Algoritmos de suma y multiplicación basados en el modelo de etiquetas

Nelson E. Ordóñez-Guillén, Israel M. Martínez-Pérez
Departamento de Ciencias de la Computación
Centro de Investigación Científica
y de Educación Superior de Ensenada
Ensenada, B.C., México 22860
Email: {nordonez, israelmp}@cicese.edu.mx

***Resumen*—En el presente trabajo, se presentan algoritmos aritméticos basados en el modelo de etiquetas de cómputo biomolecular. Se incluye un algoritmo de suma que está basado en compuertas lógicas XOR y XNOR que no utiliza clear cuando el resultado se escribe en un área nueva en las hebras y que no registra los acarreos en las hebras, por lo que la complejidad espacial de las mismas es $O(n)$ bits. También se incluye un algoritmo de multiplicación basado en el algoritmo del granjero, que se basa en corrimientos de bits y en sumas para obtener el producto. Al final se discuten algunas consideraciones acerca de la operación clear y del registro de los bits de acarreo en las hebras.**

I. Introducción

Desde su definición en [5], el modelo de etiquetas ha sido ampliamente tratado en la literatura. Aprovechando su demostrada universalidad [4], se ha utilizado principalmente para proponer algoritmos enfocados en problemas NP-Completo. Sin embargo, en los últimos años, ha surgido el interés por el desarrollo de algoritmos aritméticos que sirvan como bloques de construcción para nuevos algoritmos enfocados en otro tipo de problemas. Por ejemplo, Chang *et al.* [2] proponen tres algoritmos biomoleculares para la implementación paralela de la resta, comparación y aritmética modular, respectivamente. Estos algoritmos son utilizados en la factorización de dos números primos grandes, con el propósito de demostrar la viabilidad del cómputo biomolecular en la ruptura del sistema de cifrado de llave pública RSA. Por su parte, Yang *et al.* [6] introducen un esquema de codificación de tres segmentos para un número real, y un algoritmo de suma basado en este esquema. Se resuelve el problema del uso de bit de acarreo y de préstamo en las hebras. Finalmente, realizan una simulación para probar la validez del algoritmo. Guo y Zhang [3] describen algoritmos para cada una de las cuatro operaciones aritméticas. A la fecha, el trabajo que implementa aritmética con el modelo de etiquetas más reciente es [1]. Sus principales contribuciones consisten en que a diferencia de trabajos anteriores, se presenta un algoritmo de suma que no incluye la operación clear, cuya implementación bioquímica es problemática. Además, no es necesario el registro de bits de acarreo, por lo que se reduce considerablemente el tamaño de las hebras. Presenta un análisis comparativo de los algoritmos previos con el algoritmo propuesto, resultando que se supera significativamente la rapidez (número de pasos) y la cantidad de tubos necesarios para el cómputo. Se propone una simplificación para el caso de adición de constantes y se revisa un caso en el que se utiliza el algoritmo en una aplicación típica de supercómputo, como es el caso del cálculo de la norma Euclidiana utilizando la representación en el Sistema Numérico Logarítmico.

En el presente artículo, se propone un algoritmo para la adición (sustracción) de dos números enteros, basado en las operaciones del modelo de etiquetas. Este algoritmo utiliza las funciones que implementan compuertas lógicas XOR [1] y XNOR. No se utiliza la operación CLEAR cuando el resultado se escribe en nuevos espacios dentro de las hebras. No se registran acarreos en las hebras, según el enfoque propuesto por [1]. También se incluye un algoritmo para la multiplicación de dos enteros que opera de la misma forma que el algoritmo del granjero. Este algoritmo realiza multiplicaciones y divisiones por un factor de 2, sucesivas en los operandos. En números representados en binario, estas operaciones consisten en simples corrimientos de bits. Con esto, la multiplicación de dos números de n bits se puede realizar mediante apuntadores que implementen los corrimientos de bits y sumas acumulativas.

II. Modelo de etiquetas

Introducido en [5], es un modelo de cómputo biomolecular que utiliza hebras de ADN como sustrato físico para almacenar información. Utiliza la operación de separación de hebras por hibridación como mecanismo central de su lógica computacional. La información se representa a través de dos tipos de hebras simples de ADN, llamadas *hebras de memoria* y *hebras etiqueta*. Una hebra de memoria se divide en n regiones, donde cada región codifica un bit y es una secuencia única en toda la hebra. A cada región, le corresponde una hebra etiqueta. Cuando una región tiene adherida su correspondiente etiqueta, codifica un 1 binario, en caso contrario, un 0. Se denomina *complejo de memoria* a una hebra de memoria con sus etiquetas. Cada complejo codifica, de esta forma, una cadena de n bits. El modelo considera conjuntos masivos de hebras (cadenas) almacenados en tubos de prueba. Típicamente, el cómputo dentro del modelo, inicia con un tubo de prueba. Sobre este tubo se realiza una secuencia de bio-operaciones (algoritmo biomolecular) que actúan sobre todas las hebras contenidas en tal tubo. Estas bio-operaciones son:

- **Separar** (dividir) un conjunto de cadenas en dos conjuntos, dependiendo el estado de un bit en las cadenas.
- **Combinar** (unir) dos conjuntos de cadenas.

- **Encender** un bit en todas las hebras de un conjunto.
- **Apagar** un bit en todas las hebras de un conjunto.
- **Deshechar** todas las cadenas de un conjunto.

Estas operaciones permiten realizar cómputo de manera universal [4].

III. Algoritmo de suma

Este algoritmo se basa en la operación de la compuerta lógica XOR. El Algoritmo 1 produce la operación $o = a$ Xor b en cada cadena del tubo de entrada T, de acuerdo a la Tabla I.

Algorithm 1: Xor(T, o, a, b)

Input: Tubo T, posiciones de bit o: salida, a: entrada 1, b: entrada 2
Output: Tubo T con operación $o = a$ Xor b en cada cadena

1 SEPARATE(T, T_1, T, a);
2 SEPARATE(T_1, T_3, T_1, b);
3 SEPARATE(T, T_2, T, b);
4 COMBINE(T_1, T_2);
5 SET(T_1, o);
6 COMBINE(T, T_3);
7 COMBINE(T, T_1);
8 Return T;

Tabla I: Tabla de verdad de la función XOR.

a	b	o
0	0	0
0	1	1
1	0	1
1	1	0

Las operaciones de separación, clasifican las cadenas del tubo de entrada T en tres grupos, dependiendo de los bits de entrada a y b: en el tubo T permanecen las cadenas con ambos bits apagados; el tubo T_3 mantiene las cadenas con ambos bits encendidos; los tubos T_1 y T_2 almacenan las cadenas con alguno de los bits encendido. Estas cadenas se unen en T_1 para encender el bit de salida o en cada una de ellas. Finalmente, las cadenas en T_1 y T_3 son regresadas al tubo de entrada T. Para evitar el uso de la operación CLEAR, este procedimiento supone que el bit de salida, se encuentra apagado al inicio, lo cual implica el uso de una región nueva en las hebras. De manera simétrica, la función Xnor se realiza con algunos cambios simples para encender el bit de salida o en los tubos T y T_3 (Los que mantienen las cadenas con un bit encendido).

El Algoritmo 2 utiliza estas funciones para implementar un sumador completo de 1 bit, de acuerdo a la Tabla II.

La función toma un tubo T con las cadenas sin acarreo de entrada ($C_{in} = 0$) y un tubo T_c con las cadenas con acarreo de entrada ($C_{in} = 1$). El tubo T_c inicialmente se encuentra vacío y solamente se utiliza para sumas sucesivas

Algorithm 2: FullAdder(T, T_c, o, a, b)

Input: Tubos T y T_c, posiciones de bit o: salida, s_1: sumando 1, s_2: sumando 2
Output: Tubo T con cadenas sin acarreo, tubo T_c con cadenas con acarreo

1 Xor(T, o, s_1, s_2);
2 Xnor(T_c, o, s_1, s_2);
3 SEPARATE(T, T_2, T, s_1);
4 SEPARATE(T_c, T_1, T_c, s_1);
5 SEPARATE(T_2, T_c, T, s_2);
6 SEPARATE(T_1, T_c, T, s_2);
7 Return T, T_c;

Tabla II: Tabla de verdad de la función sumador completo.

C_{in}	s_1	s_2	o	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

de bits, como se explicará más adelante. Para evitar el uso de CLEAR, también se supone que el bit resultado o se encuentra apagado. En la Tabla II se puede apreciar que para $C_{in} = 0$, la salida o corresponde a la función s_1 Xor s_2, y para $C_{in} = 1$ corresponde a la función s_1 Xnor s_2. Estas operaciones se realizan en las líneas 1-2. Las operaciones SEPARATE de las líneas 3 y 5, envían al tubo T_c las cadenas que estaban en T y que contienen ambos bits s_1 y s_2 encendidos. De manera similar, las operaciones SEPARATE de las líneas 4 y 6, transfieren al tubo T las cadenas que estaban en T_c y que contienen ambos bits s_1 y s_2 apagados.

Para realizar la suma de dos números de n bits, se requieren n llamadas sucesivas al Algoritmo 2 cambiando la posición de bit en cada llamada. El acarreo se registra de forma implícita, dependiendo del tubo que contiene las cadenas. El formato de las cadenas es el siguiente: n bits para el primer número (posiciones de bit $[1..n]$), n bits para el segundo número (posiciones de bit $[n+1..2n]$) y $n+1$ bits para el resultado (posiciones de bit $[2n+1..3n+1]$). Así, cada cadena es de $3n+1$ bits de longitud.

Algorithm 3: n-bitAdder(T)

Input: Tubo T con cadenas de $3n+1$ bits
Output: Tubo T con suma registrada en los bits $[2n+1..3n+1]$.

1 **for** $i \leftarrow 1$ **to** n **do**
2 | FullAdder(T, T_c, $3n+2-i$, $n+1-i$, $2n+1-i$);
3 **end**
4 SET($T_c, 2n+1$);
5 COMBINE(T, T_c);
6 Return T;

La Figura 1 muestra la secuencia de sumas de bits para una cadena particular que codifica los números 6 y 4 en tres bits. La operación SET de la línea 4, enciende el bit más significativo

i	Número 1	Número 2	Resultado	Tubo final
1	1 1 0	1 0 0	0 0 0 0	T
2	1 1 0	1 0 0	0 0 1 0	T
3	1 1 0	1 0 0	0 0 1 0	T_c

Fig. 1: Secuencia de sumas de bits para una cadena que codifica los números 6 y 4 en tres bits. Se muestran las tres iteraciones y se resaltan en rojo los bits sumados en cada iteración. Se indica el tubo en el que se almacena la hebra al final de una iteración.

Multiplicando	Multiplicador
17	23
8	~~46~~
4	~~92~~
2	~~184~~
1	368
Result→	391

Fig. 2: Ejemplo de ejecución del algoritmo del granjero para los enteros 17 y 23. En cada paso, el multiplicando se divide por 2 ignorando el residuo, y el multiplicador se duplica. El algoritmo termina cuando el multiplicando es 1. El resultado es la suma de todos los términos del multiplicador donde el multiplicando es impar.

(acarreo) de la cadenas en T_c y la operación COMBINE regresa tales cadenas al tubo final T. De esta forma, el resultado final es el número 1010 binario que representa al 10 decimal.

IV. Algoritmo de multiplicación

El algoritmo del granjero o algoritmo palurdo es una forma muy simple de realizar una multiplicación. Se denomina de esta manera puesto que no requiere el conocimiento de tablas de multiplicación, únicamente saber obtener el doble y la mitad de las cantidades, y al final sumar ciertos resultados intermedios. En este algoritmo, el multiplicando se va dividiendo por 2 e ignorando el residuo, mientras que el multiplicador se va duplicando, esto se hace mientras el multiplicando sea mayor que 1. Al final, el resultado es la suma de la columna del multiplicador, donde se consideran únicamente los términos donde el multiplicando es impar (Figura 2).

Cuando se representan en binario, las operaciones de división y multiplicación por 2, son simples corrimientos de bits, a la derecha y a la izquierda, respectivamente. Esto se puede realizar manejando "apuntadores" en la cadena de bits para señalar la posición del bit menos significativo y dejando "bits de corrimiento" para estos apuntadores. De esta forma, mover el apuntador a la derecha, significa un corrimiento a la izquierda y viceversa. En la Figura 3 se muestra el formato de las cadenas para la multiplicación de 2 números de n bits y

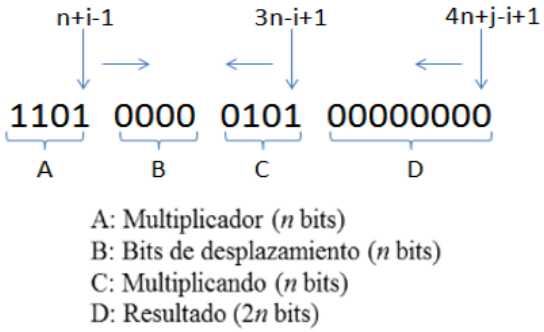

Fig. 3: Formato de las cadenas para multiplicar 2 números de 4 bits. Se indica la posición inicial de los apuntadores y la dirección en la que se mueven.

las posiciones iniciales de los apuntadores en los operandos y en el resultado, dependiendo de n y de $i = [1..n]$.

Usando el formato anterior en las cadenas de bits a multiplicar, el Algoritmo 4 implementa esta operación.

Algorithm 4: Peasant_mult(T)

Input: Tubo T con cadenas a multiplicar
Output: Tubo T con producto en los bits $[3n+1..4n]$.

```
1  for i ← 1 to n + 1 do
2      SEPARATE(T, T, T_0, 3n − i + 1);
3      for j ← n + 1 − i downto 1 do
4          FullAdder(T, T_c, 4n + j − i + 1, j,
           4n + j − i + 1);
5      end
6      SET(T_c, 4n − i + 1);
7      COMBINE(T, T_c);
8      COMBINE(T, T_0);
9  end
10 Return T;
```

El ciclo exterior maneja la posición de los apuntadores de ambos operandos, desplazando el apuntador del multiplicador un bit a la derecha (corrimiento a la izquierda, multiplicación por 2) y el apuntador del multiplicando, un bit a la izquierda (corrimiento a la derecha, división por 2), en cada iteración. La operación SEPARATE, al inicio de cada iteración del ciclo externo, deja en el tubo T únicamente las cadenas donde el bit menos significativo del multiplicando es 1 (número impar) y pone en T_0 las cadenas donde este bit es 0 (número par). El ciclo interior sólo considera el tubo T y realiza la suma acumulativa del multiplicador, y reescribe esta suma acumulada en cada iteración, bit a bit, en el área de resultado. Es importante notar que esta suma será de $n, n+1, ..., 2n$ bits, respectivamente a cada iteración del ciclo exterior, por lo que el apuntador en el resultado considera una posición más de bit en cada iteración. Las tres operaciones al final del algoritmo, son para considerar los acarreos finales en cada suma y unir los conjuntos de cadenas de T_c y T_0 en T.

V. Discusión

Aunque la definición original del modelo de etiquetas considera la operación CLEAR, también se incluyen discusiones acerca de que su implementación bioquímica es propensa a errores, sin embargo, se puede descartar del modelo sin que éste pierda su poder de cómputo. Hasta la fecha, se considera no deseable la inclusión de esta operación en los algoritmos. En el caso de los algoritmos aritméticos, es indispensable el uso de CLEAR, cuando es necesario reescribir resultados en áreas previamente utilizadas en las hebras. Por ejemplo, en el caso de la multiplicación, es necesario escribir en las hebras los $O(n)$ resultados intermedios de $O(n)$ bits, para después sumar estos resultados y escribirlos en otro espacio de $2n$ bits. Lo anterior aumenta considerablemente la longitud de las hebras resultando en un costo prohibitivo en la complejidad espacial cuando se utilizan los algoritmos en aplicaciones de supercómputo. Por otro lado, el enfoque del manejo del acarreo en la suma de [1] es difícil de superar, dado que el acarreo es indispensable en la suma y el modelo de etiquetas cuenta con pocos mecanismos para su manejo (hebras o tubos). De aquí, que su enfoque resulta óptimo, ya que no utiliza espacio en las hebras para el registro de bits de acarreo.

Agradecimientos

Este trabajo fue apoyado por el Consejo Nacional de Ciencia y Tecnología (CONACyT).

Referencias

[1] M. G. Arnold, "An improved DNA-sticker addition algorithm and its application to logarithmic arithmetic," in *DNA Computing and Molecular Programming*, vol. 6937. Springer, 2011, pp. 34–48.

[2] W.-L. Chang, M. Ho, y M. Guo, "Fast parallel molecular algorithms for dna-based computation: factoring integers," in *Bioinformatics and Bioengineering, 2004. BIBE 2004. Proceedings. Fourth IEEE Symposium on*, 2004, pp. 125–133.

[3] P. Guo y H. Zhang, "DNA implementation of arithmetic operations," in *2009 Fifth International Conference on Natural Computation*, vol. 6. IEEE, 2009, pp. 153–159.

[4] L. Kari, G. Paun, G. Rozenberg, A. Salomaa, y S. Yu, "DNA computing, sticker systems, and universality," *Acta Informatica*, vol. 35, no. 5, pp. 401–420, 1998.

[5] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund, y L. M. Adleman, "A sticker based model for DNA computation," *Journal of Computational Biology*, vol. 5, no. 4, pp. 615–629, 1998.

[6] Y.-x. Yang, A.-m. Wang, y J.-l. Ma, "A DNA computing algorithm of addition arithmetic," in *2009 First International Workshop on Education Technology and Computer Science*, vol. 1. IEEE, 2009, pp. 1056–1059.