Acelerando Operaciones SQL en GPU con CUDA

Integrante: Israel Peña

Motivación

- Consultas SQL se demoran mucho si Base de Datos es grande.

- Existen métodos para procesar gran cantidad de datos.
  - Data Mining
  - MapReduce

- Estudiar la implementación de una Interface de SQL que procesa los datos en GPU.

Introducción (SQLite)

- Base de Datos Open Source.
- SQLite está escrita para ser compilada directamente en el código fuente de la Aplicación
- Arquitectura Simple:
  - Interface de Usuario
    (Librería de Funciones y estructuras en C)
  - Procesador de Comando
    (Como un Compilador, crea un set de pasos discretos Opcodes)
  - Máquina Virtual
    (Ejecuta los Opcodes)
- Uso secuencial muy simple

```
sqlite3_open("nombreDataBase.db", &db);
r = sqlite3_exec(db, sql, &test_callback, &rows, &err);
```

Procesador de Comando (Opcodes)

- Traduce una Query a pasos discretos.
- Opcodes parecido a Assembly
- Ejemplo:
  - ***Integer*** carga un entero en cierto registro.
  - ***Column*** carga data de una columna en un registro.
  - ***Le*** indica si dato del registro 1 es menor o igual al dato del registro 3
  - ***Ge*** indica si dato del registro 2 es mayor o igual al dato del registro 3
  - Del 6 al 14 se ejecutan por cada línea.

```
1. SELECT id, uniformi, normali5
   FROM test
   WHERE uniformi > 60 AND normali5 < 0
```

```
0:   Trace        0   0   0
1:   Integer      60  1   0
2:   Integer      0   2   0
3:   Goto         0   17  0
4:   OpenRead     0   2   0
5:   Rewind       0   15  0
6:   Column       0   1   3
7:   Le           1   14  3
8:   Column       0   2   3
9:   Ge           2   14  3
10:  Column       0   0   5
11:  Column       0   1   6
12:  Column       0   2   7
13:  ResultRow    5   3   0
14:  Next         0   6   0
15:  Close        0   0   0
16:  Halt         0   0   0
17:  Transaction  0   0   0
18:  VerifyCookie 0   1   0
19:  TableLock    0   2   0
20:  Goto         0   4   0
```

Sphyraena

- Creada por Peter Bakkum (2010)
- API necesaria para ejecutar consultas SQL en GPU
- Estructuras de Datos relevantes:

```
15 struct sphyraena_results {
16 int rows;
17 int columns;
18 int stride;
19 int types [];
20 int offsets [];
21 char r [];
22 };
```

```
1 struct sphyraena {
2 sqlite3 *db;
3 sphyraena_data *data_cpu;
4 char *data_gpu;
5 sphyraena_results *results_cpu;
6 sphyraena_results *results_gpu;
7 sphyraena_stmt *stmt_cpu;
8 size_t data_size;
9 size_t results_size;
10 int pinned_memory;
11 int threads_per_block;
12 int stream_width;
13 };
```

Sphyraena (ejemplo de Uso)

```
sqlite3 *db;
sphyraena sphy;
// Inicializa SQLite usando dbfile.db
sqlite3_open("dbfile.db", &db);

// #define DATA_SIZE (SPHYRAENA_MB * 128)
// #define RESULTS_SIZE (SPHYRAENA_MB * 128)
sphyraena_init(&sphy, db, DATA_SIZE, RESULTS_SIZE, 0);

// Transforma la data en test
// a un formato de filas y columnas
sphyraena_prepare_data(s, "SELECT * FROM test");

// mueve la data de memoria principal a memoria de GPU
sphyraena_transfer_data(&sphy);
    // Dentro de la función:
    r = cudaMemcpy(s->data_gpu, s->data_cpu->d,
      s->data_cpu->rows * s->data_cpu->stride,
      cudaMemcpyHostToDevice);
```

```
// Ejecuta una consulta en GPU
sphyraena_select(&sphy,
"SELECT column1, column2 FROM test_table
WHERE column1 < column2", 0);
    // Dentro de la función
    // Se obtienen los Opcodes de SQLite3
    // y se transforman al formato de Sphyraena
    // también se ejecuta la máquina virtual con:
    sphyraena_vm(s);

// Transfiere los resultados de la consulta
// de la GPU a la memoria principal
sphyraena_transfer_results(&sphy);
```

Máquina Virtual (CUDA)

- En secuencial, la Máquina Virtual de SQLite se encarga de todo
- Se copian al device la data y la consulta como Opcodes
- Luego se define la cantidad de bloques y threads y se llama al kernel
  - `s->threads_per_block es 192`
  - `int blocks = (s->data_cpu->rows + s->threads_per_block - 1) / s->threads_per_block;`
  - `int thread_rows = 1;`
  - `int start_row = 0;`
  - `VmKernel<<<blocks, s->threads_per_block>>>((char*)s->data_gpu, s->results_gpu, start_row, blocks, thread_rows);`
- El Kernel es básicamente un switch case donde recorre cada opcode con un while y realiza la acción que se requiere, luego guarda los resultados si es necesario

Experimentos (Hardware y Software)

- Intel® Core™ i3-9100F CPU @ 3.60GHz × 4
- Ubuntu 20.04.2
- NVIDIA GeForce GTX 1650 SUPER
- CUDA 11.4
- SQLite 3.6.22

Anteriores Experimentos:

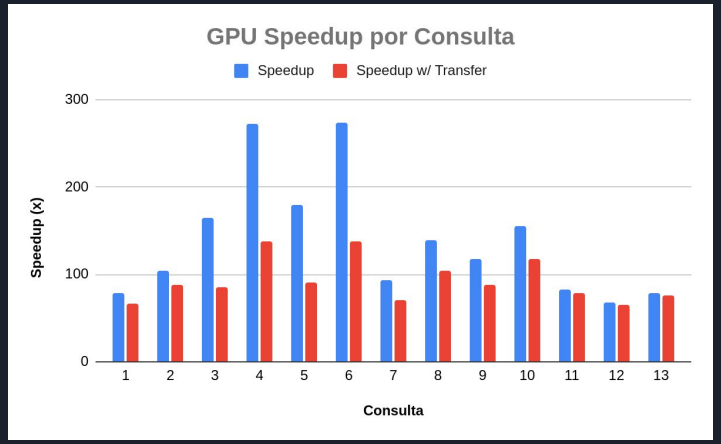
- Intel Xeon X5550 2.66GHz quad-core
- Linux 2.6.24
- NVIDIA Tesla C1060
- CUDA 2.2
- SQLite 3.6.22

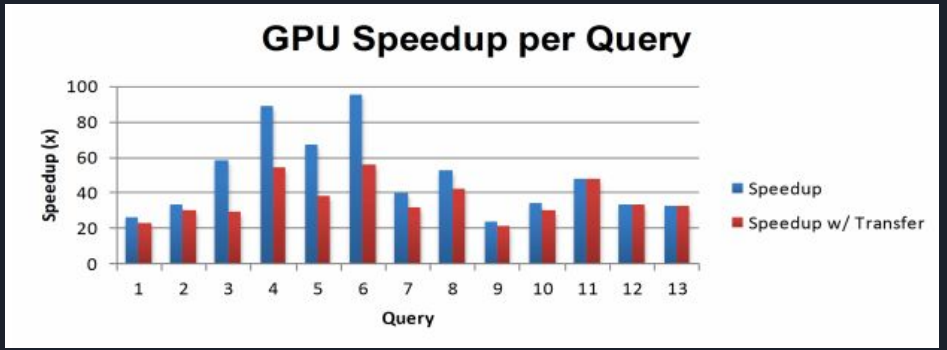
Generación de Base de Datos

- Se proporciona un programa en C que genera Datos para la Base de Datos (No Funciona)
- Implementación de generación Base de Datos en Python con SQLite3
- Schema Base de Datos (Tabla Test)
  - id INTEGER PRIMARY KEY
  - uniformi INTEGER
  - normali5 INTEGER
  - normali20 INTEGER
  - uniformf FLOAT
  - normalf5 FLOAT
  - normalf20 FLOAT

Experimentos (Consultas)

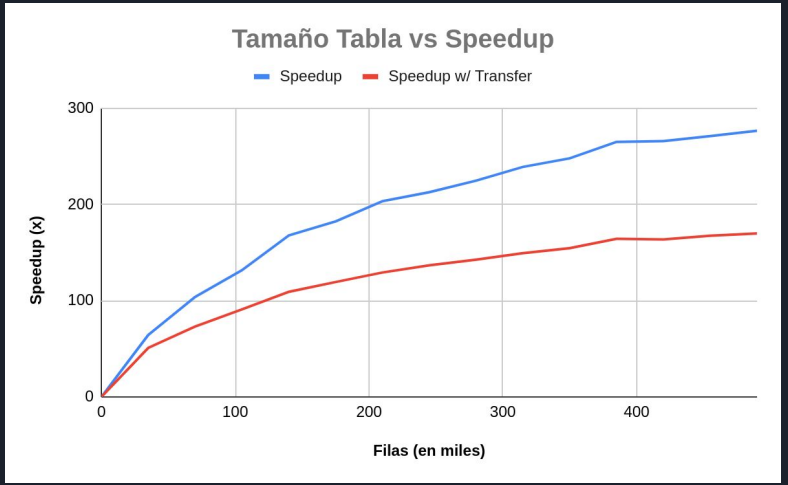
- 13 Consultas
  - 1, 3, 5, 7 y 9 son consultas a enteros.
  - 2, 4, 6, 8 y 10 son consultas a floats.
  - 11, 12 y 13 son Aggregation
- 100.000 Filas
- Comparación Nuevo vs Antiguo:

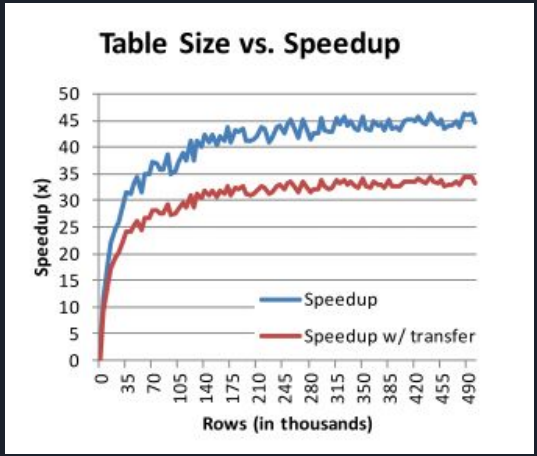




Experimentos (Tamaño Tabla vs Speedup)

- Se tomaron datos de tablas con tamaño cada 35.000 filas.
- Promedio de todas las consultas.
- Comparación Nuevo vs Antiguo:





Conclusiones

- El Speedup a las consultas con Float es mayor.
- El Speedup tomando en cuenta la transferencia baja mucho en comparación a sin tomar en cuenta la transferencia.
- Con Hardware y Software nuevo el Speedup es mayor.
- La proporción del Speedup con transferencia y sin transferencia se mantiene muy parecida a través de los años.
- Se tuvo que hacer varios cambios para correr el código.

Bibliografía

- Accelerating SQL Database Operations on a GPU with CUDA
  https://pbbakkum.com/db/bakkum.sql.db.gpu.extended.pdf

- Código fuente de librería Sphyraena.
  https://github.com/bakks/sphyraena

Gracias

Demo