Procesamiento paralelo de registros de accidentes y multas en la ciudad de Montevideo

Lesly Acuña y Renzo Massobrio
Facultad de Ingeniería, Universidad de la República
Email: {lesly.acuna, renzom}@fing.edu.uy

***Resumen*—El presente artículo presenta el análisis de las estadísticas de accidentes de tránsito y el registro de multas aplicadas en la ciudad de Montevideo en el período 2006–2010. Se propone encontrar datos de accidentes y multas por barrio, clasificadas según distintos criterios. Debido al gran volúmen de datos a procesar, se utilizan técnicas de programación paralelas y se utiliza infraestructura de cómputo de alto desempeño. Se presenta un análisis experimental que muestra que el enfoque de paralelismo elegido resulta de alta aplicabilidad al problema, logrando muy buenos resultados de eficiencia computacional.**

I. Descripción del problema

Los Datos Abiertos de Gobierno son datos recolectados y/o producidos por el Gobierno con el fin de posibilitar la lectura, seguimiento, reutilización y combinación con otras fuentes de información. Son una herramienta que fortalece la transparencia de las agencias gubernamentales, a la vez que estimula la participación ciudadana [1]. Las características deseables en un conjunto de datos abiertos incluyen:

- Disponibilidad total y acceso gratuito (preferentemente web)
- Posibilidad de combinarlos con otras fuentes de información y distribuirlos de forma gratuita.
- Utilización de un formato abierto, cuya especificación esté disponible públicamente y de manera gratuita.

El Portal del Estado Uruguayo incluye una página específica para datos abiertos: datos.gub.uy. En particular, la Intendencia de Montevideo promueve de forma permanente la publicación de todos los datos que maneja tal como se manifiesta en la resolución 640/10 [2].

En el marco del programa de Datos Abiertos del Gobierno la Intendencia de Montevideo publica estadisticas históricas de infracciones de tránsito y de siniestros de tránsito con heridos [3] [4]. Ambos conjuntos de datos se encuentran georeferenciados. Se propone asignar cada registro de los conjuntos de datos al barrio en donde sucedió el hecho. Los barrios a utilizar son los definidos en el mapa de barrios provisto por la Intendencia de Montevideo a través del Catálogo de Datos Abiertos [6]. Debido al gran volumen de datos a procesar, se presenta una implementación multihilada siguiendo una estrategia de descomposición de dominio, que logra reducir los tiempos de procesamiento de estos datos.

El resto del artículo se organiza de la siguiente manera: la Sección II describe el preprocesamiento realizado sobre los datos, la justificación de usar computación de alto desempeño, y los detalles de implementación. La Sección III presenta el análisis experimental realizados para evaluar el desempeño del algoritmo propuesto y los resultados obtenidos. Finalmente en la Sección IV se presentan los principales resultados obtenidos del procesamiento de los datos junto a las conclusiones generales del trabajo.

II. Implementación de la solución

La siguiente sección detalla la estrategia de resolución adoptada, la justificación del uso de computación de alto desempeño para resolver el problema planteado, y los detalles de implementación.

II-A. Preprocesamiento de los datos

Para poder asignar cada registro al barrio correspondiente es necesario contar con las coordenadas (latitud y longitud) de cada uno de los registros. En el caso del juego de datos correspondiente a los accidentes, las coordenadas de cada registro ya vienen incluídas, siendo solamente necesario trasladarlas del sistema de referencia geográfico utilizado por la Intendencia de Montevideo (SIRGAS 2000) al tradicional (WGS 84) utilizando la herramienta libre QGIS [7].

Sin embargo, en el caso de los datos correspondientes a las multas de tránsito, la geolocalización no está dada por coordenadas sino por la intersección de dos vías de tránsito. Por esta razón fue necesario generar un archivo con las coordenadas de cada intersección de las vías de Montevideo. Este archivo se fue generado a partir del mapa de vías provisto por la Intendencia de Montevideo [5]. De esta forma, a la hora de procesar un registro de multa, primero se obtienen los identificadores del par de vías donde ocurrió el hecho, y luego se busca en el archivo generado las coordenadas geográficas correspondientes a esa intersección, para finalmente poder asignarla a un determinado barrio. Esto hace que el procesamiento de un registro de multas sea más lento que el de un registro de accidentes.

Al analizar los datos observamos que los registros de multas correspondientes al año 2010 eran en realidad una copia de los del año 2009. Por esta razón los descartamos, e informamos a la Unidad Técnica de Tránsito y Transporte de la Intendencia de Montevideo de este error.

Por su parte, el mapa de barrios se encuentra en formato Shapefile, lo que hace díficil su manipulación de forma programática sin una herramienta de información geográfica. Por esta razón, se utilizó QGIS para obtener la lista de puntos que delimitan cada barrio y sus coordenadas. De esta forma, dadas las coordenadas de un punto correspondiente a un registro a procesar, y una lista de puntos que delimitan a un barrio, es

posible calcular si dicho punto se encuentra dentro o fuera de ese barrio.

II-B. Justificación del uso de HPC

Los conjuntos de datos disponibles presentan la información referida al período 2006—2010. Para el caso de los accidentes se cuenta con 41121 registros, mientras que en el caso de las multas esta cifra asciende a 873083. Adicionalmente, se cuenta con un total de 63 barrios en la ciudad de Montevideo. Encontrar el barrio al cuál pertenece un determinado registro implíca un esfuerzo computacional considerable (ver Seccion III), lo que se traduce en tiempos de ejecución prolongados en una implementación secuencial. Adicionalmente, es de esperar que los conjuntos de datos sean actualizados con el correr de los años, por lo que la solución implementada debe escalar bien ante el aumento de los datos a procesar.

II-C. Estrategia de resolución

El problema resulta fácilmente paralelizable siguiendo un esquema de descomposición de dominio: la asignación de un registro a un barrio no depende de otros registros. De este modo, se decidió realizar una solución utilizando programación multihilada para poder procesar los datos de forma más eficiente.

Se implementaron dos algoritmos: uno para procesar el conjunto de datos de multas y otro el de accidentes. Para la implementación se utilizó un modelo maestro/esclavo usando un esquema de descomposición de dominio. La implementación se realizó en el lenguaje Python. La elección de este lenguaje se debe en gran parte a las facilidades que ofrece en el procesamiento de cadenas de carácteres y en la lectura/escritura de archivos. Si bien desde el punto de vista de eficiencia computacional puede ser inferior a otras alternativas (e.g. C/C++), el análisis experimental mostró buenos resultados.

La implementación es similar tanto para el procesamiento de los accidentes como de las multas, salvo por el paso adicional que debe hacerse en el caso de las multas para encontrar las coordenadas de la intersección de las vías que figuran en el registro. En primer lugar, se carga el archivo con las coordenadas de los puntos que delimitan a cada barrio, de forma que esta información quede accesible por todos los hilos. En el caso de las multas, se carga también el archivo con la intersección de las vías para que sea visible por todos los hilos. Luego se calcula la cantidad de líneas del archivo con los registros a procesar utilizando el comando linux *wc -l file* y se divide en tantas partes como número de hilos disponibles para ejecutar utilizando el comando *split -l lines file*.

Luego, se crean los hilos y se le asigna a cada hilo un archivo para procesar. Cada hilo lee secuencialmente el archivo que le fue asignado, donde cada línea es un registro a procesar. En el caso de las multas, primero lee el cruce de vías que figura en el registro, y busca en la lista que se encuentra globalmente disponible las coordenadas correspondientes a dicho cruce. Una vez que se tiene las coordenadas del registro, se itera sobre la lista de barrios, verificando si dicho registro se encuentra dentro o fuera. Cuando se encuentra el barrio al que pertenece el registro, se actualiza el contador de dicho barrio. Al finalizar, el hilo deja sus resultados (conteo de multas/accidentes en cada barrio), en una cola de resultados para ser procesados por el maestro.

Cuando todos los esclavos finalizan, el maestro itera sobre la cola de resultados, sumando los contadores de cada esclavo (separados por barrios). Imprime los resultados en pantalla, los almacena en un archivo para luego poder ser procesados utilizando QGIS, y borra los archivos temporales generados.

II-D. Post-procesamiento de los resultados

La salida de ambos algoritmos es un archivo CSV con el identificador de cada barrio y el contador de registros pertenecientes a ese barrio. Esos archivos pueden ser fácilmente importados a QGIS, donde se puede hacer un *join* por identificador de barrio con el mapa de barrios de la Intendencia. De esta manera, se tiene asociado a cada barrio en el mapa el contador resultante del procesamiento de los datos. Posteriormente se coloreó el mapa de forma graduada de acuerdo a estos contadores, de forma de tener una representación visualmente atractiva que permita ver los resultados obtenidos luego del procesamiento.

III. Análisis experimental

La siguiente sección presenta el análisis experimental realizado para validar la estrategia de paralelismo utilizada. Se presentan los principales resultados y sus conclusiones.

III-A. Métricas de Paralelización

Debido a que no se cuenta con un algoritmo secuencial para comparar la mejora de desempeño al aplicar paralelismo, para medir la calidad de los algoritmos se utiliza como parametros los tiempos de ejecución y la paralelicibilidad.

La paralelicibilidad se define como

$$Paralelicibilidad = T_1/T_N$$

Donde T_1 es el tiempo de ejecución del algoritmo paralelo en una unidad de cómputo y T_N es el tiempo de ejecución del algoritmo paralelo en N unidades de cómputo. La paralelicibilidad es una medida de que tanto es posible mejorar el desempeño al utilizar recursos de cómputo adicionales para ejecutar el algoritmo paralelo.

III-B. Evaluación experimental

Como se explicó en II-B, la cantidad de datos disponibles es considerable, por lo que es deseable que el algoritmo escale al aumentar el número de cores. Para la evaluación estadística de los tiempos de ejecución se realizaron 20 ejecuciones con 1, 4, 8, 16, y 24 cores.

Experimentos iniciales mostraron alta variabilidad en los tiempos de ejecución del algoritmo al utilizar la misma cantidad de núcleos en distintas ejecuciones. El problema se debía al intenso uso del disco para leer los archivos de entrada, lo que repercutía en la performance del NFS. Este problema se resolvió copiando previamente los archivos a leer a la carpeta /tmp de forma que quedaran en almacenamiento local a la máquina donde se realizaba la ejecución. Esto derivó en una notoria mejora, por lo que la evaluación experimental se llevó acabo siguiendo esta estrategia.

El ambiente de ejecución fue el Cluster Fing, donde se utilizaron las siguientes colas de trabajo:

- *serial:* con un core disponible y con hasta 240 horas de walltime, utilizado para correr el caso del algorimo de multas con un solo hilo.
- *small jobs:* con 16 cores disponibles y hasta 240 horas de walltime, utilizados para los casos de 4 y 8 hilos en el caso de algoritmo de multas.
- *quick jobs:* con limite de 64 cores y 4 horas de walltime, se utilizó para correr todos los algoritmos de accidentes y los casos de mas hilos del algoritmo de multas.

En el Cluster Fing los nodos de trabajo disponibles se componen de máquinas Intel Xeon CPU E5430, E5520, E5-2650, y de máquinas AMD Opteron Processor 6172, 6272 y 6376. Las máquinas utilizadas en la cola de quick jobs son máquinas AMD Opteron Processor 6272 @ 2.09Ghz. A continuación se presentan las pruebas realizadas al correr el algoritmo en un nodo utilizando 1, 4, 8, 16 y 24 cores. Se realizan 20 ejecuciones independientes.

Las Tablas I y III muestran los tiempos de ejecución para cada una de las 20 ejecuciónes de los algoritmos de accidentes y multas respectivamente, utilizando distinta cantidad de recursos de cómputo. En cada caso se calcula el tiempo de ejecución promedio y mínimo asi como su desviación estándar. Los tiempos en las tablas son expresados en el formato mm:ss:ms (minutos, segundos, milésimas de segundo).

Las Tablas II y IV presentan los tiempos de ejecución promedio y la paralelicibilidad de los algoritmo de accidentes y multas respectivamente. El tiempo de ejecución promedio mostrado en estas tablas está expresado en segundos debido a que fue el usado de referencia para las gráficas de las figuras 1 y 3 que se muestran en la sección siguiente.

Tabla I: Tiempos de ejecución del algoritmo de accidentes

Nro Ejecución	**Nro Procesadores**				
	1	4	8	16	24
1	02:39:8	00:40:0	00:24:0	00:13:4	00:09:2
2	02:38:3	00:40:8	00:24:6	00:13:6	00:09:6
3	02:37:8	00:41:5	00:21:6	00:12:9	00:09:5
4	02:41:9	00:40:8	00:23:6	00:13:5	00:09:5
5	02:38:2	00:40:4	00:24:0	00:13:0	00:10:0
6	02:40:7	00:40:5	00:22:3	00:12:9	00:09:2
7	02:38:6	00:40:9	00:23:1	00:12:9	00:09:6
8	02:39:6	00:41:1	00:23:9	00:12:9	00:10:9
9	02:39:5	00:41:3	00:22:2	00:14:0	00:10:0
10	02:38:5	00:41:6	00:24:0	00:13:1	00:09:8
11	02:38:3	00:40:8	00:23:4	00:13:1	00:10:0
12	02:39:1	00:40:6	00:24:8	00:13:0	00:09:6
13	02:40:3	00:40:6	00:23:0	00:13:0	00:09:7
14	02:36:1	00:42:0	00:23:1	00:12:6	00:09:8
15	02:38:5	00:41:8	00:23:2	00:12:9	00:09:5
16	02:38:0	00:40:8	00:23:2	00:13:8	00:09:7
17	02:38:2	00:40:5	00:22:6	00:16:4	00:10:2
18	02:39:3	00:40:4	00:22:9	00:13:4	00:09:8
19	02:37:3	00:41:0	00:22:5	00:12:5	00:09:8
20	02:36:9	00:40:4	00:21:7	00:13:0	00:12:3
Promedio	**02:38:7**	**00:40:9**	**00:23:2**	**00:13:3**	**00:09:9**
Mín valor	**02:36:1**	**00:40:0**	**00:21:6**	**00:12:5**	**00:09:2**
Desviacion Est.	**1,5E-05**	**6,08E-06**	**1,02E-05**	**9,47E-06**	**7,90E-06**

Tabla II: Tiempo Ejecución promedio, paralelicibilidad y efectividad del algoritmo de accidentes

Procesadores	Tiempo ejecución (s)	Paralelicibilidad
4	40,93s	3,8793
8	23,23s	6,8351
16	13,33s	11,9115
24	9,94s	15,9738

Tabla III: Tiempos de ejecución del algoritmo de multas

Nro Ejecución	**Nro Procesadores**				
	1	4	8	16	24
1	44:55:9	12:15:8	05:20:8	03:49:9	02:17:5
2	44:49:6	12:05:9	05:51:1	03:20:9	02:32:1
3	44:46:5	12:00:2	05:35:9	03:31:9	02:22:9
4	45:02:6	12:16:3	05:41:0	03:46:4	02:14:2
5	44:59:4	11:59:7	05:30:7	03:13:1	02:18:4
6	44:48:8	12:05:6	05:28:3	03:15:0	02:15:2
7	44:48:2	12:00:6	05:11:9	03:26:4	02:17:2
8	45:09:3	12:00:5	05:10:6	03:22:7	02:15:4
9	44:42:8	12:01:2	05:36:9	03:13:1	02:18:3
10	44:58:4	12:03:2	05:41:3	03:14:6	02:23:5
11	46:02:9	12:02:6	05:43:7	03:12:9	02:21:9
12	45:01:3	11:50:6	05:27:8	03:14:6	02:13:9
13	45:02:2	11:42:7	05:52:7	03:10:3	02:17:1
14	46:28:1	11:39:0	06:05:6	03:14:4	02:17:7
15	45:39:6	11:36:0	05:39:5	03:11:0	02:15:5
16	45:29:6	11:40:0	05:24:7	03:10:5	02:14:2
17	45:17:0	11:38:1	05:33:5	03:12:6	02:21:1
18	44:57:3	11:35:4	05:35:7	03:14:4	02:25:9
19	44:51:5	11:34:8	05:36:2	03:11:7	02:12:8
20	44:48:4	11:39:0	05:56:3	03:13:2	02:15:0
Promedio	**45:08:0**	**11:53:3**	**05:36:2**	**03:19:0**	**02:18:5**
Mín valor	**44:42:8**	**11:34:8**	**05:10:6**	**03:10:3**	**02:12:8**
Desviacion Est.	**0,000318**	**0,000161**	**0,000160**	**0,000132**	**5,54E-05**

Tabla IV: Tiempos de ejecución promedio y paralelicibilidad del algoritmo de multas

Procesadores	Tiempo ejecución (s)	Paralelicibilidad
4	713,39s	3,7960
8	336,26s	8,0534
16	199,03s	13,6061
24	138,52s	19,5497

III-C. Discución de resultados experimentales

Analizando los resultados obtenidos en la sección anterior se observa que la implementación de forma paralela del algoritmo de accidentes obtiene una mejora en el tiempo de más del 90 % con 24 recursos de cómputo frente al uso de un único núcleo. En el caso del algoritmo de multas, esta mejora en el tiempo asciende a un 95 %.

Además, la paralelicibilidad de los algoritmos nos indica que las soluciones propuestas pueden escalar facilmente tanto al agregar nuevos recursos de cómputo como en el eventual caso que los conjuntos de datos aumenten su tamaño.

Las Figuras 1 y 2 muestran los tiempos de ejecución promedios y la paralelicibilidad del algoritmo de accidentes en función de la cantidad de núcleos utilizados. Las Figuras 3 y 4 muestran el equivalente para el algoritmo de multas.

Se observa en los gráficos de paralelicibilidad que su tendencia es muy próxima a la lineal, lo que permite concluir

que el algoritmo escalaría bien ante el aumento de la cantidad de los datos, siendo suficiente aumentar la cantidad de recursos de cómputo para lograr procesar mayores volúmenes de datos.

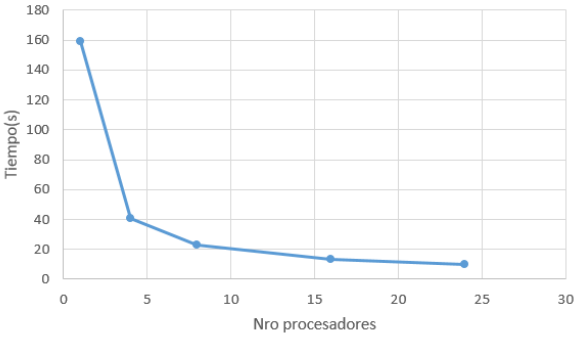

Figura 1: Tiempo de ejecucion de algoritmo de accidentes en funcion de la cantidad de procesadores utilizados.

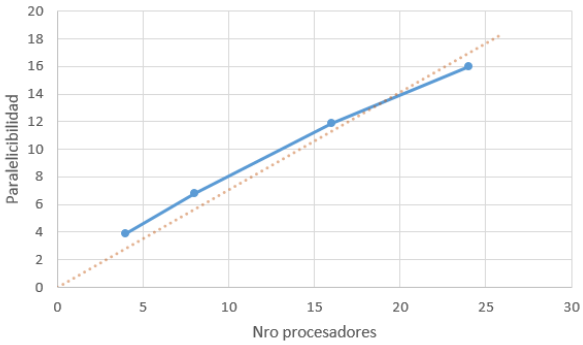

Figura 2: Paralelicibilidad del algoritmo de accidentes

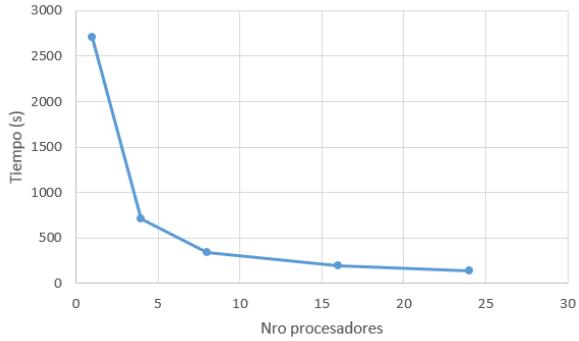

Figura 3: Tiempo de ejecucion de algoritmo de multas en funcion de la cantidad de procesadores utilizados.

IV. Resultados obtenidos y conclusiones

En esta sección se presentan los principales resultados junto a algunas conclusiones generales del proyecto.

IV-A. Resultados del procesamiento

Las Figuras 5 y 6 muestran los resultados obtenidos para el conjunto total de accidentes y multas respectivamente. Se

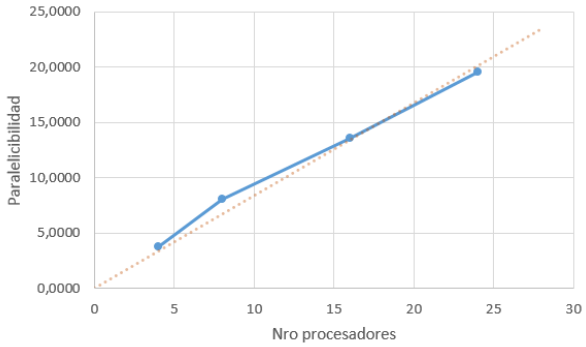

Figura 4: Paralelicibilidad del algoritmo de multas

puede observar que, como es esperable, la mayor concentración se da en las zonas céntricas de la ciudad donde el tránsito es mayor.

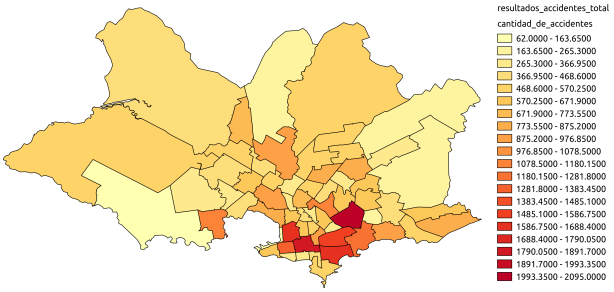

Figura 5: Total de accidentes

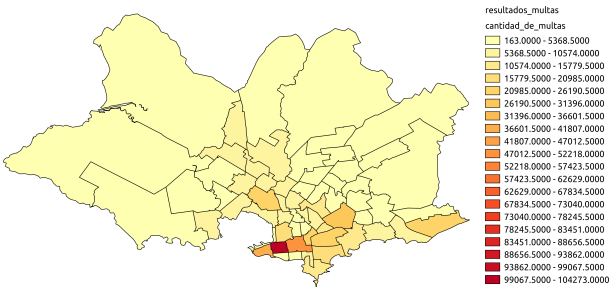

Figura 6: Total de multas

De forma obtener otros resultados relevantes, se procesaron los conjuntos de datos siguiendo la misma estrategia pero separando previamente según distintos criterios. Para el caso de los accidentes, se separó el procesamiento según la gravedad del accidente (leve, grave o fatal). En el caso de las multas, se separó según la franja horaria en que se impuso la multa (madrugada (00:00-07:59), mañana/tarde (08:00-15:59), tarde/noche(16:00-23:59)). Las Figuras 7–9 muestran los resultados de los accidentes de tránsito discriminados según su gravedad. Las Figuras 10–12 muestran el total de multas discriminados según franja horaria.

En el caso de las multas, no se observan grandes cambios en las distintas franjas horarias, excepto un aumento en algunos barrios más allá del centro en la franja correspondiente a la

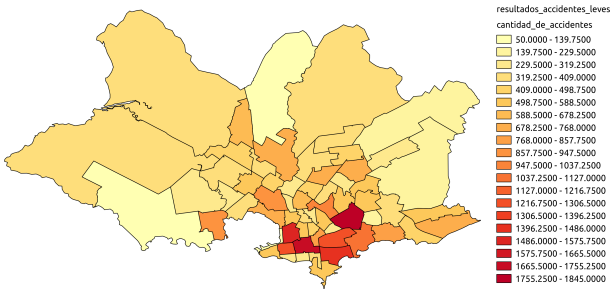

Figura 7: Total de accidentes leves

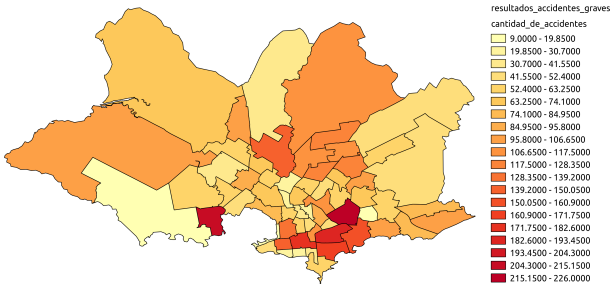

Figura 8: Total de accidentes graves

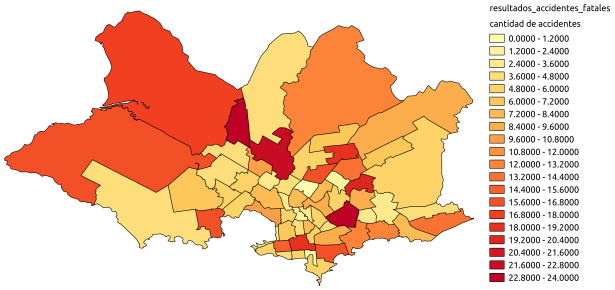

Figura 9: Total de accidentes fatales

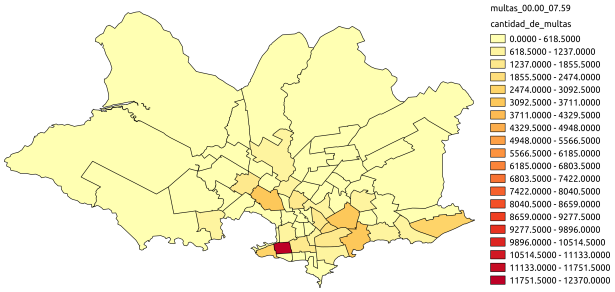

Figura 10: Total de multas 00:00–07:59

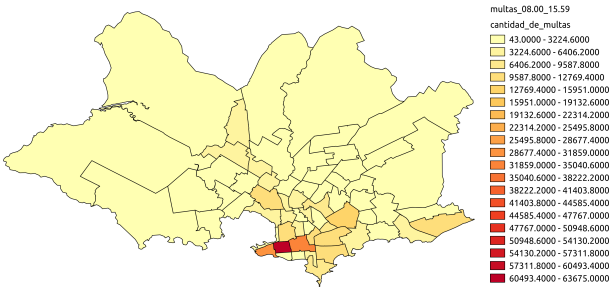

Figura 11: Total de multas 08:00–15:59

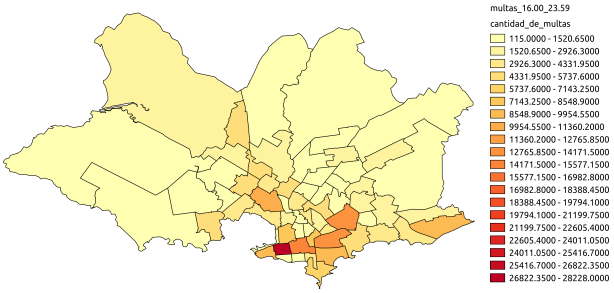

Figura 12: Total de multas 16:00–23:59

tarde/noche. Para el caso de los accidentes, la separación según gravedad del accidente arroja resultados muy interesantes, que muestran que mientras que el total de accidentes se concentra sobre todo en zonas céntricas de la ciudad, si se consideran solamente los accidentes graves y especialmente los fatales, las zonas más periféricas de la ciudad son las que presentan mayores números. Esto podría explicarse por las velocidades a las que se transita por estas zonas en comparación a las zonas céntricas.

IV-B. Conclusiones

El código fuente implementado y los resultados obtenidos, fueron alojados en un repositorio en Github [8] de forma pública y con una licencia abierta para poder ser utilizados libremente. Además, se agregaron a la lista de aplicaciones que utilizan datos abiertos que lleva el Catálogo de Datos Abiertos. Por último, se enviaron los resultados obtenidos a la Unidad Técnica de Tránsito y Transporte de la Intendencia de Montevideo, junto a algunos comentarios sobre inconsistencias que encontramos en los datos publicados por dicha unidad, con el fin de que sean corregidos y actualizados.

El algoritmo propuesto abre posibilidades a distintos tipos de cálculos, como los mostrados en la sección IV. Por ejemplo, se puede utilizar para saber la evolución de los accidentes a través de los años en un determinado barrio, ver la cantidad de multas en fechas donde tradicionalmente suele haber un aumento de las mismas (Noche de la Nostalgia, Semana de Turismo, etc.). Adicionalmente los datos de multas contienen la información del tipo de infracción cometida, lo que podría ser un interesante dato a representar. Sin embargo observamos importantes inconsitencias en estos datos, por lo que no fueron utilizados.

La evaluación experimental llevada a cabo muestra una

importante mejora de la performance para ambos algoritmos paralelos al aumentar la cantidad de recursos de cómputo. Además, la paralelicibilidad de los algoritmos sugiere que los mismos pueden escalar bien en el eventual caso de una aumento en la cantidad de datos, lo cual es esperable que suceda en los próximos años a medida que la Intendencia los libere.

REFERENCIAS

[1] Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad de la Información y del Conocimiento
Conceptos básicos de Datos Abiertos
http://www.agesic.gub.uy/innovaportal/v/1544/1/agesic/conceptos_basicos_de_datos_abiertos.html
Accedido en Marzo de 2015.

[2] Intendencia de Montevideo
Resolución Nº640/10
http://monolitos.montevideo.gub.uy/resoluci.nsf/de053405568724cf832575ae004f0467/7adaf8ec8d70033b832576d60041760f
Accedido en Marzo de 2015.

[3] Catálogo de Datos Abiertos
Multas de tránsito
https://catalogodatos.gub.uy/dataset/multas-transito
Accedido en Marzo de 2015.

[4] Catálogo de Datos Abiertos
Accidentes de tránsito Montevideo
https://catalogodatos.gub.uy/dataset/accidentes-de-transito-montevideo
Accedido en Marzo de 2015.

[5] Catálogo de Datos Abiertos
Vías de tránsito
https://catalogodatos.gub.uy/dataset/vias-de-transito-montevideo
Accedido en Marzo de 2015.

[6] Catálogo de Datos Abiertos
Límites de Barrios
https://catalogodatos.gub.uy/dataset/limites-barrios
Accedido en Marzo de 2015.

[7] QGIS
Un Sistema de Información Geográfica libre y de Código Abierto
http://www.qgis.org/es/site/
Accedido en Marzo de 2015.

[8] Procesamiento paralelo de registros de accidentes y multas en la ciudad de Montevideo.
https://github.com/renzomassobrio/hpc_transito
Accedido en Marzo de 2015.