diferentes tamaños de bloque. La tabla 1 muestra la relación existente entre la cantidad de elementos por bloque, el tamaño en *megabytes* requerido por el mismo en memoria, el porcentual de fallos de caché frente a referencias al mismo, y el tiempo total de ejecución de la aplicación (*wall time*).

Tamaño Bloque	Mb en bloque	Fallos cache	Wall Time
500^2	0,953	0,49%	01:16:02
1.000^2	3,814	0,35 %	01:14:55
5.000^2	5,960	2,92 %	01:52:21

Tabla 1. Tiempos de ejecución para el algoritmo secuencial, según tamaño de bloque

Tamaños de bloque de 1.000^2 elementos aseguran una buena fijación de los mismos en memoria caché de último nivel. Los fallos de caché demuestran un mejor manejo de la memoria respecto de la utilización de bloques de 500^2 elementos. Con un tamaño de bloque de 5.000^2 elementos la cantidad de memoria requerido por el mismo excede considerablemente el tamaño del caché L3. De la misma forma se puede apreciar que la cantidad de fallos se incrementa, derivando esto en un tiempo de ejecución mayor.
Para los diferentes tamaños de bloque se consiguen mejores tiempos de ejecución frente al algoritmo de multiplicación estándar.
La tabla 2 muestra los tiempos de ejecución para los diferentes algoritmos (se considera una cantidad de 500^2 elementos para los algoritmos por bloque).

Algoritmo	**Fallos de Caché**	**Wall Time**
Producto corriente	2,72 %	1:26:23
Por Bloques Secuencial	0,49 %	1:16:02
Por Bloques Concurrente	86,41 %	1:07:49
Producto Coordenado	0,00%	0:00:12

Tabla 2. Tiempos de ejecución según el algoritmo.

Se observa una mejora de tiempos al aplicar el algoritmo de multiplicación de matrices por bloques frente al secuencial. Una de las variables observadas al aplicar este método es el mejor aprovechamiento de la memoria caché, ya que al cada bloque es reutilizado en productos sucesivos.
Por otro lado, al paralelizar el producto por bloques, utilizando cuatro núcleos, se obtienen mejoras en los tiempos de ejecución frente a la multiplicación por bloques secuencial.
La multiplicación por bloques concurrente presenta mayor cantidad de fallos de caché, y la misma estaría debida a que el proceso concurrente accede a diferentes zonas de la matriz completa al mismo tiempo, no pudiendo establecerse *a priori* la fijación de datos en memoria caché L3. (La documentación y mejor análisis de éste particular forma parte del trabajo futuro del grupo). A pesar del uso desfavorable del caché frente a otros algoritmos, el tiempo de pared incurrido en la ejecución por bloques concurrente es más bajo. Esta ganancia es debida a la incorporación de los tres núcleos ociosos por parte de la máquina virtual.
Tanto la cantidad de fallos de caché, como los tiempos de procesamientos alcanzados por el algoritmo de multiplicación coordenado, son difícilmente comparables con las magnitudes observadas por el resto de los algoritmos.

Discusión

La aplicación de algoritmos clásicos para afrontar problemas APSP (caminos más cortos entre dos puntos, por sus siglas en inglés: *All-Pair Shortest Path*) resulta en muchos casos una solución adecuada para la descripción teórica del problema, o la utilización de grafos de pequeñas dimensiones. No obstante, en aquellos casos donde el volumen de datos es considerable, los mismos se tornan inaplicables debido a la cantidad de procesamiento requerido.
A la luz de esta situación, surge la necesidad de discutir nuevos algoritmos basados muchas veces en paradigmas de programación alternativos como la programación concurrente.
Por otro lado deben establecerse estructuras de datos alternativas para contener los datos de acuerdo a la naturaleza del problema abordado, más allá de las estructuras de datos propuestas por la literatura.

El almacenamiento coordenado ofrece una utilización de memoria más adecuada frente al almacenamiento de una matriz en formato estándar, al menos para problemas de naturaleza similar al planteado. Consecuentemente, los tiempos de procesamiento decrecen considerablemente también. No obstante, este tipo de almacenamiento, junto con otras metodologías de almacenamiento para matrices dispersas, disipa las ventajas del almacenamiento estándar de una matriz, el cual cuenta con una organización y tipo de acceso directo, a partir del cálculo de la dirección de memoria de sus elementos a través de sus índices. Luego, el diseñador de software debería considerar estas ventajas y desventajas en función de la cantidad de memoria requerida y las características de acceso del algoritmo a los datos.

Conclusión

Se abordó la digitalización de datos en relación a la circulación de un vehículo sobre puntos de interés, para ser luego modelados a través de micromodelos de tránsito. Dichos datos corresponden principalmente a las coordenadas sobre las cuales se transita y la velocidad puntual en las mismas. Este conjunto de datos se complementa por los pares ordenados que conforman un grafo dirigido teniendo en cuenta la circulación de cada tramo.
El problema de establecer el camino más favorable entre tramos resulta en altos tiempos de procesamiento.
En primer lugar se propone como solución, acelerar la ejecución de procesamiento a través del producto matricial por bloques. Esto provee mejoras en los tiempos de

ejecución al realizar un mejor uso de la memoria caché de último nivel.

Luego, se propone la paralelización del algoritmo de multiplicación por bloques utilizando hilos de Java, con lo cual se consigue una segunda mejora en los tiempos de ejecución.

No obstante, estas mejoras no son suficientes para satisfacer los tiempos de ejecución requeridos por el proceso principal de simulación, ya que el mismo realiza el cálculo de múltiples productos matriciales, lo que conlleva a grandes tiempos de computación.

Con el objeto de establecer una solución definitiva, se propone la utilización de estructuras de datos alternativas, a través de un almacenamiento coordenado para la matriz de adyacencia. Este formato de almacenamiento resulta suficiente para atenuar el gran grado de dispersión de la matriz de adyacencia, y disminuir considerablemente la cantidad de memoria necesaria para mantenerla.

Finalmente se adecuan los algoritmos de producto matricial a la nueva estructura de datos consiguiendo tiempos de cómputo aceptables.

Agradecimientos

Se agradece al Departamento de Ingeniería en Sistemas de Información de la Facultad Regional Mendoza (UTN) por proveer un espacio natural para la investigación y desarrollo.

Referencias

[1]. Graph Theory with Algorithms and its Applications in Applied Science and Technology. Santanu Saha RAY. Springer Verlag. USA, 2013. ISBN: 978-81-322-0749-8.

[2]. Algorithms in Java. Third Edition. Robert SEDGEWICK. Addison Wesley. USA, 2003. ISBN: 978-0201361216.

[3]. La Congestión del Tránsito urbano: Causas y Consecuencias Económicas y Sociales. Ian THOMPSON, Alberto BULL. CEPAL (Naciones Unidas, División de Recursos Naturales e Infraestructura, Unidad de Transporte). Serie Recursos Naturales e Infraestructura (25). Chile, 2001. ISBN: 92-1-321865-6.

[4]. Transportation Systems Analisis: Models and Applications, 2º Edition. Ding-Zhu DU. Managing Editor, Panos M. Pardalos (University of Florida). Springer Verlag. USA, 2009. ISBN: 978-0-387-75856-5.

[5]. Ingeniería de Tránsito. Fundamentos y Aplicaciones 7ª Edición. Rafael CAL Y MAYOR, James CARDENAS. Alfaomega, 1995. ISBN: 970-15-0109-8.

[6]. Computer Architecture. A Quantitative Approach. 5º Edition. John HENNESSY, David PATTERSON. Morgan Kaufmann Pluslishers. USA, 2012. ISBN: 978-0-12-383872-8.

[7]. Intel. Descripción técnica del procesador utilizado, disponible en http://ark.intel.com/products/42915.

[8]. GARMIN. Descripción del producto disponible en https://buy.garmin.com/en-GB/GB/prod8703.html

[9]. Java and the Java Virtual Machine: Definition, Verification, Validation. Robert STÄRK, Joachim SCHMID, Egon BÖRGER. Springer Verlag, USA, 2001. ISBN-13: 978-3540420880.

[10]. Seguimiento de procesos en Linux. Disponible en http://www.cyberhades.com/2012/03/05/10-comandos-utiles-para-manejar-los-procesos-desde-la-consola-linux/.

[11]. Algebraic Graph Theory: Morphisms, Monoids and Matrices. Ulrich KNAUER. Editor: Carsten Carstensen (Berlin). Alemania, 2011. ISBN: 978-3-11-025408-2.

[12]. Numerical Linear Algebra. Text in Applied Mathematics. Grégoire ALLAIRE, Sidi Mahmoud KABER. Springer Verlag. USA, 2007. ISBN: 978-0387341590

[13]. Multicore Application Programming. Darryl GOVE. Addison-Wesley. USA, 2010. ISBN: 978-0-321-71137-3.

[14]. Professional Multicore Programming. Cameron HUGHES, Tracey HUGHES. Wiley Publishing Inc. USA, 2008. ISBN: 978-0-470-28962-4.

[15]. Modern Multithreading, Implementing Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs. Richard H. CARVER, Kuo-Chung TAI. Wiley-Interscience. USA, 2006. ISBN: 978-0-471-72504-6.

[16]. Pthreads Programming. Bradford NICHOLS, Dick BUTTLAR, Jacqueline PROULX. O'Reilly & Associates, USA, 1996. ISBN: 1-5692-115-1.

[17]. Programming with Posix Threads. David BUTENHOF. Addison Wesley Longman Inc. USA, 1997. ISBN: 0-201-63392-2.

[18]. Memory Systems: Cache, DRAM, Disk. Bruce JACOB, Spencer NG, David WANG. Morgan Kaufmann Publishers. USA, 2008. ISBN: 978-0-12-3797-51-3.

[19]. Sparse Matrices. Reginald TEWARSON. Department of Applied Mathematics and Statistics. Academic Press Inc. USA, 1973. ISBN: 9780126856507.

Datos de Contacto:

Julio Monetti. UTN – FR Mendoza. Rodriguez 273 - Mendoza. jmonetti@frm.utn.edu.ar