Ciencias Matemáticas, Vol. 33, No. 2, Pag. 145-151, 2019
Recibido 04-2019, Aceptado 09-2019

Las Redes de Interconexión y La Teoría de Gráficas
Interconnection Networks and Graph Theory

María de Luz Gasca Soto*

Resumen El objetivo de este documento es dar un panorama general sobre cómo la Teoría de Gráficas resulta ser una poderosa herramienta para el análisis, diseño y, en general, el estudio de Redes de Interconexón. Dado que hay una gran variedad de gráficas que sirven para representar una red de interconexión, sólo se usará al Hipercubo para ilustrar el impacto de la Teoría de Gráficas en las redes de interconexión y viceversa.

Abstract The purpose of this document is to give a general overview of how Graph Theory results be a powerful tool for analysis, design and, in general, the study of Interconnection Networks. Already that there is a great variety of graphs that serve to represent an interconnection network, only the Hypercube will be used to illustrate the impact of Graph Theory on interconnection networks and vice versa.

Palabras Clave
Computación en Paralelo, Teoría de Gráficas, Redes de Interconexión, Hipercubo

[1] *Departamento de Matemáticas, Universidad Nacional Autónoma de México, México, luz.gasca@gmail.com, luzg@ciencias.unam.mx*
***Autor para Correspondencia**

Introducción

En ciencias de la computación, los modelos de cómputo sirven para describir entidades reales, llamadas computadoras y, además, se usan como herramientas para pensar en el problema y expresar algoritmos.

Los primeros modelos de computación fueron la máquina de Turing y las gramáticas formales. Modelos más recientes son: máquina de acceso aleatorio (RAM), maquina paralela de acceso aleatorio (PRAM), redes de interconexión, entre otros.

En una red de interconexión, cada procesador cuenta con su propia unidad de memoria y se conecta con otros procesadores mediante enlaces directos entre ellos; dos procesadores conectados, por un enlace, pueden intercambiar datos de forma simultánea.

La estructura matemática para modelar una red de interconexión es una gráfica no dirigida G, donde cada procesador P_i es un vértice de G y si hay un enlace entre dos procesadores, P_i y P_j, en la red entonces existe la arista (P_i, P_j).

La Teoría de Gráficas resulta ser una poderosa herramienta para el análisis, diseño y, en general, el estudio de Redes de Interconexón. Diferentes tipos de redes de interconexión resultan ser gráficas hamiltonianas, con diversos árboles generadores ajenos por aristas e, incluso, resultan ser gráficas de Cayley. En términos generales, se procura usar todos los atributos, propiedades y bondades de las gráficas para manipular y resolver problemas asociados con las redes de interconexión.

Una de las principales tareas de una red de interconexión es transportar (emitir y recibir) datos entre los procesadores, de manera eficiente. Esto se logra no sólo encontrando la ruta más corta, sino también econtrando árboles generadores. Cuando la gráfica asociada a la red de interconexión es una Gráfica de Cayley, moverse sobre la gráfica subyacente significa manipular los operadores del grupo asociado a la gráfica de Cayley.

Dado que hay una gran variedad de gráficas que sirven para representar una red de interconexión, sólo usaré el Hipercubo para ilustrar el impacto de la Teoría de Gráficas en las redes de interconexión y viceversa.

1. Computación Paralela

Existen diversos tipos de computadoras en paralelo, varían de 2 a 2^{16} procesadores, o más, los cuales pueden estar organizados de diversas maneras, de hecho, no existe un modelo genérico en la arquitectura en paralelo que pueda adaptarse a todas las computadoras en paralelo.

Diseñar algoritmos en paralelo, analizarlos y probar que son correctos es mucho más difícil que hacerlo para algoritmos secuenciales, por lo que analizar, diseñar y estudiar algoritmos paralelos es todo un reto.

Los criterios más importantes para evaluar un algoritmo paralelo son: tiempo de ejecución, número de procesadores, número total de pasos realizados, probabilidad de éxito y otras técnicas usadas para medir tales criterios. Estos criterios y las técnicas que los involucran, así como los resultados de tales evaluaciones constituyen el análisis del algoritmo.

El tiempo de ejecución o desempeño computacional de un algoritmo paralelo se define como el tiempo (cantidad de operaciones elementales) que le toma a un algoritmo resolver un problema en una máquina paralela; depende tanto del tamaño del ejemplar (datos de entrada), $n = |E|$, como del número de procesadores, p: $T(n, p)$.

Los modelos de cómputo en paralelo difieren en la forma como los procesadores se comunican y sincronizan; los dos principales modelos son Memoria Compartida y Redes de Interconexión.

En el paradigma de memoria compartida los procesadores se conectan a la memoria simple y compartida; los procesadores usan esta memoria para comunicarse. Cualquier par de procesadores que deseen intercambiar datos pueden hacerlo vía la memoria compartida. Ésta la manera más facil de modelar la comunicación. Además cada procesador, tiene una unidad de entrada y una unidad de salida para comunicarse con el mundo exterior. Todo procesador tiene acceso a cualquier variable.

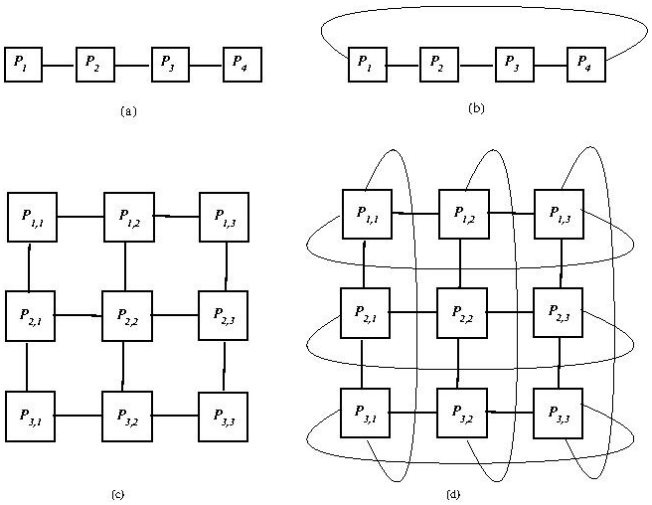

Figura 1. Topologías básicas.

Una red de Iinterconexión se modela con una gráfica, donde cada vértice representa un procesador y existe arista si dos procesadores están conectados, física o lógicamente, cada arista (i, j) representa un enlace de comunicación de dos vías entre los procesadores i y j; es decir, dos procesadores conectados por un enlace pueden intercambiar datos de forma directa y simultánea. Los procesadores se comunican por mensajes, vía los enlaces. Cada procesador tiene su propia memoria local y no hay memoria compartida disponible; cada procesador tiene acceso sólo a sus variables. Existen varias formas de organizar los procesadores llamadas topologías o arquitecturas. La Figura 1 muestra cuatro arquitecturas básicas: (a) arreglo lineal; (b) anillo; (c) malla y (d) toro.

2. El Hipercubo

En esta sección se presenta la red de interconexión Hipercubo, así como algunas propiedades topólogicas de la misma y, además, se muestran e ilustran algunos algoritmos que sacan provecho de la arquitectura.

Un hipercubo Q_d se define como: $Q_d = K_2$, si $d = 1$ y $Q_d = Q_{d-1} \times K_2$, si $d > 1$. El hipercubo consiste de un conjunto de $n = 2^d$ procesadores interconectados en un cubo booleano $d-$dimensional, llamado $d-$cubo. Etiquetamos los nodos de un hipercubo con etiquetas booleanas (de ceros y unos) de longitud d. Dos procesadores están conectados si y sólo si sus etiquetas difieren en solamente un bit. La Figura 2 presenta tres hipercubos: Q_1, Q_2, Q_3, con una etiquetación para sus nodos.

El hipercubo tiene una estructura recursiva. Podemos extender un cubo $d-$dimensional a uno $(d+1)-$dimensional conectando los correspondientes procesadores de dos $d-$cubos: un sub-cubo tendrá el bit más significativo igual a 0 y el otro tendrá el bit más significativo igual a 1.

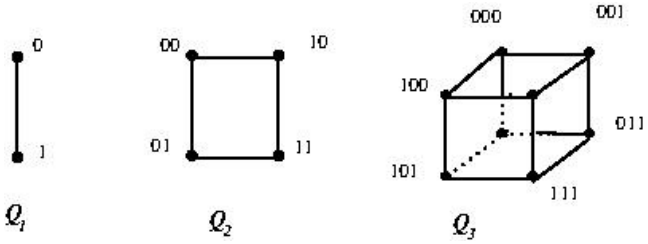

Figura 2. Hipercubos Q_1, Q_2, Q_3

Para un $d-$cubo, la distancia entre cualesquiera dos procesadores P_i y P_j es igual al número de bits en los que difieren; esto significa que podemos ir del procesador P_i al P_j intercambiando a lo más d bits, uno a la vez.

El hipercubo provee una excelente conexión, ya que existen diferentes rutas entre dos procesadores; es decir, podemos cambiar los bits apropiadamente en cualquier orden.

2.1 Propiedades del Hipercubo

El hipercubo es popular por ser una gráfica regular, tener un diámetro pequeño, contar con diversas e importantes propiedades como gráfica. Además por ser muy manipulable; es decir, muchos cálculos pueden realizarse de manera rápida y simple. Algunas propiedades del hipercubo fueron dadas por Saad y Schultz [13] y se resumen en el siguiente resultado.

Teorema 1. Todo $d-$cubo, Q_d, donde $d = \log n$; $n = 2^d$, tiene las siguientes propiedades:
(a) No tiene ciclos de longitud impar; (b) su diámetro es d;
(c) Es una gráfica $d-$conexa; (d) Es una gráfica simétrica por vértices; (e) Hay $(d!) \cdot 2^d$ diferentes maneras de enumerar sus vértices; (f) La distancia entre cualesquiera dos nodos (procesadores) i y j es igual al número de bits en los que difieren; así, la distancia es al menos d.

Recordemos que el diámetro es la mayor de las distancias, de las rutas más cortas entre dos nodos y determina el número de saltos que un mensaje podría realizar. Una gráfica G es simétrica por vértices si dados dos vértices u, v, existe un automorfismo ϕ entre los vértices tal que $\phi(u) = v$.

Se tiene la siguiente caracterización del hipercubo,[13]:

Teorema 2. Una gráfica $G = (V, A)$ es un $d-$cubo si y sólo si las siguientes cuatro condiciones se satisfacen:
(1) V tiene 2^n vértices; (2) Cada vértice tiene grado n; (3) G es conexa; (4) Cualquier par de nodos adyacentes A y B son tales que los nodos adyacentes a A y los adyacentes a B están relacionados en una correspondencia uno a uno.

Por ejemplo, el toro T_4 es una gráfica conexa con 2^4 nodos, cada vertice tiene grado 4 y no es difícil ver que cumple el punto (4) del Teorema 2, por lo tanto T_4 es Q_4. La Figura 3 muestra al toro T_4 y al hipercubo Q_4.

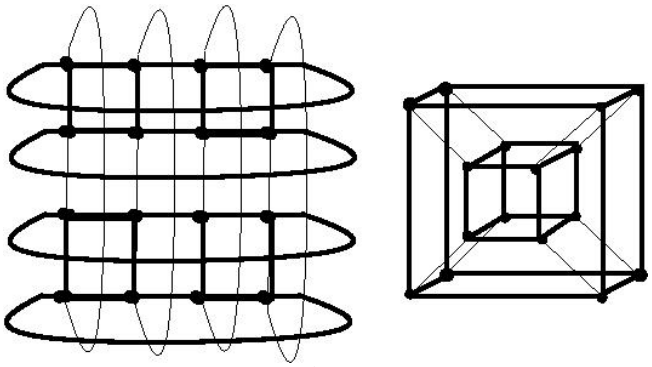

Figura 3. El toro T_4 es Q_4

La capacidad que tiene una red para funcionar a pesar de que alguno de sus componentes falle es conocida como tolerancia a fallas o tolerancia a errores; el cual es un concepto importante en las redes de interconexión.

La tolerancia a fallas de un hipercubo Q_d es $d-1$, dado que Q_d es una gráfica $d-$conexa, se puede observar que hay una estrecha relación entre la conexidad y la tolerancia a fallas.

2.2 Operaciones en el Hipercubo

A continuación diseñamos algoritmos sobre el Hipercubo síncrono para cuatro problemas simples: Ruteo, Suma de n elementos, Difusión de la Información y Multiplicación de Matrices.

2.2.1 Suma en un Hipercubo.

Sea A un arreglo de tamaño $n = 2^d$. Cada entrada $A(i)$ se almacena en la memoria local de cada procesador P_i, i, $0 \leq i < n$. Determinar la suma S de los n elementos del arreglo A, sobre el Hipercubo Q_d, y almacenarla en P_0. El Hipercubo Q_d es síncrono con n procesadores: $P_0, P_1, ..., P_{n-1}$.

Diseño del Algoritmo. El algoritmo directo consiste de d iteraciones. En la primera iteración calcula la suma de las parejas de elementos de procesadores, cuyos índices difieren en el bit más significativo; almacena esas sumas en el sub-cubo $Q_{(d-1)}$, cuyos bits más significativos sean 0. En las siguientes iteraciones procede de manera similar, sobre el subcubo de una dimensión menor, [9].

Consideremos ahora un ejemplo específico con ocho elementos. Sea $A = [6,1,5,4,3,7,2,8]$. Determinar la suma de los elementos en A.

Solución. Como $n = 8$, la suma será sobre el hipercubo Q_3. Los datos son: $A(0) = 6, A(1) = 1, A(2) = 5, A(3) = 4, A(4) = 3$, $A(5) = 7, A(6) = 2, A(7) = 8$. La Figura 4 muestra cómo van efectuándose las sumas sobre el Hipercubo Q_3. Después de la primera iteración P_0 contiene $A(0) = 6$; P_1 tiene $A(1) = 8$; P_2 contiene $A(2) = 7$; P_3 contiene $A(3) = 12$. Después de la segunda iteración, $P_0 : A(0) = 16$; $P_1 : A(1) = 20$ y, finalmente, en la tercera iteración, $P_0 : A(0) = 36$.

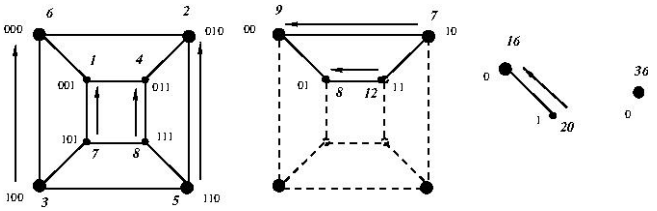

Figura 4. Ejemplo de la suma en el hipercubo Q_3.

2.2.2 Enrutamiento (*routing*) en un hipercubo

Cada procesador P_i, sobre el hipercubo, requiere mandar un mensaje al procesador $P_{\sigma(i)}$, el cual mantiene la variable $x_{\sigma(i)}$. Todos los mensajes son enviados al mismo tiempo. El objetivo es dirigirlos (enrutarlos) todos, a la vez, por el hipercubo rápidamente. Además, cada arista del hipercubo puede trasmitir sólo un mensaje a vez.

El problema consiste en encontrar las rutas más cortas, entre fuentes y destinos, minimizando los conflictos que podrían surgir al tratar de usar la misma ruta al mismo tiempo. Además, si dos mensajes tratan de usar la misma ruta al mismo tiempo uno de ellos deberá esperar en un buffer; así que también, hay que optimizar los requerimientos de espacio para el buffer.

Diseño del Algoritmo. La mejor ruta depende de una permutación particular de la etiqueta de los nodos. Sin embargo, generalmente no es posible analizar la permutación para encontrar la mejor ruta, ya que está distribuida entre los procesadores. Por lo tanto, buscamos un esquema que funcione bien, en promedio.

La clave de la estrategia es usar aleatoriedad. El enrutamiento consta de dos fases. En la primera fase, cada procesador P_i envía un mensaje a un procesador elegido aleatoriamente, bajo una distribución uniforme, de entre todos los procesadores e independientemente del destino, digamos que tal procesador destino es $P_{\sigma(i)}$. En la segunda fase, el mensaje se envía a lo largo de la ruta más corta entre el procesador que recibió el mensaje, $P_{\sigma(i)}$, y el destino final.

Todos los mensajes son enviados de la misma manera, así podemos concentrarnos únicamente en un mensaje, es decir, del nodo i al j. Consideremos la representación binaria de $i : b_1 b_2 \cdots b_d$ con posible inicio de ceros, y la de $j : c_1 c_2 \cdots c_d$.

En la primera fase, elegimos aleatoriamente un procesador a, considerando la representación binaria de i, bit por bit, y decidiendo aleatoriamente, con probabilidad 1/2, si la ruta se dirige al siguiente vecino o no.

Si decidimos no enviar el mensaje, inmediatamente hacemos la siguiente elección, sin esperar la siguiente ronda. Suponemos que el cálculo local es mucho más rápido que el paso del mensaje. Cuando hacemos la selección concerniente al último bit la ruta aleatoria queda construida. Cada elección se realiza localmente.

Como todos los procesadores envían mensajes al mismo tiempo, puede haber más de un mensaje esperando a ser enviado a través de la misma ruta, generándose conflictos. En

este caso, los mensajes son almacenados en una cola (buffer), en forma aleatoria si hay más de uno, y son enviados cuando la arista queda disponible. Cada número k, en un rango apropiado, tiene la misma probabilidad de ser elegido como un destino aleatorio.

El enrutamiento del procesador a, al destino j se realiza de manera determinista. Supongamos que los procesadores a y j difieren en t bits, con índices tales que $k_1 < k_2 < \cdots < k_t$. El mensaje es enviado al cambiar k_1, luego k_2 y así de manera sucesiva.

Este esquema de enrutamiento es simple de implementar. Las rutas no necesariamente son las más cortas. La longitud de cada ruta, sin embargo, es a lo más $2d$. La principal propiedad de estas rutas es que, con muy alta probabilidad, generan pocos conflictos. Así, se espera que la permutación se termine en $O(d)$ pasos.

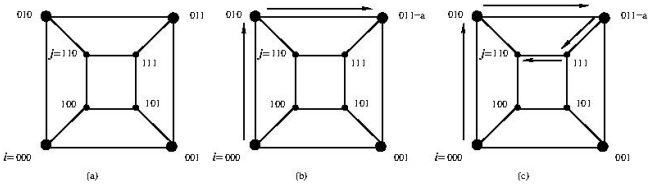

Figura 5. Enrutamiento en un hipercubo

Ilustramos con un ejemplo en Q_3. Supongamos que el procesador origen es $i = 000$ y el destino es $j = 110$. Con probablidad 1/2 preguntamos si enviamos el mensaje a 100, supongamos que la respuesta es no. Luego preguntamos si cambiamos el segundo bit, es decir, si se envía a 010, sea la respuesta sí. Finalmente, preguntamos si se envía hacia el 011 y sea sí la respuesta. De esta manera, el procesador aleatorio es el $a = 011$. La Figura 5(b), muestra la ruta parcial obtenida, sobre el hipercubo. Ahora tenemos: $a = 011$ y $j = 110$, la ruta queda: $a = 001 \rightarrow \mathbf{1}11 \rightarrow 1\mathbf{1}1 \rightarrow 11\mathbf{0} = j$. La Figura 5(c), presenta la ruta final.

2.2.3 Difusión de la Información, *Broadcasting*.

El objetivo es enviar un elemento X, almacenado en el registro $D(0)$ del procesador P_0 a todos los otros procesadores P_i de un hipercubo Q_p, donde $p = n^d$.

Un árbol generador proporciona la solución a este problema. Hay que tener rutas alternas, por si falla un enlace, esto se consigue construyendo árboles generadores ajenos por aristas. Si tales árboles son poco profundos y balanceados, la comunicación será más eficiente.

Se tiene que un hipercubo Q_{2d} posee d arboles generadores ajenos por aristas y con las aristas sobrantes se forma una trayectoria de longitud d; para generar esos d árboles se aprovecha la estructura de la red (hipercubo), [2].

Diseño de un Algoritmo. La estrategia es simple: Iniciamos desde la dimensión de menor orden hasta la de mayor orden, consecutivamente, en d iteraciones. Durante la primera iteración, P_0 manda una copia de X a P_1 usando el enlace entre ellos, (P_0, P_1); durante la segunda iteración, P_0 y P_1 envían copias de X a P_2 y P_3, respectivamente, utilizando los enlaces

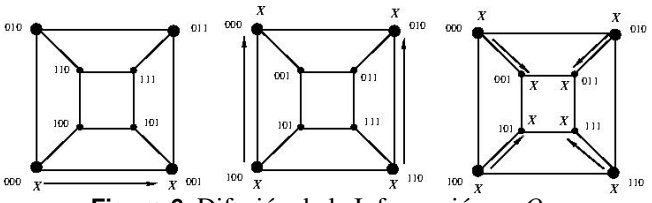

Figura 6. Difusión de la Información en Q_3.

(P_0, P_2) y (P_1, P_3); y así sucesivamente, [9].

En la Figura 6 se ilustra gráficamente cómo se va difundiendo la información, iteración por iteración sobre Q_3.

2.3 Encajes o Inmersiones

En esta sección, analizaremos el problema de mapear otras topologías (anillos y mallas) en el hipercubo. Dada una gráfica $G = (V, A)$ con $|V| \leq 2^d$, asignamos los vértices de la gráfica a los nodos del hipercubo de modo que todo vértice adyacente en la gráfica corresponda a nodos adyacentes en el $d-$cubo. Existen principalmente dos razones por las cuales tales asignaciones son importantes: (1) Algunos algoritmos pueden ser desarrollados para otra arquitectura para la cual funciona perfectamente. Entonces se podría aplicar el mismo algoritmo con un poco de esfuerzo de programación. Si la arquitectura original se puede mapear en el hipercubo, éste será fácil de lograr. (2) Un problema dado puede tener una estructura bien definida que conduce a un patrón particular de comunicación. El mapeo de la estructura puede resultar en un ahorro sustancial en el tiempo de comunicación. Si la malla está perfectamente mapeada en el hipercubo, entonces sólo se requiere comunicación local entre los nodos del hipercubo lo que resulta en ahorros importantes en tiempos de transferencia.

2.3.1 Inmersión sobre anillos y arreglos lineales

Dado un anillo con 2^n vértices, consideremos el problema de asignar sus vértices en los nodos de un hipercubo de tal manera que se preserven las adyacencias entre los nodos. Otra forma de ver este problema es buscando un ciclo de longitud $N = 2^n$ que pase por cada nodo una y sólo una vez, es decir, buscamos un ciclo hamiltoniano en un hipercubo. Si numeramos los nodos de un hipercubo de modo que dos nodos vecinos difieren en uno y sólo un bit, un circuito hamiltoniano simplemente representa una secuencia de números binarios de n bits tales que cualesquiera dos números sucesivos tienen solo un bit diferente de modo que todos los números binarios de n bits serán representados. Las sucesiones binarias con estas propiedades son llamadas Códigos Gray. Hay distintas formas en las cuales un código Gray puede ser generado pero el método más conocido es el Código Gray Reflejado, se construye de la siguiente manera: Comenzamos con la sucesión de los dos números de un bit, 0 y 1; éste es un código Gray 1-bit. Para construir un código Gray 2-bit tomamos la misma sucesión e insertamos un 0 en frente de cada número, después tomamos la sucesión en orden inverso e insertamos un 1 en frente de cada número. Así, ob-

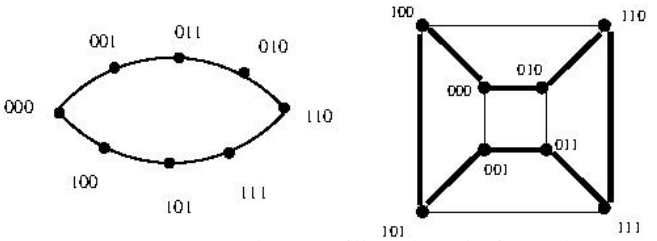

Figura 7. Mapeo de un Anillo sobre el Hipercubo

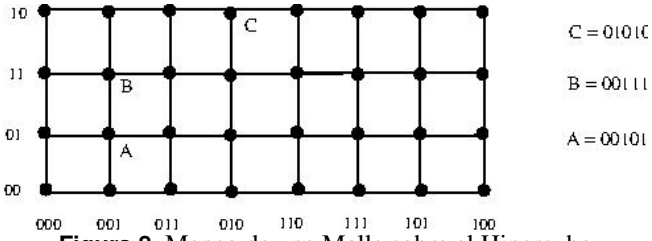

Figura 8. Mapeo de una Malla sobre el Hipercubo

tenemos las sucesiones: $G_1 = \{0,1\}$, $G_2 = \{0\mathbf{0}, 0\mathbf{1}, 1\mathbf{1}, 1\mathbf{0}\}$ y $G_3 = \{0\mathbf{00}, 0\mathbf{01}, 0\mathbf{11}, 0\mathbf{10}, 1\mathbf{10}, 1\mathbf{11}, 1\mathbf{01}, 1\mathbf{00}\}$

De manera general, sea G_i^R la sucesión obtenida a partir de G_i invirtiendo el orden de las entradas y por $0G_i$ a la sucesión obtenida al poner un 0 a cada elemento de la sucesión. Análogamente para $1G_i$. Así, el código Gray puede ser generado de manera recursiva por: $G_{n+1} = \{0G_n, 1G_n^R\}$.

Los códigos Gray nos permiten mapear anillos, cuyas longitudes son potencias de 2, en hipercubos. Supongamos ahora que tenemos un anillo de longitud ℓ que queremos mapear en un hipercubo. Primero, observemos que el mapeo es posible sólo cuando ℓ es par ya que de acuerdo con el Teorema 1, un hipercubo no admite ciclos de longitud impar. Por lo tanto, suponemos que $4 \leq \ell \leq 2^n$. El problema es encontrar un ciclo de longitud ℓ en el $n-$cubo, donde ℓ es par.

Sea $m = (\ell - 2)/2$ y denotamos por $G_{n-1}(m)$ el Código Gray $(n-1)$ bits que consta de los primeros m elementos de G_{n-1}. Entonces, un ciclo que tiene la propiedad deseada es el siguiente: $\{0G_{n-1}(m), 1G_{n-1}(m)^R\}$.

Cuando $\ell = 2^n$, se obtiene un caso particular de la fórmula $G_{n+1} = \{0G_n, 1G_n^R\}$. Así, podemos afirmar lo siguiente.

Proposición 1. Un anillo de longitud ℓ puede ser mapeado en el $n-$cubo cuando ℓ es par y $4 \leq \ell \leq 2^n$, [13].

Por ejemplo, en Figura 7 el anillo de dimensión 8, generado por el código Gray, es incrustado en el $3-$cubo y cumple que $4 \leq 8 \leq 2^3$.

Es fácil realizar la inmersión de un arreglo lineal, en lugar de un anillo, en el $n-$cubo. Es suficiente mapear los nodos del arreglo lineal $P_0, P_1, ..., P_l$ de longitud arbitraria $l \leq 2^{n-1}$, sucesivamente, en los nodos $g_0, g_1, ..., g_l$ del $n-$cubo. Dado un arreglo lineal de longitud arbitraria ℓ, el $n-$cubo de dimensión más pequeña en el cual puede ser mapeado es el cubo de dimensión $n = \lceil \log_2(\ell + 1) \rceil$.

2.3.2 Inmersión de Mallas en el Hipercubo

La inmersión de mallas, de cualquier dimensión, en el hipercubo es una de las propiedades más atractivas de la topología del $n-$cubo y es una de las razones principales por la cual la arquitectura del hipercubo esxitosa. También usa Códigos Gray. Sea $M = m_1 \times m_2 \times \cdots \times m_d$ una malla en el espacio $d-$dimensional R^d, además presuponemos que el tamaño de la malla en cada dirección es una potencia de 2; es decir, para cada $i, m_i = 2^{p_i}$. Sea $n = p_1 + p_2 + \cdots + p_d$ y consideremos el problema de mapear los vértices de la malla en el $n-$cubo, de tal manera que se asigne un vértice de la malla a un nodo del $n-$cubo. Observemos que tenemos los nodos suficientes para acomodar un vértice de la malla por nodo en el $n-$cubo.

Entenderemos por mapeo de la malla en el cubo a la asignación de vértices de la malla en los nodos del $n-$cubo, de tal manera que la propiedad de preservar adyacencias se conserve, es decir, tal que dos vértices vecinos en la malla son asignados a nodos vecinos en el $n-$cubo.

A continuación se ejemplifica el proceso de inmersión. Consideremos una malla bi-dimensional y de 8×4; es decir, tenemos $d = 2$, $p_1 = 3, p_2 = 2$, $n = p_1 + p_2 = 5$. Un número binario A de cualquier nodo del 5-cubo lo podemos separar en dos partes: sus primeros 3 bits y sus últimos 2 bits, lo cual escribimos de la siguiente forma: $A = b_1, b_2, b_3, c_1, c_2$ donde b_i y c_j son bits 0 ó 1. Se desprende de la definición del $n-$cubo que cuando los dos últimos bits son fijos, los 2^{p_1} nodos resultantes forman un p_1-cubo, con $p_1 = 3$. Del mismo modo, cada vez que fijamos los primeros tres bits obtenemos un p_2-cubo. Entonces el mapeo es claro. La elección de un código Gray de 3 bits para la dirección X y un código Gray de 2 bits para la dirección Y, el vértice (x_i, y_j) de la malla es asignado al nodo b_1, b_2, b_3, c_1, c_2 donde b_1, b_2, b_3 es el código Gray de 3 bits para x_i, mientras $c_1 c_2$ es el código Gray de 2 bits para y_j.

El mapeo se ilustra en la Figura 8 donde se obtiene el número de nodo binario de cualquier vértice de la malla mediante la concatenación de sus coordenadas binaria x y y. Por ejemplo, el nodo $A = 00101$ del $5-$cubo, se mapea al vértice $(001, 01)$ de la malla. También se ilustran los nodos $B = 00111$ y $C = 01010$

Así, si llamamos sucesión Gray a cualquier subsucesión de un código Gray, observamos que cualquier columna de la malla forman una sucesión Gray y cualquier renglón de puntos de la malla forma una sucesión Gray. Entonces nos referimos a los código antes definidos como 2D-código Gray. Se tiene el siguiente resultado.

Teorema 3. Cualquier malla $M = m_1 \times m_2 \times \cdots \times m_d$ en el espacio $d-$dimensional R^d, donde $m_i = 2^{p_i}$ puede ser mapeada en un $n-$cubo donde $n = p_1 + p_2 + \cdots + p_d$. La numeración de los vértices en la malla es cualquier numeración de tal manera que su restricción a cada $i-$ésima variable es una sucesión Gray, [13].

3. El Hipercubo Aumentado

El Hipercubo aumentado es una variante del Hipercubo, a Q_n se le agregan aristas complemento; Esto es, los vertices con etiquetas x y $\bar{x}$ estarán unidos por una arista. Por ejemplo,

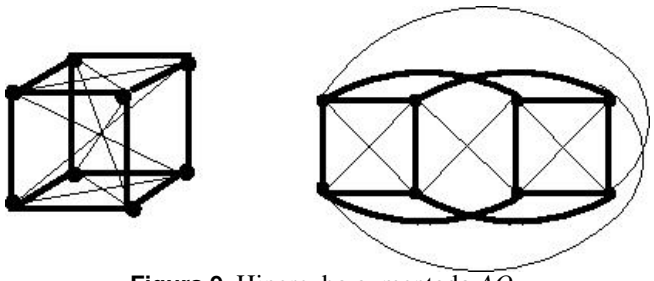

Figura 9. Hipercubo aumentado AQ_3

en Q_3 están las aristas $(000,111)$ y $(001,110)$.

Sea $n \geq 1$. El cubo aumentado AQ_n de dimensión n tiene 2^n vértices, cada vértice es una $n-$ada de elementos en $\{0,1\}$. Definimos $AQ_1 = K_2$. Para $n \geq 2$,AQ_n se obtiene tomando dos copias del cubo aumentado AQ_{n-1}, denotadas por AQ^0_{n-1} y AQ^1_{n-1}, y agregando $2 \times 2^{n-1}$ aristas entre las dos copias quedando: $V(AQ^0_{n-1}) = \{0,a_2,a_3,...,a_n : a_i = 0 \ o \ a_i = 1\}$ y $V(AQ^1_{n-1}) = \{1,b_2,b_3,...,b_n : b_i = 0 \ o \ b_i = 1\}$. Un vértice $A = 0,a_2,a_3,...,a_n$, de AQ^0_{n-1}, está unido a un vértice $B = 1,b_2,b_3,...,b_n$, de AQ^1_{n-1}, si y sólo si para toda $i, 2 \leq i \leq n$, se cumple (1) $a_i = b_i$, en este caso, AB es una arista del hipercubo, ó (2) $a_i = \overline{b_i}$, aquí, AB es una arista complemento. La construcción recursiva de AQ_n se representa como: $AQ_n = AQ^0_{n-1} \diamond AQ^1_{n-1}$, [6].

La Figura 9 presenta dos ilustraciones de AQ_3.

3.1 Propiedades del Hipercubo Aumentado

El hipercubo aumentado AQ_n es una gráfica simétrica por vértices, es $(2n-1)-$regular, es $(2n-1)-$conexa y, además, es una gráfica de Cayley.

Algunas propiedades, dadas por Choudum y Sunitha [6], del hipercubo aumentado AQ_n se resumen en los siguientes resultados

Teorema 4. Sea AQ_n un hipercubo aumentado.
(1) El diámetro de AQ_n es $\lceil n/2 \rceil$, $\forall n \geq 1$; (2) Entre cualesquiera dos vértices $x,y \in V(AQ_n)$ existen $2n-1$ trayectorias internamente disjuntas de x a y, para $n \geq 4$. (3) Se tiene que $\kappa(AQ_1) = 1$, $\kappa(AQ_2) = 3$, $\kappa(AQ_3) = 4$ y $\kappa(AQ_n) = 2n-1$, para $n \geq 4$.

Teorema 5. Sea AQ_n un hipercubo aumentado.
(1) Para toda $n \geq 2$, el hipercubo aumentado AQ_n contiene un $2k-$ciclo para cada $k, 2 \leq k \geq 2^{n-1}$; (2) Para $n \geq 3$, AQ_n contiene dos árboles binarios completos ajenos por aristas sobre $2^n - 1$ vértices y cuya raíz está en O^n. (3) Para $n \geq 3$, AQ_n contiene $n-1$ árboles generadores ajenos por aristas $(S_1,...,S_{n-1})$ y con las aristas restantes se forma un árbol R_n.

Se ilustra el Teorema 5, con Figura 10 el inciso (2) y con la Figura 11 el (3).

3.2 Algoritmos para el Hipercubo Aumentado

Un algoritmo de enrutamiento para AQ_n utiliza la ruta más corta entre cualquier para nodos. En particular, usa el siguiente resultado, [6].

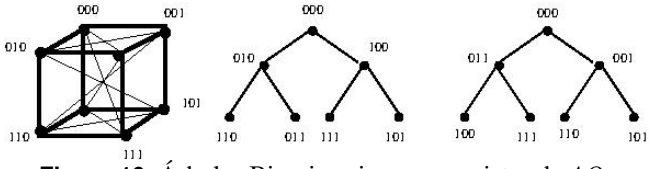

Figura 10. Árboles Binarios ajenos por aristas de AQ_3

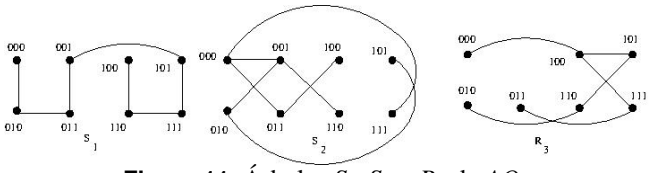

Figura 11. Árboles S_1, S_2 y R_n de AQ_3

Teorema 6. Sea $x,y \in V(AQ_n) = V(AQ^0_{n-1}) \diamond V(AQ^1_{n-1})$. (1) Si $x,y \in V(AQ^0_{n-1})$, existe una ruta más corta de x a y en AQ_n tal que todos sus vértices están en AQ^0_{n-1} y análogamente para AQ^1_{n-1}. (2) Sean $x \in V(AQ^0_{n-1})$ y $y \in V(AQ^1_{n-1})$, entonces (2i) existe una $(x,y)-$ruta más corta en AQ_n con todos sus vértices en AQ^1_{n-1}, excepto x. (2ii) existe una $(x,y)-$ruta más corta en AQ_n con todos sus vértices en AQ^0_{n-1}, excepto y.

3.2.1 Algoritmo de Enrutamiento para AQ_n.

Se usa el teorema anterior para enviar un mensaje desde un vértice origen S a un destino $D = (d_1,d_2,...,d_n)$, a través de la $(S,D)-$ruta más corta. Mientras el mensaje se envía a lo largo de la trayectoria, un vértice activo $C = (c_1,...,c_n)$ realiza las tres tareas siguientes: (1) Calcula $[C \oplus_2 D] = (c_1 \oplus_2 d_1, c_2 \oplus_2 d_2, ..., c_n \oplus_2 d_n)$. (2) Busca en $[C \oplus_2 D]$ el menor índice tal que $c_i \oplus_2 d_i = 1$. (3) Si $(c_{i+1} \oplus_2 d_{i+1} = 0)$ entonces cambia la $i-$ésima entrada de C, c_i por d_i; envía el mensaje al siguiente nodo activo $C' = (d_1,...,d_i,c_{i+1},...,c_n)$. Si $(c_{i+1} \oplus_2 d_{i+1} = 1)$ entonces cambia la $i-$ésima entrada de C, c_i por d_i; envía el mensaje al siguiente nodo activo $C' = (d_1,...,d_i,\overline{c_{i+1},...,c_n})$.

Por ejemplo, un enrutamiento de 000000 a 101011 en AQ_6, sería $000000 \to 100000 \to 101000 \to 101011$

3.2.2 Difusión de la Información para AQ_n.

El algoritmo de difusión de la información para cubos aumentados es similar al algoritmo para hipercubos. Como AQ_n es una gráfica simétrica por vértices, basta describir el algoritmo con origen en el $0^n = 00\cdots0$, (n veces).

El algoritmo inicia enviando el mensaje desde 0^n a todas las aristas incidentes a él. Se etiquetan todos los nodos y aristas cuyo vértice inicial es 0^n. Después, mientras exista un vértice $x = O^k A$, $2 \leq k < n$, tal que $x_{k+1} = 1$ e ingrado 1, en paralelo, envía el mensaje desde x a través de las aristas que no están etiquetadas incidentes a él, [6].

La Figura 12 ilustra el algoritmo. Este algoritmo toma $\lceil n/2 \rceil$ pasos para trasmitir el mensaje a todos los vértices al mismo tiempo. En comparación con el hipercubo el número de vértices que reciben el mensaje a la vez es mayor.

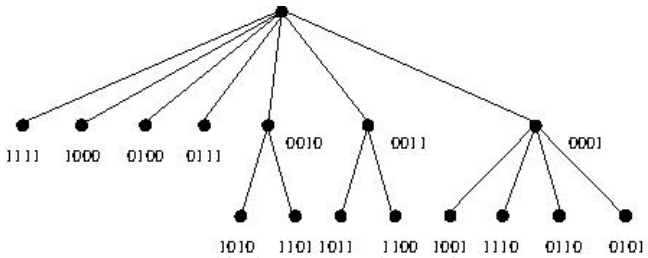

Figura 12. Difusión de la Información en AQ_4.

4. Conclusiones

Durante el desarrollo de este documento revisamos propiedades topológicas y algorítmicas del Hipercubo y algunas de sus variantes del Hipercubo.

En redes de interconexión, la simulación de una arquitectura por otra es importante. El problema de la simulación de una red por otra se modela como un problema de inmersión en gráficas. Las redes de ciclos o trayectorias son adecuadas para el diseño de algoritmos sencillos con bajos costos de comunicación. Estos han motivado a un gran número de investigación en los caminos o ciclos que son inmersibles en otras redes de interconexión.

Se ha observado que el Hipercubo y sus variantes tienen estructuras recursivas, y así todas las pruebas proceden de resultados conocidos y aplicando inducción sobre el orden de la gráfica. En el proceso de las demostraciones, para probar la base de inducción en gráficas cuyo orden es pequeño, regularmente se utiliza la observación o verificación directa; en la hipótesis de inducción es la construcción de un camino o de un ciclo requerido por algunas propiedades estructurales de las gráficas. Así, algunos investigadores, consideran que la clave de estudiar problemas de inmersión de ciclos y trayectorias tanto en el Hipercubo como en sus variantes y, en general, en otras gráficas.

Algunas aportaciones importantes de la Teoría de Gráficas a las Redes de interconexión son: (1) El diseño de algoritmos de comunicacion óptimos: Si sabemos que el diámetro de la gráfica es pequeño, podemos tomar ventaja de ello; de igual forma si sabemos que se tienen estructuras ajenas por aristas, como árboles generadores, ciclos hamiltonianos o trayectorias. (2) Alta Confiabilidad: las propiedades de la gráfica garantizan el comportamiento de la red. (3) Rica estructura: Las propiedades de inmersión garantizan que se puede modificar levemente un algoritmo al cambiar de arquitectura. La descomposición de la gráfica en árboles, trayectorias, ciclos y otras subgráficas facilitan no sólo el diseño de algoritmos sino también la demostración y verificación de propiedades.

Referencias

[1] B. W. Arden and K. W. Tang. Representation and routing of cayley graphs. Technical report, Dep. of Electrical Engineering, Univerity of Rochester, 1989. EL-89-02.

[2] B. Barden, R. Libeskind-Hadas, J. Davis, and W. Williams. On the edge-disjoint spanning trees in hypercubes. *Information Processing Letters*, 70:13–16, 1999.

[3] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. American Elsevier, New York, 1976.

[4] G. Chartrand. *Introductory Graph Theory*. Dover Publication, 1977.

[5] O.R Chartrand, G. & Oellermann. *Applied an Algorithmic Graph Theory*. International Series in Pure and Applied Mathematics. McGraw-Hill, 1993.

[6] S.A Choudum and V. Sunitha. Augmented cubes. *Networks*, 40:71–84, 2002.

[7] G. De Marco and U. Vaccaro. Broadcasting in hypercubes and star graphs with dynamic faults. *Information Processing Letters*, 66:321–326, 1998.

[8] L.H. Hsu and Ch. K. Lin. *Graph Theory and Interconnection Networks*. Taylor and Francis Group, 2009.

[9] J. JaJa. *Introduction to parallel algorithms*. Addison Wesley, 1992.

[10] S. Lakshmivarahan and S.K. Dhall. Ring, torus and hypercube architectures/ algorithms for parallel computing. *Parallel Computing*, 25:1877–1906, 1999.

[11] S. Latifi and A. El-Amaway. On the folded hypercubes. *I Proc. ICCP*, pages 180–187, 1989.

[12] U. Manber. *Introduction to Algorithms. A creative Approach*. Addison Wesley, 1999.

[13] Y. Saad and M.H. Schultz. Topological properties of hypercubes. *IEEE Trans. on Computer*, 37:867–872, 1988.

[14] S. Wagner and M. Wild. An decomposing the hypercube q_n into n isomorphic edge-disjoint trees. *Discrete Mathematics*, 312:1819–1822, 2012.

[15] J. Werapun, S. Itakosum, and V. Boonjing. An efficient parallel construction of optimal independent spanning trees on hypercubes. *J. Parallel Distrib. Comput.*, 72:1713–1724, 2012.

[16] J.M. Xu and M.J. Ma. Survey on path and cycles embedding in some networks. *Front. Math.*, 40:217–252, 2009.

[17] J.S Yang, J.M Chang, S.M Tang, and Y.L. Wang. Parallel construction of optimal independent spanning trees on hypercubes. *Parallel Computing*, 33:73–79, 2007.