Generación de tabla de frecuencias para compresión por árbol de contexto con GPGPU

Vejar Romero, Ángel César
Maestría en Sistemas Computacionales
ITLP
La Paz, B.C.S, México
vectangel@gmail.com

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

***Resumen*— En este trabajo se presenta la implementación del algoritmo de compressión de datos de Huffman sobre la plataforma CUDA. Este algoritmo es ampliamente utilizado en el mundo de la criptografía por su alto nivel de compresión. Se describen los procesos que se llevan a cabo para la compressión. Además, se describe el procedimiento generación de tabla de frecuencias paralelizado mediante la GPU y secuencial haciendo comparativas de tiempo de ejecución con archivos de diferentes tamaños.**

I. Introducción

La compresión de datos se encuentra muy inmersa en la vida diaria de las personas, todos aprovechan los beneficios que brinda, ya sea tomando una fotografía, escuchando música o haciendo un respaldo de documentos. Desde el enfoque de las Ciencias de la Computación, la compresión de datos es la recodificación de cierta cantidad de información para que ocupe un menor espacio de almacenamiento que en su forma original.

En términos generales existen dos tipos de compresión, sin pérdida de información y con pérdida de información.

- Compresión con pérdida: cuando los datos son procesados se toma en cuenta uno o más criterios para poder prescindir de información que se encuentra albergada en el archivo original y que por su naturaleza no es importante su permanencia ya que con o sin ella la apreciación seria casi la misma que si no estuviera comprimido. Esto se presenta bastante en archivos multimedia donde son eliminados ciertos sonidos inaudibles al oído humano como es el caso de los archivos *.mp3, o calidad de imagen para su visualización más rápida en internet.
- Compresión sin pérdida: todos y cada uno de los datos que componen el archivo original son imprescindibles, ya que el archivo descomprimido debe ser idéntico al archivo original; Este método se usa en archivos como bases de datos o textos.

Dentro de los algoritmos sin pérdida, existen dos grandes familias:

- RLE (Run Lenght Encoding): Un algoritmo de funcionamiento sencillo que se encarga de tomar las secuencias de datos repetidas y cambiarlas por un único valor más el recuento. Tienen la ventaja de ser más rápidos que los algoritmos que usan los arboles de contexto, sin embargo, no aseguran la mejor tasa de compresión.
- Árbol de contexto: Asegura una codificación de longitud mínima pero requiere más tiempo dado que hace un doble barrido de los datos, uno para crear la tabla de frecuencias que da origen al árbol y otra para recodificar.

En este trabajo, nos enfocamos en una versión en paralelo del algoritmo de Huffman, que utiliza un árbol de contexto. Este algoritmo, asegura la recodificación de longitud mínima de la información a costa de hacer dos lecturas de los datos originales. Se pretende, por medio del procesamiento paralelo, disminuir el tiempo que requiere el algoritmo para comprimir, lo cual es su principal desventaja.

Las tecnologías de procesamiento paralelo han alcanzado todas las esferas actuales de la computación. Desde las computadoras más poderosas del planeta, incluidas en el Top500 [1], hasta los teléfonos inteligentes confían en la capacidad de sus procesadores de ejecutar más de una instrucción de manera concurrente.

CUDA es una arquitectura de procesamiento paralelo que aprovecha la potencia de las GPU de NVIDIA la cual brinda a los desarrolladores de aplicaciones de diversas áreas del conocimiento como ingeniería, investigación, genética, etc. más recursos para realizar computación de propósito general sobre procesadores gráficos (GPGPU); incrementando el rendimiento de las aplicaciones.

Algunos trabajos anteriores han propuesto una manera de manipular la creación del árbol de contexto dividiendo el archivo de entrada para que cada unidad de procesamiento cree su propio árbol de contexto basándose en el segmento de datos que le fue asignado; esto haría que cada segmento fuera comprimido al máximo, haciendo que la secuencia de datos final también estuviera comprimida al máximo [2,3]. Sin embargo eso hace que el proceso de manipulación de cada segmento ya sea para comprimir o descomprimir, tenga que pagar el precio de utilizar un poco más de memoria para poder guardar la información para recuperar los árboles de contexto.

ISBN: 978-607-97128-5-3

II. El código de Huffman

David Albert Huffman (9 de Agosto de 1925 – 7 de Octubre de 1979) fue el creador del popular algoritmo en el año 1951 en el Instituto de Tecnología de Massachusetts (MIT), el cual, fue descrito en su trabajo "*A Method For the Construction Minimum-Redundancy Codes*" [2].

El primer paso que realiza el algoritmo para lograr la compresión de información es realizar la primera lectura del archivo que se desea comprimir para realizar el conteo de los caracteres que posee. Una vez que ha creado la tabla de frecuencias se ordena de menor a mayor y se crea una lista ligada que posea los elementos de la tabla, la cual servirá para crear el árbol de contexto. Se procede a crear un nodo que contenga la suma de las frecuencias de los primeros dos elementos de la lista creando un árbol binario entre el nodo recién creado (como padre) y los dos primeros elementos (como hijos), después, se ordena el mini-árbol como se ve en la figura 1, este paso se repite tantas veces como sea necesario hasta que la lista tome forma de árbol binario. A este árbol se le conoce como árbol de contexto de longitud de descripción mínima (*DML context tree*) [4].

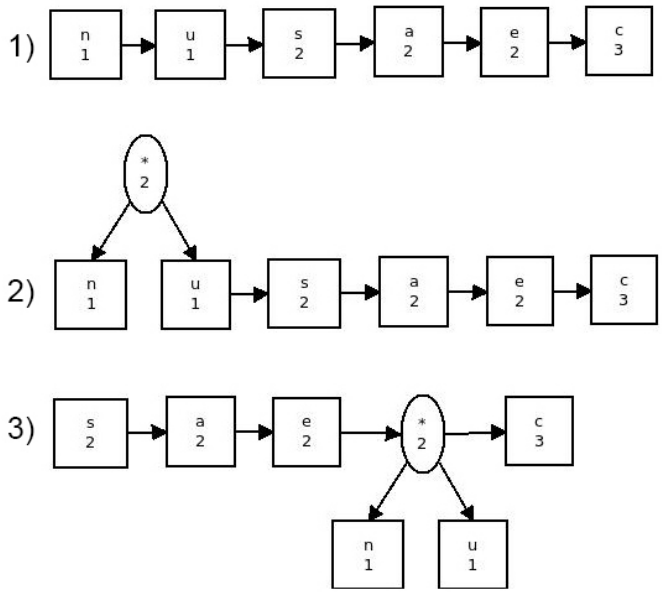

Fig. 1. Creación de árbol DML.

Una vez creado el árbol se realizan varios recorridos desde la raíz hasta cada una de las hojas del mismo donde se encuentran los distintos caracteres que conforman el archivo de entrada con el objetivo de crear una tabla que posea los códigos de longitud variable (*VLC Variable Lenght Code*) como se muestra en la figura 2. La característica que posee esta tabla es que a los símbolos que tienen una frecuencia más alta se les asigna un código más corto y a los que tienen menos frecuencia un código más largo.

Una vez que se tiene el árbol, se realiza la segunda lectura de los datos a comprimir y se hace la sustitución del símbolo por la del código de longitud variable que le corresponde creando un archivo que tiene un tamaño menor que el original.

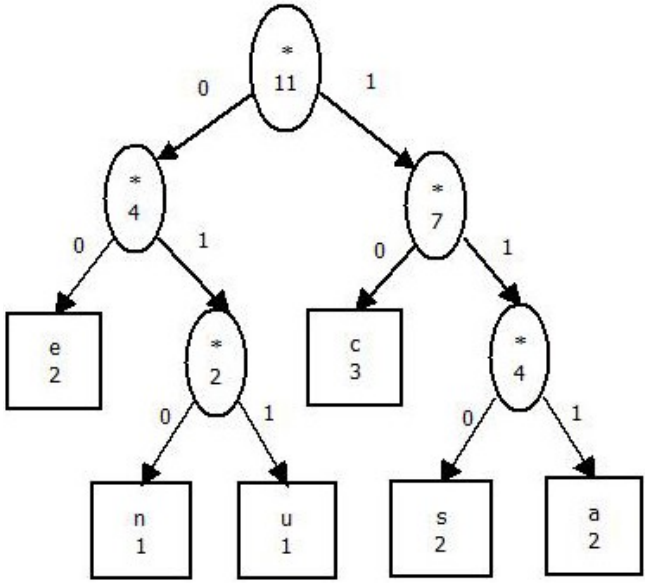

Fig. 2. Árbol VLC.

TABLA I. EJEMPLO DE TABLA DE FRECUENCIAS DE CÓDIGO VARIABLE (VLC)

Código ASCII	*Simbolo*	*VLC*
101	e	00
110	n	010
117	u	011
99	c	10
115	s	110
97	a	111

El proceso de descompresión inicia leyendo los datos comprimidos y la información de la tabla de frecuencias que fue previamente guardada en el proceso de compresión con el fin de recrear el árbol de contexto. Cuando se posee nuevamente el árbol, se realiza el proceso inverso que es cambiar el código de longitud variable por el símbolo original.

III. CUDA

Compute Unified Device Architecture (CUDA) es el nombre de la arquitectura desarrollada por la empresa NVIDIA para las tarjetas gráficas de su marca que tiene como propósito hacer accesible la tecnología para implementar cómputo de propósito general por medio de lenguajes de alto nivel.

CUDA brinda los beneficios de los lenguajes de alto nivel como son C, OpenCL, Fortran y C++ a los que se agregan algunas instrucciones sencillas. Las GPU están conformadas por varios multiprocesadores y unidades de memoria que son capaces de manipular información con los núcleos de cómputo que poseen. Los núcleos de cómputo (CUDA *Cores*) se

ISBN: 978-607-97128-5-3

agrupan en bloques y se encargan de ejecutar las instrucciones por medio de hilos; estas instrucciones se pueden categorizar en las de tipo SIMD (*Single Instruction Multiple Data*) donde cada unidad de procesamiento realiza la misma función sobre datos diferentes [5-8].

Los elementos principales de CUDA son:

Host, es el nombre que se le da a la computadora donde se encuentra instalada la tarjeta gráfica, cuando una sección de código se ejecuta del lado del *host*, se entiende que es manejada por la CPU (Unidad Central de Procesamiento).

Device, con este nombre se conoce a la tarjeta gráfica, se entiende que cuando una función se ejecuta del lado del *device*, es manejada por la GPU (Unidad de Procesamiento Gráfico).

Kernel, son funciones que se ejecutan del lado del *device* y equivalen a un hilo de ejecución manejado por un CUDA *core*.

Bloque, conjunto de hilos que son procesados en el mismo multiprocesador.

IV. Estrategia de paralelización

La estrategia que se siguió para realizar la fase de conteo fue dividir el archivo a comprimir en el número total de hilos disponibles en la tarjeta (el producto del número de bloques por el número de hilos en cada bloque) para que a cada hilo de ejecución se le asigne la misma cantidad de bytes a manipular, como se muestra en la figura 3. Como no en todos los casos se pueden tener todos los hilos con la misma cantidad de carga de trabajo en el peor de los casos un hilo trabajaría con menos carga que el resto, de tal suerte que no los retrase.

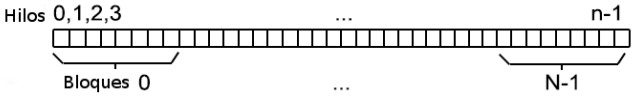

Fig. 3. Distribución de memoria por bloques e hilos.

V. Generación de la tabla de frecuencias en paralelo

Al inicio de la ejecución se obtiene la longitud total de caracteres a procesar y esta cantidad se copia a una variable global del *device* junto con el contenido del archivo. Del lado del *host* se crea un arrelgo de enteros de dimensión 256 (los posibles valores de cada byte) donde se guarda el conteo total de cada símbolo y del lado del *device* se crea un arreglo que tiene dimensiones del número de bloques por el número de hilos en cada bloque por 256, para, en primera instancia, realizar un conteo parcial.

Cuando se ejecuta el *kernel* que realiza los conteos, a cada hilo se le asigna una porción del archivo cuya longitud se calcula mediante (1), donde *l* es la longitud en bytes de la porción a contabilizar por cada hilo, *f* es el tamaño total del archivo, *b* el número de bloques con que se ejecutó el programa y *t* el número de hilos de cada bloque. De esta forma, cada hilo genera su propia tabla de conteo parcial.

$$l = ceil\left(\frac{f}{b * t}\right) \quad (1)$$

Una vez que se ha concluido la ejecución de todos los hilos de la primera fase, en un segundo *kernel*, que se ejecuta con un bloque y 256 hilos se realiza la sumatoria de cada una de las tablas de conteos parciales, como lo muestra la figura 4. Primero se suman las tablas de los hilos por cada bloque y después las de cada uno de los bloques, ubicando el conteo total en la primera tabla del arreglo de sumas parciales.

Finalizados los dos *kernels*, debido a la diferencia de memoria entre las variables del host y del *device* se copia sólo el segmento de memoria que tiene el conteo total y se aloja en un arreglo del lado del *host*.

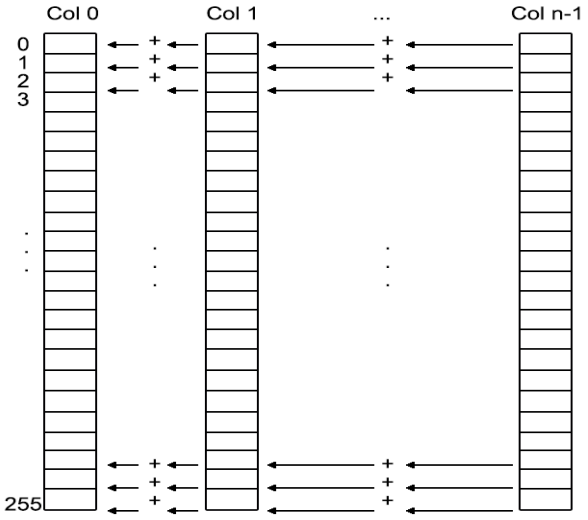

Fig. 4. Sumatoria de conteos parciales por hilo.

VI. Pruebas

Las pruebas se llevaron a cabo en una computadora que tiene las siguientes características:

TABLA II. Configuración del equipo de pruebas.

Nombre	***Descripción***
Sistema operativo	Linux Mint 17.3 Cinnamon 64 bit
Procesador	Intel Core™ i7-4790 CPU @ 3.60 GHz x 8
Disco duro	Kingston (240GB) estado sólido SSD NOW
Tarjeta de gráfica	GTX Titan X (12 GB)
Placa base	Gigabyte Technology Co.
Version de gcc	Version 4.8.4
Version de CUDA	7.5

Se realizaron conteos con diversos tamaños de archivos, las tablas III y IV, muestran los resultados obtenidos para archivos entre 100 y 800 megabytes y entre uno y dos gigabytes respectivamente.

ISBN: 978-607-97128-5-3

TABLA III. TIEMPOS DE EJECUCION EN SECUENCIAL Y CON GPU.

	Tiempo		
Tamaño(MB)	***Secuencial(s)***	***GPU(s)***	***Aceleración(s)***
100	1.35	0.23	5.87
200	2.46	0.33	7.45
400	4.89	0.50	9.78
600	7.20	0.69	10.43
800	9.51	0.88	10.81

TABLA IV. TIEMPOS DE EJECUCION EN SECUENCIAL Y CON GPU.

	Tiempo		
Tamaño(GB)	***Secuencial(s)***	***GPU(s)***	***Aceleración(s)***
1.0	12.17	1.13	10.77
1.2	14.59	1.39	10.50
1.4	17.09	1.70	10.05
1.6	19.52	2.05	9.52
1.8	21.77	2.26	9.63
2.0	24.36	2.60	9.37

VII. CONCLUSIONES Y TRABAJO FUTURO

Se puede observar que los tiempos de ejecución obtenidos en las pruebas de conteo, para comparar la versión secuencial y la paralela, son prometedores. Al aumentar el tamaño del archivo, se incrementa la diferencia entre los tiempos, consiguiéndose una aceleración cercana a 10 veces menos tiempo en paralelo que en secuencial.

Hasta el tamaño de 2.0 GB, se puede constatar que el tiempo de transferencia de los datos del *host* al *device*, no constituyo un cuello de botella para la fase de generación de la tabla de frecuencias.

Cabe señalar que en la fase de recodificación no será necesario hacer una nueva transferencia del *host* al *device*, dado que los datos permanecen en la memoria de la tarjeta gráfica.

Como trabajo futuro se desarrollará el módulo de la recodificación de la información tanto para compresión como para descompresión de datos.

VIII. REFERENCIAS

[1] Top500, “TOP500 Supercomputing Sites,” *http://www.top500.org/*. 2010.

[2] N. Krishnan and D. Baron, “A universal parallel two-pass MDL context tree compression algorithm,” *IEEE J. Sel. Top. Signal Process*, vol. 9, no. 4, pp. 741–748, 2015.

[3] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[4] M. Nelson and J. Gailly, T*he Data Compression Book*, 2nd edition. 1995.

[5] J. Sanders and E. Kandrot, *CUDA by Example*, 1st ed., vol. 54, no. 2. Boston, MA, USA: Addison-Wesley Professional, 2010.

[6] J. Cheng, M. Grossman, and T. KcKercher, *Professional CUDA C programming*, Wrox, 2014.

[7] D. Kirk and W. Hwu, *Programming massively parallel processors : a hands-on approach*, 3rd ed. Morgan Kaufmann, 2016.

[8] N. Wilt, *The CUDA handbook : a comprehensive guide to GPU programming*, 1st ed. Addison-Wesley Professional, 2013.