Implementación en un FPGA del Modelo de Compresión de Datos PPMC

Virgilio Zúñiga-Grajeda, Claudia Feregrino-Uribe

Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica, Óptica y Electrónica, INAOE.
e-mail: virgilio@ccc.inaoep.mx, cferegrino@inaoep.mx
Luis Enrique Erro #1 Apdo Postal 51 y 216, CP 72000, Tonantzintla, Puebla México.

Abstract: *Electronic comunications and computer systems require lossless data compression techniques to reduce the amount of data to transmit or store. Compression algorithms like PPM (Prediction by Partial Matching) provide good compress ratios but present speed limitations [1]. We propose a hardware implementation of the PPMC algorithm [2] in an FPGA to increase compression speed.*

Resumen: *Las comunicaciones electrónicas y sistemas computacionales requieren del desarrollo de técnicas de compresión de datos sin pérdida para reducir el uso de grandes cantidades de datos a transmitir o almacenar. Algoritmos de compresión como el PPM (Prediction by Partial Matching) proveen una buena razón de compresión pero presentan limitaciones de velocidad [1]. Proponemos implementar en hardware [2], particularmente en un FPGA, el algoritmo PPMC para incrementarla velocidad de compresión.*

Palabras clave: PPMC, FPGA, CAM, Compresión de datos, Hardware.

1. Introducción.

Debido a los avances tecnológicos en áreas como la electrónica y la computación, la cantidad de datos a manejar ha crecido de tal forma que es preciso utilizar algoritmos de compresión que ofrezcan una buena razón de compresión y al mismo tiempo una velocidad aceptable según su aplicación. El algoritmo PPM presenta una buena razón de compresión, pero su velocidad puede llegar a ser inadecuada para algunas aplicaciones como la transmisión de datos.

El algoritmo PPM mantiene un diccionario con un modelo estadístico de los datos, asignando probabilidades a los símbolos y mandando estas probabilidades a un codificador aritmético.

El modelo estadístico en su forma más simple cuenta el número de veces que ocurre cada símbolo en el pasado y le asigna una probabilidad con base en ese número. Un modelo más complejo es el basado en *contexto*, donde no sólo la frecuencia del

símbolo es usada para predecir sino que también la frecuencia del símbolo ocurrido cuando una secuencia de símbolos lo preceden inmediatamente. Dichos símbolos son llamados *contexto* y el número de ellos es el *orden* del contexto. Por ejemplo, en la frase: "compresió" se puede predecir que el siguiente símbolo en la cadena es "n". La frase es el contexto y el orden es 9, porque es el número de símbolos que la conforman.

Como regla general, entre más grande es el orden del modelo mejor será la predicción. Desafortunadamente, la magnitud del espacio requerido para almacenar un contexto tan largo es prácticamente imposible de manejar. Aún restringiendo el contexto a 4 caracteres, habría (usando el típico byte de 8 bits) más de 4 billones de contextos posibles [3].

Se ha demostrado que implementaciones software de este tipo de algoritmos consumen gran cantidad de recursos, alrededor de 100Kb de memoria [4], además de ser lentos en comparación con implementaciones software de algoritmos más sencillos como los de la familia LZ [5,6]. Sin embargo, han presentado razones de compresión superiores a cualquiera de los algoritmos tanto de dicha familia como otros algoritmos poco complejos. Lo cual lleva a pensar que un mapeo adecuado de este tipo de algoritmos a una implementación hardware podría mejorar considerablemente la velocidad de compresión.

1.1 Funcionamiento del algoritmo.

Un modelo PPMC de orden *O* lee un símbolo *s* y considera los *O* símbolos como el contexto actual. Después busca el contexto seguido del símbolo *s*. Si el símbolo no se encuentra, el modelo "escapa" al siguiente contexto de orden más bajo *O-1* mediante la transmisión de un código de "escape". Este proceso continúa hasta que el símbolo se encuentra o el modelo alcanza el orden 0. Si el símbolo no se encuentra en el orden 0, entonces un escape final es transmitido y el símbolo *s* se predice por el orden *-1*, donde todos los símbolos tienen la misma probabilidad (1/256, donde el alfabeto tiene 256 símbolos). Por último, el diccionario se actualiza añadiendo el símbolo *s* al contexto correspondiente.

Las siguientes fórmulas se usan para calcular las probabilidades de los símbolos y del escape:

$$p(s \mid contexto) = \frac{f}{t+k} \qquad \text{y} \qquad p(esc \mid contexto) = \frac{k}{t+k} \qquad (1)$$

Donde *p(s|contexto)* es la probabilidad de que el símbolo *s* suceda dado que *contexto* ha ocurrido, *f* es la frecuencia del símbolo *s*, *k* es el número de símbolos diferentes en el contexto actual y *t* es la suma de las frecuencias de todos los símbolos en el contexto actual.

2. Arquitectura propuesta.

Una implementación hardware de este tipo de algoritmos requiere de un diseño minucioso que maximice su rendimiento y demande pocos recursos. Una forma de maximizar el rendimiento de algoritmos de compresión en hardware es usando estructuras eficientes para almacenar datos. Entre estas estructuras están los arreglos CAM (Content-Addressable Memory) que permiten la búsqueda simultánea en todas las entradas del diccionario. La ventaja de utilizar este tipo de estructuras es que el proceso de compresión puede acelerarse considerablemente.

Por simplicidad, se implementó el algoritmo PPMC de orden 0, por lo que contamos con dos diccionarios, uno para contextos de orden 0 y otro para orden -1. Se ha ideado un esquema para obtener los datos estadísticos, por medio de las mismas señales de control conectadas a la CAM como se muestra en la Fig.1.

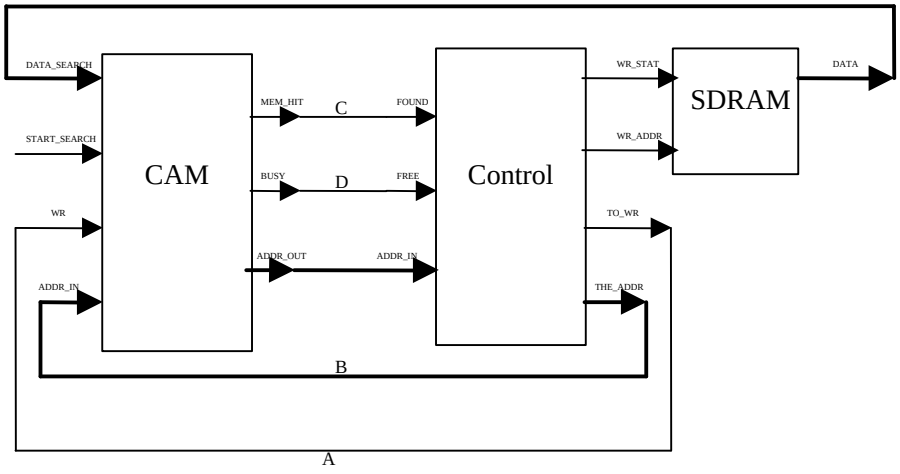

Fig.1 Arreglo CAM conectado al módulo de control.

Tenemos las siguientes consideraciones que son la base que sustentan la obtención del modelo estadístico.

- El número de veces que se activa la señal A durante la lectura de la cadena completa, corresponde a las veces que no se encontró el símbolo dado y esto coincide con la cantidad de símbolos de escape necesarios, por lo que la señal A da la cantidad total de escapes.

- La señal B de la CAM contiene la dirección que corresponde a cada símbolo encontrado si se presenta un símbolo anteriormente examinado, esta dirección permite saber cuál es el símbolo cuya frecuencia se debe actualizar.

- La señal C, que se activa cuando se ha encontrado un símbolo, indica que se debe actualizar la frecuencia.

- El número de veces que se activa la señal D es igual al número de búsquedas que se hacen en la CAM, lo que corresponde a la cantidad total de símbolos.

Los datos a comprimir se almacenan en una memoria externa SDRAM que forma parte de la tarjeta de desarrollo. Para esto se utiliza un software creado por XESS que permite escribir datos desde la PC. El sistema obtiene de la memoria SDRAM cada uno de los símbolos de la cadena a comprimir y escribe en ella las frecuencias obtenidas por el módulo de control.

3. Implementación.

La implementación se realizó en VHDL, mediante el software Active-HDL 6.2, que cuenta con un editor de formas de onda donde se puede observar con detalle las señales del diseño.

Se utilizó el software Xilinx ISE 6.1.03i para sintetizar la descripción del circuito en un FPGA Spartan-II de 2.5V y 50,000 compuertas que forma parte de la tarjeta de desarrollo XSA-50 de XESS.

4. Resultados experimentales.

Para ejemplificar el funcionamiento del circuito tomamos una cadena de caracteres. Por claridad, representamos el espacio con un guión bajo. La cadena es la siguiente:

alicia_morales_reyes

Se realiza la simulación post-síntesis del circuito y se obtienen los resultados mostrados en la Tabla 1.

Evento	Cadena	Búsqueda en orden 0	Probabilidad de escape	Probabilidad del símbolo
1	a	esc	1	1/256
2	l	esc	1/2	1/256
3	i	esc	1/2	1/256
4	c	esc	1/2	1/256
5	i	✓		1/8
6	a	✓		1/9
7	_	esc	2/5	1/256
8	m	esc	5/12	1/256
9	o	esc	3/7	1/256
10	r	esc	7/16	1/256
11	a	✓		2/10
12	l	✓		1/19
13	e	esc	2/5	1/256
14	s	esc	9/22	1/256
15	_	✓		1/25
16	r	✓		1/26
17	e	✓		1/27
18	y	esc	10/27	1/256
19	e	✓		2/30
20	s	✓		1/31

Tabla 1 Modelo estadístico correspondiente al texto “alicia_morales_reyes”.

En la primera columna se enumera cada evento, es decir, la lectura de un símbolo. En la segunda columna se muestra cada uno de los símbolos leídos de la cadena. Si el símbolo se encuentra en el orden 0, se calcula su probabilidad según las fórmulas en (1), e indicado en la tabla con ✓. Si el símbolo no se encuentra en el orden 0, se escapa de este orden enviando la probabilidad del escape y la probabilidad del símbolo en el orden -1. El modelo transmite al codificador las probabilidades de ocurrencia, en forma de frecuencias acumuladas, según se obtengan. En este caso las probabilidades son: 1, 1/256, 1/2, 1/256, 1/2, 1/256, 1/2, 1/256, 1/8, 1/9, 2/5, 1/256, 5/12, 1/256, 3/7, 1/256, 7/16, 1/256,…

Después de procesar la cadena de prueba, las probabilidades de los símbolos de acuerdo al estado actual del diccionario son las mostradas en la Tabla 2. Esta tabla se obtuvo por separado, como se hace en [7], por lo que comprobamos el correcto funcionamiento del algoritmo.

Orden = 0				Orden = -1			
Predicciones		*c*	*p*	Predicciones		*c*	*p*
?	a	3	3/31	?	?	1	1/#*L*
?	l	2	2/31				
?	i	2	2/31				
?	c	1	1/31				
?	_	2	2/31				
?	m	1	1/31				
?	o	1	1/31				
?	r	2	2/31				
?	e	3	3/31				
?	s	2	2/31				
?	y	1	1/31				
?	escape	11	11/31				

Tabla 2 Probabilidades de los símbolos después de procesar la cadena "alicia_morales_reyes".

Donde *c* es la frecuencia de cada uno de los símbolos incluyendo el escape, *p* es la probabilidad de ocurrencia de cada uno de ellos, ? representa el símbolo que no fue encontrado en el orden 0 y *L* es el alfabeto utilizado. En el orden –l se le asigna una probabilidad de 1/#*L* al símbolo ? , donde #*L* es la cardinalidad del alfabeto *L*.

Con respecto a la razón de compresión, 0.56 es lo que se obtiene con un modelo de orden 0 [2].

Los resultados de la síntesis se muestran en la Tabla 3.

Número de Slices	382 de 768	49%
Número de LUTs	659 de 1536	42%
Periodo mínimo de reloj	16.567 ns	Frecuencia Máxima 60MHz

Tabla 3 Reporte de la síntesis.

La primera búsqueda en la CAM necesita 9 ciclos de reloj. Si introducimos un documento de texto de 152,089 bytes como por ejemplo "alice29.txt" del Canterbury Corpus [8] y usamos una frecuencia de operación de 50MHz (más baja que la frecuencia máxima para evitar retrasos en las señales), el proceso tomaría .042 segundos y la velocidad de procesamiento sería 28.8Mb/s. Estos cálculos se muestran en la Tabla 4.

Para la primera búsqueda	2,304 ciclos
Para el resto de las búsquedas	3,036,666 ciclos
Para lectura/escritura de la memoria	1,825,068 ciclos
Total	2,131,038 ciclos
Total en Segundos	.042 s
Velocidad	28.8Mb/s

Tabla 4 Ciclos necesarios para procesar el archivo "alice29.txt".

5. Conclusiones.

El diseño del esquema PPMC funcionó correctamente en las simulaciones post-síntesis. Se obtuvo una buena velocidad de compresión debido a que se optimizó el proceso de búsqueda de símbolos usando arreglos de memoria CAM.
Como trabajo futuro se puede implementar un arreglo CAM más eficiente, como el IP COREs de Xilinx, el cual realiza la búsqueda en 2 ciclos de reloj. Además es posible añadir más diccionarios a la implementación y así aumentar la razón de compresión, verificando así los resultados de simulación obtenidos en [2].

Referencias.

[1] Bell, T.J. Cleary, J.G. y Witten, I.H. (1990) *"Text Compression"*, Prentice Hall, Englewood Cliffs, NJ.

[2] Feregrino Uribe, C. y Jones, S. R. (2001) *"Optimization of PPMC Model for Hardware Implementation*", Proceedings of the 2001 Euromicro Symposium on Digital Systems Design (DSD'01), IEEE Computer Society Press, p. 120 – 126, Varsovia, Polonia.

[3] Moffat (1990) *"Implementing the PPM data compression scheme"*. IEEE Transactions on Communications, Vol. 38 (11), p. 1917-1921.

[4] Hirschberg D.S. y Lelewer D.A. (1992) *"Context Modeling for Text Compression"*, Image and Text Compression, J.A. Storer, ed., Kluwer Academic Publishers, Norwell, MA, p. 85-112.

[5] Ziv, J. y A. Lempel (1977) *"A Universal Algorithm for Sequential Data Compression"*, IEEE Transactions on Information Theory, Vol. 23, No. 3, Mayo, p. 530-536.

[6] Ziv, J. y A. Lempel (1978) *"Compression of Individual Sequences by Variable Rate Source Coding"*, IEEE Transactions on Information Theory, Vol. 24, No. 5, p. 530-536.

[7] Cleary, J. Teahan, W. y Witten, I. (1995) *"Unbounded length contexts for PPM"*. In J.A. Storer and M. Cohn, editors, Proceedings DCC-95. IEEE Computer Society Press.

[8] Disponible en la página: http://corpus.canterbury.ac.nz/index.html. Verificado Mayo de 2004.