Aplicación Paralela para Entorno Empresarial Usando GPGPU

José Roberto Aguilera Angulo
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Baz, B.C.S., México
jroagu@hotmail.com

Marco Antonio Castro Liera
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Baz, B.C.S., México
mcastro@itlp.edu.mx

Iliana Castro Liera
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Baz, B.C.S., México
icastro@itlp.edu.mx

Jesús Antonio Castro
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Baz, B.C.S., México
jcastro@itlp.edu.mx

***Abstract*— En el presente artículo se describe la implementación de una técnica de paralelismo aplicada al procesamiento de información financiera almacenada en una base de datos relacional, mediante Unidades de Procesamiento Gráfico de Propósito General (GPGPU, por sus siglas en inglés), con lo que se obtuvo una reducción del tiempo de cómputo requerido para el cálculo de intereses financieros.**

Palabras Clave*— *CUDA, GPGPU, CPU, Bases de Datos, Cálculo de Intereses

I. INTRODUCCIÓN

Es evidente que las organizaciones crecen y cambian sus necesidades constantemente, esto debido a múltiples factores que afectan su operación. Mantener al día la información y los procesos de negocio ha sido y seguirá siendo tema de estudio en el área de sistemas de cómputo; los nuevos retos que el mercado impone y los adelantos tecnológicos hacen que los sistemas de negocios requieran evolución; la necesidad imperante de las empresas de no sólo contar con información exacta sino pertinente, hace la diferencia entre una empresa exitosa que puede tomar oportunamente una buena decisión y una empresa del pasado. Desarrollar sistemas capaces de obtener respuestas en el menor tiempo posible ayuda a la organización a solventar estos retos.

Las aplicaciones de negocios tradicionales son excelentes generadores de información, misma que en la mayoría de las organizaciones se almacena en bases de datos. En ocasiones en necesario realizar cálculos exhaustivos sobre dicha información y es aquí donde los motores de bases de datos resultan poco eficientes. La tarea de optimizar estos cálculos mediante Unidades de Procesamiento Gráfico de Propósito General implica poder extraer estos datos de la bases de datos, procesarlos y posteriormente escribir los resultados correctos, para que la organización pueda continuar operando con estos nuevos resultados.

En el sector financiero existe un gran número de oportunidades para llevar a cabo optimización de procesos. En este artículo se explica cómo es posible implementar el cálculo de intereses diarios a un universo de clientes almacenados en una base de datos de negocio, generando nuevos saldos para las cuentas de estos clientes, sin mantener bloqueada la base de datos por tiempos prolongados y garantizando la operatividad de la organización.

Actualmente no existe mucho trabajo sobre la manipulación de bases de datos mediante GPGPU de aquí la iniciativa propuesta en el presente artículo.

A continuación se describe cómo las GPGPUs y las nuevas estrategias de programación pueden ayudar a las organizaciones a procesar el creciente volumen de información de forma adecuada, obteniendo con esto: reducción de tiempo de procesamiento de volúmenes considérales de datos, disminución de tiempo de bloqueo de las bases de datos de negocio, reducción de consumos eléctricos y generación de resultados de forma correcta para la toma oportuna de decisiones.

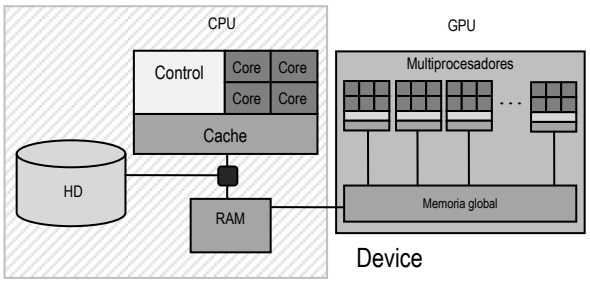
II. PARALELIZACIÓN MEDIANTE CUDA Y GPGPU

Pese a que cualquier algoritmo que es implementable en una CPU lo es también en una GPU, dado que ambas siguen el modelo de la máquina de von Neumann/Turing, esas implementaciones no serán igualmente eficientes. Los algoritmos con un alto grado de paralelismo, sin necesidad de estructuras de datos complejas y con una alta intensidad aritmética, son los que mayores beneficios obtienen de su implementación en la GPU. [1]

En la figura 1 se ilustran los componentes de la arquitectura utilizada, entre los cuales destacan la CPU encargada de iniciar

la ejecución del programa en CUDA, así como la coordinación y el intercambio de datos entre los distintos tipos de memoria y la GPU encargada de la ejecución de las instrucciones que se procesan en paralelo.

Fig. 1. Interconexión de componentes y diferencias entre la CPU y la GPU



Las diferencias más notables entre la CPU y la GPU son: el número de núcleos (Cores en inglés), que es mayor en la GPU (de aquí su alta capacidad de multiprocesamiento) y la limitación de memoria de la GPU en comparación con la alta disponibilidad de la misma en la CPU.

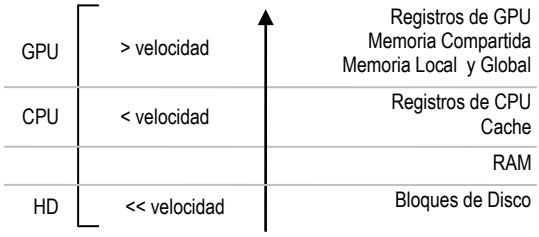
Para poder procesar en la GPU un dato almacenado en el disco duro (HD por sus siglas en inglés) es necesario que la CPU realice una lectura al disco y almacene la información en memoria RAM en arreglos lineales se copia la información hacia la memoria de la GPU para que esta pueda realizar el procesamiento. Finalmente los datos procesados son copiados nuevamente a memoria RAM en la CPU quien se encarga de rescribirlos en el disco duro.

A. GPGPU

Las unidades de procesamiento gráfico de propósito general GPGPU (o simple mente GPU) son un coprocesador dedicado al procesamiento de gráficos u operaciones de punto flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y/o aplicaciones 3D interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la unidad central de procesamiento puede dedicarse a otro tipo de cálculos. Lo anterior no implica que este dispositivo esté limitado al procesamiento de imágenes, de aquí la propuesta del presente artículo.

Uno de los principales inconvenientes de esta tecnología es el acceso a memoria, más aun si la información a procesar se encuentra almacenada en ubicaciones a las cuales el dispositivo no tiene acceso de forma automática. En la figura 2 se describe la jerarquía de acceso a memoria en este esquema de trabajo.

Fig. 2. Interconexión de componentes y diferencias entre la CPU y la GPU



B. CUDA

La arquitectura CUDA (Compute Unified Device Architecture) surgió en 2006, cuando NVIDIA lanzó sus tarjetas GeForce 8800 GTX las que por primera vez incluyeron elementos dirigidos específicamente a posibilitar la solución de problemas de propósito general. Poco después, NVIDIA lanzó el compilador CUDA C, el primer lenguaje de programación para desarrollar aplicaciones de propósito general sobre una GPU, con la finalidad de captar la mayor cantidad de programadores posibles que adoptasen esta arquitectura. CUDA C es un lenguaje muy similar a C, al cual se le agregaron un conjunto de instrucciones que harían posible la programación en paralelo en un sólo equipo de cómputo. [2,3]. En la arquitectura de CUDA, se visualizan los siguientes elementos:

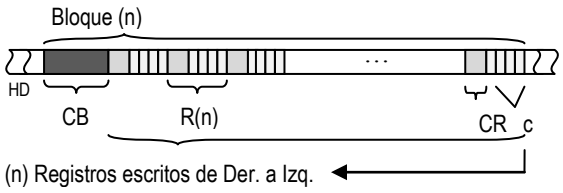
- Threads, hilos: Donde se ejecutan las funciones. Los hilos pueden procesarse simultáneamente y cuentan con memoria local para almacenar sus datos.
- Blocks, bloques: Un conjunto de hilos que se ejecutan en un multiprocesador. Cada bloque cuenta con una zona de memoria, denominada memoria compartida, accesible para todos sus hilos.
- Grid, malla: Un conjunto de bloques. Cada malla se ejecuta sobre una GPU distinta y cuenta con memoria accesible para todos los bloques que la componen. [3,4]

III. OPTIMIZACIÓN DEL ACCESO A DISCO.

Al ser el disco duro un dispositivo de acceso secuencial, poco se puede hacer para que su funcionamiento sea más veloz, ya que paralelizar su acceso es imposible. Sin embargo, es posible hacer más eficiente el proceso usando una estrategia de accesos por bloques y no por registros.

Debido a que la información de una organización debe perdurar en el tiempo, se hace imprescindible el uso este tipo de dispositivos de almacenamiento. Las bases de datos y las aplicaciones de negocios son las encargadas de: acrecentar, mantener y manipular los archivos en el disco. Debido a la complejidad natural de toda base de datos de propósito general, se hace notoria la ineficiencia de estas para realizar cálculos exhaustivos con grandes volúmenes de datos. Por ejemplo, el proceso de cálculo de intereses financieros tradicional que consiste comúnmente en recorrer la base de datos de manera secuencial por todos los registros de los clientes a los que les aplicará una serie de operaciones aritméticas para determinar los saldos de las cuentas. Dependiendo del número de clientes, esto puede resultar ineficiente produciendo un retraso del proceso. En la figura 3 se ilustra como la base datos construida en PostgreSQL 9.1 almacena realmente los datos en disco.

Fig. 3. Bloques de datos en disco en PostgreSQL 9.1



Un bloque o página (8K / 8192 bytes) está compuesto por los siguientes elementos: [5]

CB: Cabecera de bloque de 24 bytes de longitud.

R: Registro, constituido por la cabecera de elemento más el conjunto de datos en sí; este elemento es de tamaño variable.

CR: Cabecera del registro de tamaño fijo 23 bytes; va incluida en cada registro.

C: Campos del registro, de tamaño y cantidad variables; el tamaño de estos elementos depende del tipo de datos definido para cada columna en la tabla.

Cada tabla creada en la base de datos utiliza un archivo autónomo con un identificador numérico y sin extensión. Si una tabla o índice llegasen a ser mayores que 1GB, se divide a nivel del sistema en archivos con un máximo de 1GB cada uno, conservando el mismo identificador seguido de un índice.[6]

Un determinado número de registros (R) es introducido en cada bloque de tamaño fijo de 8192 bytes. Se reserva esta cantidad de espacio en disco, independientemente de que la información a almacenar sea de un tamaño menor. Los registros se escriben en orden inverso, es decir desde el final del bloque hacia la cabecera del mismo hasta llenarlo. Cada nuevo bloque es colocado al finalizar el anterior, consecutivamente.

A. Estrategia de optimización del acceso a disco.

Dado que CUDA y la GPU no pueden tener acceso directo a disco ni al manejador de la base de datos, es necesario detener el servidor de base de datos para poder manipular su información; de aquí que se requiera una comprensión clara del sistema de archivos de la base de datos, de las operaciones que se efectuaran en ellos y del funcionamiento de las aplicaciones de negocio que hacen uso de la información, ya que se pueden producir daños importantes que impidan la operabiliad de la base de datos.

A diferencia del acceso secuencial registro por registro que realiza el manejador de la base de datos al archivo objetivo, la estrategia propuesta en este artículo plantea la posibilidad de leer y escribir n bloques con un solo acceso a disco, del tal forma que se disminuya el número de interrupciones por lectura y escritura, logrando con esto la optimización de acceso y permitiendo que al leer n bloques se puedan procesar registros de forma paralela por el programa en CUDA. En las figuras 4 y 5 se ilustra lo anterior.

Fig. 4. Programación lineal en el manejador.

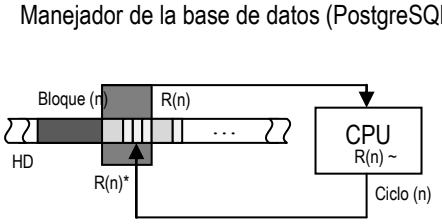
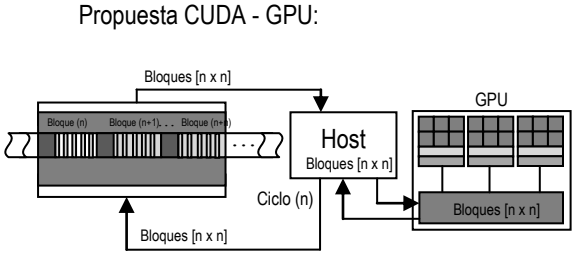


Fig. 5. Programación paralela propuesta que se basa en la optimización del acceso a disco y el cálculo paralelo de bloques de registros.

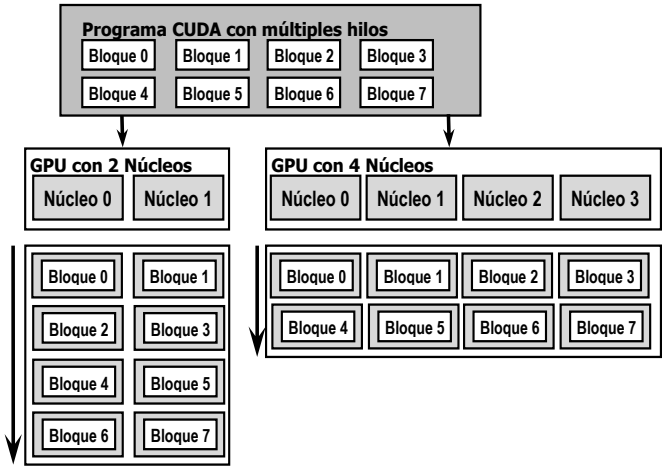


IV. OPTIMIZACIÓN DEL CÁLCULO DE INTERESES

Dado que el conjunto de operaciones a aplicar para el cálculo de intereses financieros diarios [7,8] es el mismo para el universo de clientes y considerando que el resultado de un individuo no es entrada para otro, se puede decir que esta es una tarea embarazosamente paralelizable. Con base en la anterior premisa es posible dividir el trabajo entre el número de multiprocesadores de que dispone la GPU.

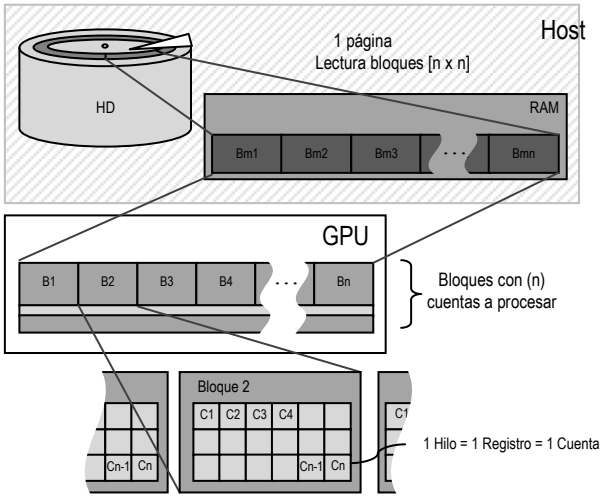
Esta división de trabajo involucra realizar una distribución equitativa en función del número de núcleos de la GPU, como se muestra en la figura 6. Lo anterior con el propósito de que todos los multiprocesadores puedan terminar al mismo tiempo y así evitar retrasos debido a sobrecarga. [4].

Fig. 6. Distribución de bloques.



En la implementación propuesta se dividió el trabajo de acuerdo al número bloques leídos del disco, de tal forma que un bloque en el disco sea un bloque en memoria RAM y también sea un bloque para la GPU. La GPU procesa una página leída del archivo de la base de datos en cada bloque y cada hilo en el bloque procesa un registro. Ver figura 7.

Fig. 7. Implementación propuesta.



A. Seudocódigo.

```
Inicio CUDA
 Lectura de archivo (producto)
    Ciclo (n productos)
        Determinación de factores
        Preparación de arreglo en memoria RAM
    Fin Ciclo
 Copia arreglo CPU → GPU

 Ciclo (n clientes)
       Lectura de (n) bloques de los archivos (saldos y
      vencimientos)
        Preparación de arreglos en memoria RAM
        Copia arreglos CPU → GPU
              Invocación función Cálculo  GPU
        Copia arreglos GPU → CPU
        Escritura de archivo (Saldos)
  Fin Ciclo
Fin CUDA.
```

```
Invocación función Cálculo  GPU

Inicio GPU
     Determinación de días trascurridos (Fecha actual
     - fecha límite de pago)
    Determinación de días en mora si fuese el caso
    Cálculo de interés a los días determinados
     Cálculo de interés mora a los días determinados si
    fuese el caso
    Cálculo de saldo de la cuenta
    Asignación de resultados a memoria global GPU
Fin GPU
```

V. RESULTADOS

Se realizaron pruebas con la implementación de ambas estrategias: programación lineal en la manejador de la base datos de PostgreSQL 9.1 y CUDA – GPU. Se utilizó la plataforma Windows de 64 bits y la librería CUDA 5, un procesador Intel Core i5-3210M CPU @ 2.50Ghz, 6 GB de memoria RAM, una tarjeta gráfica NVIDIA GeForce GT 640M LE con 1GB de memoria global.

Se logró manipular satisfactoriamente los archivos de PostgreSQL 9.1, mediante el algoritmo en CUDA, garantizando la integridad de los datos así como la operabiliad de la base de datos.

Como se muestra en la tabla I se consiguió la optimización tiempo de procesamiento para 200,000 registros, obteniendo los mismos resultados para los saldos de las cuentas.

TABLA I. RESULTADOS

Estrategia	*Tiempo (s)*
PostgreSQL 9.1 CPU	5097.183
CUDA GPU	1.000051

La implementación de esté algoritmo en la GPU demostró ser 5000 veces más veloz que la solución programada en la base de datos.

VI. CONCLUSIONES

Los resultados obtenidos comprobaron la disminución del tiempo requerido para calcular los intereses de 200,000 cuentas, lo que demuestra que un cambio en la estrategia de programación puede ayudar a optimizar un gran número de recursos y tomar decisiones oportunas.

También se demostró la aplicabilidad de GPGPUs en aplicaciones distintas a cálculos científicos y de procesamiento de imágenes, así como la posibilidad de procesar datos almacenados en archivos de bases de datos y garantizar su posterior operabilidad.

REFERENCIAS

[1] Brenes A., (2010) Arquitectura de Computadores II GPU, ASSENMBLA 19 páginas, https://www.assembla.com/spaces/bMYNBGrV8r34MWeJe5aVNr/documents/bcpxh4wdCr37VreJe5cbCb/download/bcpxh4wdCr37VreJe5cbCb

[2] NVIDIA Corp., (2012) CUDA C Programming Guide., NVIDIA Corp. 173 páginas, Santa Clara California. Version 4.2

[3] Sanders, J. Y Kandrot, E. (2010) Cuda by Example. An Introduction to General-Purpose GPU Programming. 1 Ed. Ann Harbor, Michigan, Pearson, 290 páginas.

[4] Castro, I., (2011) Paralelización de Algoritmos de Optimización Basados en Poblaciones por Medio de GPCPU, Tesis de A Maestría, Instituto Tecnológico de La Paz, 112 páginas B.C.S., México.

[5] Martínez, R., (2011) ¿Dónde están nuestros datos en el disco?, PostgreSQL-es, Centro de servicios de tecnologías de la información" (USIT) de la Universidad de Oslo, Oslo Noruega.

[6] The PostgreSQL Global Development, (1996-2013) PostgreSQL 9.1.9 Documentation, University of California, http://www.postgresql.org/docs/9.1/interactive/index.html

[7] CINIF Consejo Mexicano para la Investigación y Desarrollo de Normas de Información Financiera (2006), Normas de información financiera, páginas 832, México.

[8] Aching, C., (2002) Matemáticas Financieras para Toma de Decisiones Empresariales. 5 Ed. Editorial Serie Mypes, 306 páginas.