Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Guadalajara

Proyecto Final: Computo Paralelo y Distribuido

Generación en paralelo de una matriz de adyacencia a partir de conteos de afecciones médicas en registros de egresos hospitalarios

Luis Miguel Benitez Ruiz

994361

Presentación del problema

El problema que se trabaja es la generación de una matriz de adyacencia que representa la 'cercanía' entre afecciones médicas a partir de una base de datos de egresos hospitalarios. Lo que se pretende es mostrar gráficamente la relación entre diferentes afecciones médicas, y así poder extraer nuevo conocimiento de la gran cantidad de información de la que se dispone.

Esto es parte de un proyecto de tesis, y es un paso necesario para los experimentos a realizarse. Por esta razón es necesario tener un procedimiento que pueda obtener un subconjunto de datos en base a parámetros, y generar la matriz de adyacencia correspondiente con gran velocidad. De lo contrario todo el proceso se alentara, pues se prevé que la generación de estos grafos sea una tarea común y frecuente.

Representación de los datos

La base de datos con la que se trabaja es el SAEH (Subsistema de Automatizado de Egresos Hospitalarios). Esta es una base de datos que registra todos los egresos hospitalarios en hospitales de instituciones de salud pública federales y estatales. Cada uno de estos egresos está asociado a al menos una afección médica. Para cada persona que egresa de un hospital que utiliza el sistema SAEH, se registra todas las afecciones médicas que se hayan presentado con ese paciente. Se registra por lo menos la afección por la cual ingreso, y aquella que los médicos consideraron más importante de todas las que haya presentado, que puede ser la misma.

A continuación se describen las columnas relevantes de las dos tablas principales con las que se trabaja para extraer los datos.

CLUES	ID	AFECCION	NUM_AFEC

Tabla 1. Descripción de Tabla de afecciones.

En las descripciones mostradas, CLUES representa un código único identificador de cada instalación de atención médica. ID es un identificador único para cada uno de los egresos registrados. La columna AFECCION contiene el código ICD de una de las afecciones presentadas en el caso identificado por ID. Por lo tanto existe una entrada con el mismo ID en la tabla de afecciones por cada afección que se haya presentado en el caso representado por ese ID. NUM_AFEC es un contador para indicar el índice de la AFECCION dentro del caso representado por ID.

CLUES	ID	AFECPRIN	AFECINI	EDOCVE	INGRESO	EGRESO

Tabla 2.Descripcion de Tabla de Egresos

CLUES e ID tienen los mismos significados que en la tabla 1. EDOCVE es un código numérico que representa el estado de la republica donde se ubica el centro de atención identificado por el CLUES. AFECPRIN es la afección que los médicos hayan considerado más relevante para el paciente, y AFECINI es la afección inicial por la cual ingreso o se presentó al centro de atención. INGRESO y EGRESO son las fechas correspondientes al ingreso y al egreso del paciente en el centro de atención.

Todos los campos que representan una afección son códigos ICD (International Classification of Diseases), creado y mantenido por la OMS para representar todas las afecciones médicas que están registradas por la OMS. Adicionalmente se almacena información demográfica básica sobre el paciente, como talla, peso, sexo, lenguaje hablado, entre otras cosas. Sin embargo estos no son relevantes para el problema actual.

Los datos extraídos originalmente de la base de datos son los mostrados en las tablas 1 y 2. Cada afección está representada por un código alfanumérico de 1 carácter seguido de 3 números (el código ICD). Para almacenarlos en memoria durante la ejecución del programa se aprovecha una estructura propia del lenguaje R, llamada factor. Un factor es un vector ordenado, con una representación numérica (*value*) y otra alfanumérica (*label*) para cada elemento único en el vector.

De esta forma puede cambiarse entre una representación alfanumérica utilizando los *labels*, dándole mayor semántica a los datos y permitiendo generar visualizaciones con significado para un ser humano; y una representación numérica que facilita la transferencia y procesamiento de los datos.

Para representar los datos en el lenguaje C se utiliza precisamente esta característica, y todos los datos representando afecciones enviados al programa en C son siempre utilizando su representación numérica.

Implementación

El algoritmo que se implemento es el siguiente.

___

Entradas: *fecha_inicio,fecha_fin,estado,afeccion*

Salida: matriz de adyacencia

1) Por cada IDs de la tabla de EGRESOS que cumpla con las condiciones recibidas de entrada

> 2) agregar todas las entradas de la tabla AFECCIONES con ese ID al subconjunto *cases_subset*,
>
> 3) Reemplazar cada AFECCION en *cases_subset* por su representación numérica.
>
> 4) agregar una posición al vector *cases_count*, con el conteo, del número de afecciones para el ID actual.
>
> Es decir, cases_count contiene una entrada por cada ID en cases_subset, con el número de AFECCIONES en dicho ID. La posición *i* en cases_count corresponde al ID numero *i* extraído de EGRESOS en el paso 1)

5) Enviar *cases_subset, cases_count,* junto con un entero indicando sus tamaños y apuntador a una matriz *adj* de tamaño *NxN*, donde *N* es el número de afecciones distintas en *cases_subset.*

6) En GPU: por cada caso *cur_case* en *cases_subset*, se asigna un hilo para leer las afecciones presentes en el caso, y hacer las modificaciones en la matriz *adj* de la siguiente forma:

i) por cada afección *af* en *cur_case*

ii) por cada afección *afo* en *cur_case*

adj[af][afo] ++;

7) Transferir *adj* de vuelta al programa R

8) Asignar los valores del factor incluido en *cases_subset* a los nombres de las columnas y renglones en la matriz *adj*

9) Fin.

___

En el paso 6, es importante aclarar que debido a que las afecciones en *cases_subset* se mantienen mediante factores, al momento de convertirse a una representación numérica, estos toman los valores *1*...*N*, donde *N* es el ancho y alto de la matriz. Por lo que la representación numérica de una afección es también su número de renglón y columna en la matriz *adj.* Esto simplifica enormemente el encontrar la celda que representa una relación entre dos afecciones.

Resultados

Para la versión secuencial se utilizó un código escrito en R, de la forma más eficiente que se pudo implementar, y utilizando las funciones existentes para realizar operaciones sobre matrices de forma eficiente para tratar de obtener un buen rendimiento. Para la versión paralela se utilizó un kernel de CUDA y una tarjeta NVIDIA GeForce GT 650 M con 2 GB de RAM y 384 núcleos. El CPU de la computadora utilizada para las pruebas es un Intel i7 a 2.3 GHz con 8 núcleos y 12 GB de RAM.

El subconjunto de datos se hizo seleccionando todos los egresos presentados en el estado de Oaxaca entre el 1 y el 15 de Agosto de 2011. El estado y las fechas fueron elegidos arbitrariamente. El resultado fueron 2222 entradas en *cases_subset*, que correspondían a 1459 casos únicos, es decir, el tamaño de *cases_count* era de 1459 y la matriz de adyacencia fue de 279x279, lo que significa que se encontraron 279 afecciones únicas en *cases_count*.

Las pruebas consistieron en ejecutar la tarea de generar una matriz de adyacencia 10 veces con los mismos datos, y se reporta el promedio de tiempo en las ejecuciones. Siempre se reinicializan todas las estructuras de datos antes de la siguiente ejecución. El código secuencial que calcula el contenido de la matriz tarda 6.02 segundos, mientras que el código paralelo utiliza solamente 0.07584. Por lo que el speedup alcanzado con la versión paralela fue de poco más de 79.

Es importante notar que el código secuencial genera los contenidos de la matriz, pero separados en varios vectores que necesitaran unirse más adelante. Sin embargo, R es muy eficiente en operaciones vectoriales, mientras que en el caso paralelo los datos ya están en una sola estructura, pero deben transferirse, de CUDA hacia R, por lo que se considera que las diferencias de tiempo para este post proceso no serán significativas, sobre todo considerando el speedup alcanzado.

Conclusiones

Basándose únicamente en el speedup obtenido, se podría decir inmediatamente que es conveniente la implementación paralela utilizando CUDA para resolver este problema. Sin embargo, hace falta afinar unos detalles para que esta solución sea práctica. Resolviendo estos problemas, entonces podemos concluir sin dudas que la paralelizacion es muy conveniente para este proyecto y que debería dedicarse tiempo a implementarla para aprovecharla durante toda la vida del proyecto.

En primer lugar, se necesita generar una librería accesible desde R, el lenguaje principal de trabajo para este proyecto, pues para estas pruebas se movieron los datos 'manualmente' entre los programas en R y C. Esto es algo en lo que se está trabajando ya, pues eso permitiría eliminar multitud de pasos manuales y limitarse a hacer una llamada a una función.

El segundo problema es el de la disponibilidad del equipo, pues solo se cuenta con una computadora con GPU capaz de aprovechar el código en CUDA. LA alternativa que se está considerando para paliar este problema, es generar la librería mencionada para el punto 1, con la capacidad de ejecutar código utilizando CUDA o utilizando OpenMP. Seria responsabilidad de la librería revisar si existen dispositivos CUDA disponibles, y en caso contrario paralelizar utilizando OpenMP.

Esto permitiría mayor portabilidad de la paralelizacion del problema, por lo que sería algo más sencillo compartirla y poder generar estos grafos en más de una sola computadora. Considerando el tamaño de la base de datos que se utiliza, es buena idea prepararse para que sea utilizada por alguien más en el futuro, y preparar las herramientas que se crearan en el proceso del proyecto para que sean reutilizables.