Computación y Sistemas Vol. 4 No.4 pp. 289 - 297
© 2001, CIC - IPN. ISSN 1405-5546 Impreso en México

# CALMANT: Un Método Sistemático para la Ejecución de Algoritmos Hipercubo en Sistemas Multiprocesador

**Luis Díaz de Cerio, Miguel Valero García, Antonio González y Dolors Royo**
Departament d' Arquitectura de Computadors
Universitat Politècnica de Catalunya
C/Jordi Girona 1-3 Mòdul D6, 08034-Barcelona, Spain
Telf: +34 93 4017188, Fax: +34 93 4017055
e-mail: {ldiaz, miguel, antonio, dolors}@ac.upc.es

*Artículo recibido el 6 de marzo de 2000 ; aceptado el 21 de febrero de 2001*

## Resumen

*En este trabajo presentamos el método CALMANT (CC-cube Algorithms on Meshes And Tori) como un método sistemático para la ejecución de un cierto tipo de algoritmos, que denominamos algoritmos CC-cubo, sobre mallas y toros de varias dimensiones. Es muy frecuente encontrar algoritmos CC-cubo en la literatura (FFT, complete exchange, cálculo de valores y vectores propios, etc.) pero la aplicación directa de estos algoritmos no permite explotar eficientemente el ancho de banda que nos ofrecen las redes de interconexión en malla y toro. El método CALMANT permite reorganizar los cálculos y las comunicaciones de los algoritmos CC-cubo de manera que la eficiencia aumente notablemente. La importancia de este método no sólo radica en el aumento de la eficiencia sino que además puede ser aplicado de forma sistemática sobre diferentes tipos de arquitectura.*

**Palabras Clave:** Algoritmo CC-cubo, Segmentación de las Comunicaciones, Embedding, Hipercubo, Malla y Toro.

## 1 Introducción

Una de las metas más importantes a la hora de programar un multicomputador es el uso eficiente de la red de interconexión existente entre los nodos que componen el multicomputador. Dependiendo de la red de interconexión, la eficiencia en cuanto al tiempo de ejecución de un mismo algoritmo puede ser muy diferente de un multicomputador a otro.

Las características principales de una red de interconexión son, entre otras, su topología, el número de puertos de que disponen los nodos y el modelo de comunicación utilizado. Para poder hacer un uso eficiente de la red de interconexión, es necesario tener en cuenta estas tres características. Nuestro trabajo está enfocado concretamente a redes de interconexión con topología en hipercubo, malla o toro de varias dimensiones. En cualquiera de los casos estudiaremos arquitecturas con un puerto (one port) y con máximo número de puertos (all ports). En cuanto a los modelos de comunicación que consideraremos son: *store-and-forward, wormhole, circuit-switching* y *virtual cut-through*.

Frecuentemente, los algoritmos diseñados para un determinado modelo de multicomputador, no pueden ser ejecutados (o no son eficientes) en un modelo de multicomputador diferente. Este trabajo presenta el método CALMANT como un método sistemático para ejecutar un determinado tipo de algoritmos sobre los diferentes modelos de multicomputador mencionados anteriormente manteniendo en cualquiera de los casos una eficiencia alta. Por otro lado, aunque fuera del alcance de este trabajo, el método propuesto puede ser fácilmente extensible a otros tipos de arquitectura no tan comunes pero que pueden llegar a ser interesantes en un futuro. Éste es el caso, por ejemplo, de mallas y toros de cuatro o más dimensiones o de mallas y toros de seis u ocho vecinos.

El tipo de algoritmos a los cuales va enfocado el trabajo es el que denominamos como algoritmos de *Cálculo y Comu-*

*nicación con topología en Hipercubo (CC-cubo)*. Hay muchos algoritmos que pueden ser clasificados dentro de este tipo. Éste es el caso, por ejemplo, de la FFT, la transformada de Harley, complete exchange, cálculo de valores y vectores propios de una matriz, etc... De aquí se desprende la importancia que tiene un método sistemático como el que proponemos, para poder programar dichos algoritmos sobre multicomputadores con diferentes redes de interconexión manteniendo la eficiencia.

La eficiencia de un algoritmo no sólo depende de las características de la red de interconexión para la cual se ha diseñado el algoritmo, sino que también depende de la relación que existe entre los parámetros de la arquitectura (i.e., tiempo de cálculo, tiempo de transmisión) y los parámetros del problema (i.e., el tamaño del problema, número de operaciones por unidad de tamaño). El método CALMANT también tiene en cuenta esto y da como resultado algoritmos que se adaptan a la relación entre dichos parámetros. Para ello, el trabajo realizado tiene una componente importante de modelización del tiempo de ejecución de los algoritmos. Los modelos de tiempo de ejecución obtenidos nos permiten en cada caso tomar decisiones que optimicen el rendimiento de dichos algoritmos.

El objetivo de este trabajo es hacer una presentación del método CALMANT y dar algunos resultados. Concretamente en la sección 2 haremos una visión general del método. En la sección 3 se describen las arquitecturas y los algoritmos hacia los cuales está enfocado este trabajo. El concepto de segmentación de comunicaciones es introducido en la sección 4. Los embeddings que utilizaremos son descritos en la sección 5. En la sección 6 analizaremos cómo se lleva a cabo el encaminamiento de mensajes en los algoritmos resultantes y finalmente, en la sección 7, mostraremos algunas figuras de rendimiento.

## 2 Visión General

El método CALMANT está basado en la combinación de dos conceptos claramente diferenciados: la *segmentación de las comunicaciones* y el *embedding* de CC-cubos sobre hipercubos, mallas y toros.

Los algoritmos CC-cubo se componen de $2^d$ procesos, de manera que si representamos cada proceso por un punto y unimos mediante una línea cualquier par de procesos que se comuniquen entre sí (procesos vecinos), el resultado es un hipercubo. La ejecución de estos algoritmos consiste en un número determinado de iteraciones. En cada iteración los procesos realizan una fase de cálculo y otra de comunicación (no necesariamente en este orden). La característica principal de estos algoritmos es que en cada una de las iteraciones todos los procesos se comunican por una misma dimensión del hipercubo. De esta manera, suponiendo que existe un enlace entre cualquier par de procesos vecinos, dichos algoritmos solamente utilizarían un $1/d$ de la capacidad de comunicación total, donde $d$ es la dimensión del hipercubo. Esto es lo que sucedería, por ejemplo, si mapeamos un CC-cubo sobre un multicomputador con topología en hipercubo.

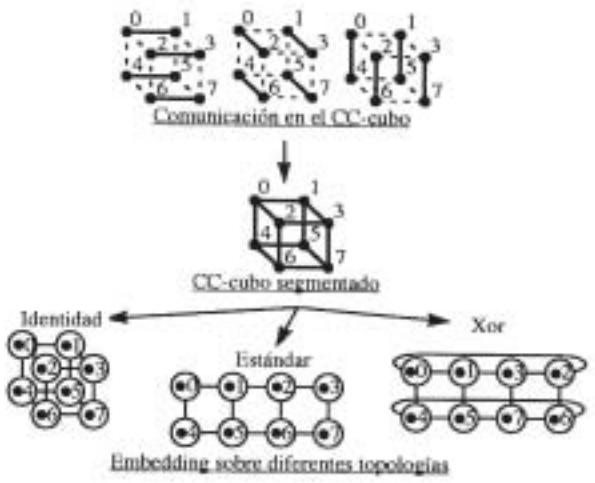


Figura 1: Elementos básicos del método CALMANT: segmentación de las comunicaciones y embedding de un algoritmo CC-cubo segmentado sobre diferentes topologías

Mediante la segmentación de las comunicaciones conseguiremos, hasta el punto que sea posible, utilizar todas las dimensiones del hipercubo simultáneamente. De esta manera seremos capaces de utilizar prácticamente al completo la capacidad de comunicación que pueda existir entre los procesos del CC-cubo.

Como muestra la figura 1, la idea es muy simple. Se trata de reorganizar el algoritmo para que, en vez de utilizar las dimensiones del hipercubo de forma secuencial, podamos utilizarlas en paralelo. En la figura se muestra mediante trazo más grueso la dimensión del hipercubo que es utilizada en cada instante de tiempo. El algoritmo resultante será denominado algoritmo *CC-cubo segmentado*.

No siempre es posible disponer de un enlace directo entre todas las parejas de procesos vecinos. Por ejemplo, en una malla o en un toro, cada nodo sólo tiene (como mucho) cuatro nodos vecinos. Así pues, si queremos ejecutar un algoritmo CC-cubo cuya dimensión sea $d > 4$ será imposible que todos los procesos vecinos se sitúen en nodos vecinos.

El *embedding* de CC-cubos sobre mallas y toros es una función que nos permitirá mapear los procesos de un CC-cubo segmentado sobre los nodos de un hipercubo, una malla o un toro. Los embeddings que proponemos realizan este mapeo de manera que, cualquier par de procesos vecinos según una misma dimensión del hipercubo, sean mapeados en nodos situados a la misma distancia en la arquitectura de destino. Además, dado un proceso localizado en un nodo determinado, sus procesos vecinos serán mapeados en nodos situados lo más cercanamente posible a dicho nodo, es de-

cir, que la distancia media existente entre un proceso y sus vecinos sea mínima.

Como muestra la figura 1, el embedding que utilizaremos para realizar el mapeo de un algoritmo CC-cubo sobre un hipercubo es el embedding que denominamos *identidad*, ya que se trata de realizar el embedding de dos grafos idénticos. Para el mapeo de CC-cubos sobre mallas y toros utilizaremos los embeddings *estándar* y *xor* respectivamente. Estos embeddings serán descritos más adelante.

# 3 Arquitecturas y Algoritmos Considerados

El método CALMANT puede ser aplicado sobre arquitecturas de diversa topología, dimensionalidad, modelo de comunicación y número de puertos. Concretamente este trabajo está enfocado a *hipercubos*, *mallas* y *toros* de dos y tres dimensiones con enlaces bidireccionales full-duplex bajo los modelos de un puerto (one port) y máximo número de puertos (all port). La diferencia entre el modelo de un puerto y el modelo de máximo número de puertos radica en que bajo el primer modelo los nodos no pueden enviar más de un mensaje simultáneamente, en cambio, bajo el segundo modelo los nodos pueden enviar simultáneamente tantos mensajes como enlaces tenga el nodo. Los modelos de comunicación que consideraremos son *store-and-forward* y *wormhole* (los resultados obtenidos para wormhole pueden ser extendidos de forma equivalente a *circuit-switching* y *virtual cut-through*).

En cuanto a los algoritmos, nos centraremos en un tipo determinado de algoritmos que denominaremos algoritmos *CC-cubo*. Un CC-cubo es un algoritmo con topología en hipercubo formado por $2^d$ procesos, de manera que el código ejecutado por cada proceso tiene la siguiente estructura:

```
do i = 1, K
   calcular x_i[1:N] y otros posibles
      datos locales.
   intercambiar x_i con el vecino
      en la dimension d_i.
enddo
```

donde $d_i$ es una de las dimensiones del hipercubo ($d_i \in [0, d-1]$) y donde $d_i$ no es necesariamente diferente a $d_j$.

El código consiste en $K$ iteraciones. Cada uno de estas iteraciones está formada por una fase de cálculo y otra de comunicación. En la fase de cálculo se calculan $N$ datos. Estos datos son representados por el vector $x_i[1:N]$. Después de la fase de cálculo se realiza la fase de comunicación en la cual el vector $x_i$ es intercambiado con uno de los vecinos en el hipercubo. Los vectores $x_i$ son variables locales, es decir, son diferentes para cada proceso. En cada fase de cálculo también pueden calcularse datos que no estén involucrados en la fase de comunicación. Nótese que en cada iteración del algoritmo anterior, los procesos envían un solo mensaje de longitud $N$ a uno de sus vecinos.

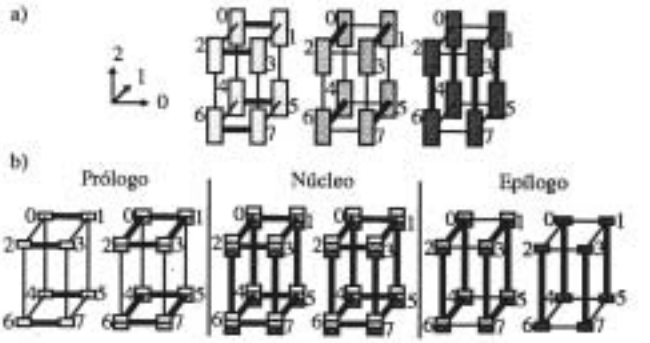


Figura 2: Ejecución de un algoritmo CC-cubo: (a) Sin segmentación de las comunicaciones; (b) Aplicando la segmentación de las comunicaciones

# 4 Segmentación de las Comunicaciones

Para explicar la técnica de la segmentación de las comunicaciones y hacer una primera evaluación cualitativa de la mejora que aporta al rendimiento de los algoritmos supondremos en esta sección que el CC-cubo se ejecuta sobre un hipercubo. Escogemos el hipercubo por simplicidad, sin embargo, la técnica puede aplicarse sobre mallas y toros tal como se verá mas adelante.

La aplicación directa de un algoritmo CC-cubo sobre un computador con topología en hipercubo, solamente utiliza una dimensión en cada iteración y el resto de dimensiones permanecen inactivas. Además, debido a que el mensaje enviado en una iteración es necesario para comenzar la iteración siguiente, no es posible explotar la posibilidad de solapar cálculo y comunicaciones.

En la figura 2.a podemos observar la aplicación directa de un algoritmo CC-cubo, con $K = d$, donde $d_1 = 0$, $d_2 = 1$, $d_3 = 2$, para el caso de un hipercubo 3-dimensional. El algoritmo consta de tres iteraciones. En cada iteración se realiza una fase de cálculo (señalado con diferente tono de gris). Una vez acabada la fase de cálculo se realiza una fase de comunicación por una dimensión diferente del hipercubo (señalada en la figura con un trazo más grueso).

Para reducir el coste de comunicación usaremos la técnica de *segmentación de las comunicaciones*. La técnica de la segmentación de las comunicaciones está basada en la idea de segmentación software propuesta por Lamm (1988) y que fue utilizada por primera por Johnsson y Krawitz (1992). Esta técnica permite reorganizar el CC-cubo de forma que, en lugar de enviar en cada iteración un mensaje grande por un enlace, se envían varios mensajes pequeños simultáneamente por varios enlaces. Para que dicha técnica pueda ser aplicada, el algoritmo CC-cubo ha de cumplir que

el cálculo de cada uno de los vectores $x_i$ sea de la siguiente manera:

```
do j = 1, N
   x_i[j] = f(xa[b], datos_locales)
      con a ≤ i, a+b ≤ j+i-1
enddo
```

Esto significa que el cálculo de $x_i[j]$ depende de sólo una parte de los vectores recibidos en iteraciones previas y de posiblemente algunos datos locales. El valor inicial del vector para cada proceso es $x_{-1}$.

La figura 2.b muestra cómo se aplica esta técnica para el caso de un hipercubo 3-dimensional. Los vectores sobre los que se realiza la fase de cálculo de cada iteración se han dividido en un número $Q$ determinado de paquetes (4 en este caso). Si el algoritmo CC-cubo cumple la propiedad mencionada anteriormente, no es necesario acabar por completo la fase de cálculo de una de las iteraciones para comenzar la fase de cálculo de la siguiente. Así pues, en la primera iteración del nuevo algoritmo, se calcula el primer paquete de $x_0$ y se envía por la dimensión 0. En la segunda iteración se puede calcular el segundo paquete de $x_0$ y el primer paquete $x_1$ ya que se dispone de los datos necesarios, los cuales fueron calculados en la iteración anterior. Una vez acabado el cálculo, el segundo paquete de $x_0$ es enviado por la dimensión 0 y simultáneamente el primer paquete $x_1$ es enviado por la dimensión 1. Así se continúa sucesivamente como muestra la figura 2.b.

Si $Q$ es mayor que el número de dimensiones del hipercubo, a partir de la iteración $d$ del algoritmo segmentado y durante $Q - d + 1$ iteraciones estaremos en condiciones de utilizar las $d$ dimensiones del hipercubo simultáneamente. El conjunto de iteraciones en las que las $d$ dimensiones del hipercubo son utilizadas de forma simultánea formarán lo que llamaremos el *núcleo*. Las $d - 1$ iteraciones anteriores al núcleo formarán el *prólogo* y las $d - 1$ iteraciones posteriores a la fase núcleo formarán el *epílogo*. Por otra parte si $Q$ es menor que el número de dimensiones del hipercubo, el prólogo estará formado por $Q - 1$ iteraciones, el núcleo estará formado por $d - Q + 1$ iteraciones, donde estaremos en condiciones de utilizar $Q$ dimensiones del hipercubo simultáneamente y el epílogo también estará formado por $Q - 1$ iteraciones. El algoritmo resultante de la aplicación de la segmentación de las comunicaciones a un CC-cubo será denominado *CC-cubo segmentado* (Díaz de Cerio *et al.*, 1996).

La técnica de la segmentación de las comunicaciones también nos permite explotar la posibilidad de solapar cálculo y comunicaciones. Nótese que en las iteraciones del algoritmo CC-cubo segmentado se calculan varios paquetes y que una vez acabado el cálculo de un paquete, éste puede comenzar a ser transmitido a la vez que se calcula el siguiente paquete (Díaz de Cerio *et al.*, 1998).

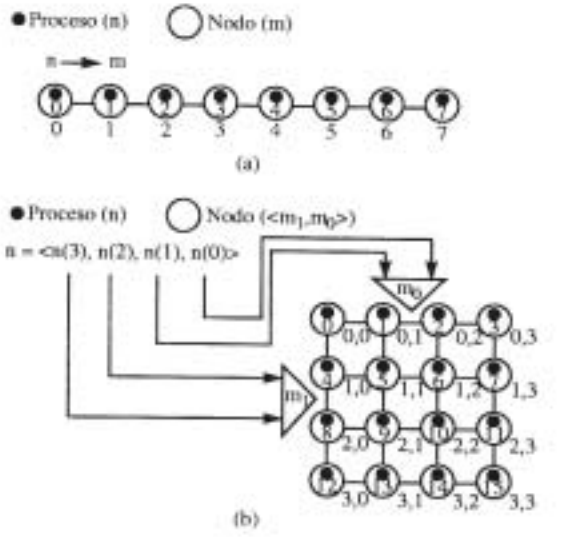


Figura 3: Embedding estándar: (a) Hipercubo de 3 dimensiones sobre una línea; (b) Hipercubo de 4 dimensiones sobre una malla de $4 \times 4$ nodos

## 5 Embeddings

El problema de programar un algoritmo CC-cubo en una malla o un toro puede verse como un problema de embedding de grafos, siendo el algoritmo el grafo origen y el multicomputador el grafo destino. En este trabajo estamos interesados en aquellos embeddings en los que se cumple que todos procesos tienen su respectivos vecinos según una determinada dimensión del hipercubo mapeados en el multicomputador a la misma distancia. Denominaremos esta propiedad como propiedad de las *distancias constantes*.

Los embeddings con distancias constantes tienen la propiedad de que todos los procesos tardan el mismo tiempo en transmitir un mensaje a su vecino en una determinada iteración del algoritmo. Debido a esto y a que la duración de las fases de cálculo son idénticas para cada proceso, se evitan los tiempos de espera asociados a desequilibrios entre los nodos en las fases de comunicación, especialmente bajo el modelo de comunicación store-and-forward donde el tiempo de comunicación es proporcional a la distancia.

Las arquitecturas en los que se enfoca este trabajo son: hipercubos, mallas y toros. El embedding que representa la ejecución de un CC-cubo sobre un hipercubo es el embedding *identidad*, ya que ambos grafos son idénticos. En mallas, el embedding que utilizaremos es el conocido como *estándar* (Matic, 1990). En el caso de toros, el embedding que utilizaremos es el denominado embedding *xor* (González *et al.*, 1995), el cual es una de las propuestas de este trabajo, ya que en el caso de toros, el embedding xor es mejor que el estándar en cuanto que la distancia media entre procesos vecinos es menor.

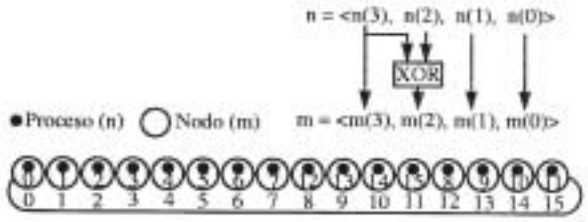


Figura 4: Embedding xor de un CC-cubo de 4 dimensiones sobre un anillo. Los procesos son etiquetados mediante n y los nodos mediante m

La aplicación del embedding estándar de un hipercubo $d$-dimensional sobre una malla $c$-dimensional puede definirse como muestra la figura 3. Matic hace una evaluación exhaustiva de dicho embedding.

El embedding estándar cumple la propiedad de las distancias constantes. Si nos fijamos en la figura 3.b podemos observar, por ejemplo, que cualquier pareja de procesos vecinos según la dimensión 1 se encuentra a distancia 2 (0-2, 1-3, 4-6, ...). Hay que resaltar que, tal y como demuestran González *et al.* (1995), el estándar embedding es óptimo en el sentido que minimiza la distancia media entre un proceso y sus vecinos.

La aplicación del embedding xor de un hipercubo $d$-dimensional sobre un toro $c$-dimensional puede definirse tal y como muestran las figuras 4 y 5, donde $\langle a$ XOR $b\rangle$ es la OR exclusiva del los bits $a$ y $b$.

El embedding xor también cumple la propiedad de las distancias constantes. Como puede verse en la figura 5, todas las parejas de procesos vecinos según la dimensión 2 se encuentran a distancia 2 (0-4, 1-5, 2-6, ...). Nótese que, en el caso del estándar embedding, las parejas de procesos según la dimensión 2 se encontrarían a distancia 4. González *et al.* (1995) también demuestran que el embedding xor es óptimo en el sentido que minimiza las distancias para el caso de anillos (toros de una dimensión).

# 6 Encaminamiento de Mensajes

El embedding determina únicamente la ubicación de los procesos del algoritmo CC-cubo segmentado en la malla o toro. Para especificar completamente el algoritmo hay que determinar cómo deben encaminarse los mensajes que han de ser intercambiados en cada una de las iteraciones. Es en este punto donde adquieren especial relevancia aspectos tales como el número de puertos, el modelo de comunicación y las posibilidades de solapamiento de cálculo y comunicaciones.

En nuestro trabajo hemos realizado un estudio exhaustivo de cómo resolver el problema del encaminamiento en cada uno de los posibles escenarios (malla/toro, un puerto/máximo número de puertos, etc.) En general, los esquemas propuestos tienden ha hacer un uso óptimo del ancho de banda de comunicación que nos ofrecen las redes de interconexión. En esta sección describimos las ideas básicas a través de ejemplos simples que permiten dar una idea de cómo ha de abordarse el problema. Para ello supondremos el modelo de comunicación wormhole sobre nodos de un solo puerto. Debido a que los encaminamientos de mensajes que proponemos están libres de conflictos, los ejemplos pueden aplicarse de igual manera bajo modelos de comunicación circuit switching o virtual cut-through.

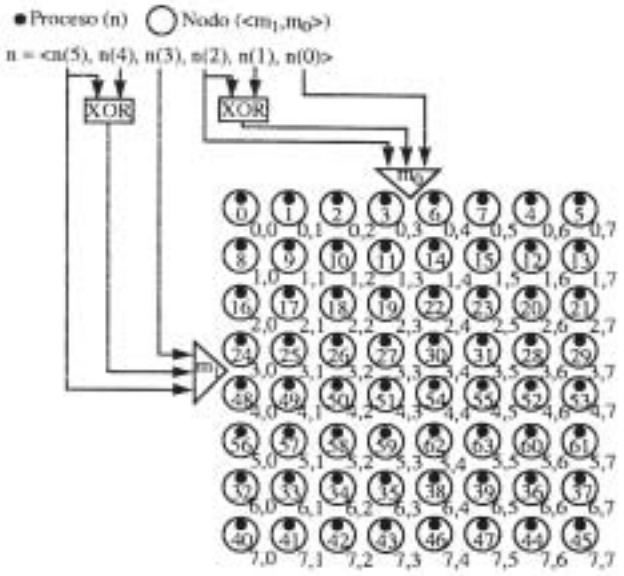


Figura 5: Embedding xor de un hipercubo de 6 dimensiones sobre un toro de $8 \times 8$ nodos. Los enlaces no se muestran por razones de claridad

## 6.1 Embedding de un CC-cubo

La figura 6.a muestra cómo se ha realizado el estándar embedding de un algoritmo CC-cubo sobre una línea de 8 nodos. En dicha figura puede verse que para comunicar todos y cada uno de los procesos con su vecino a distancia $2^i$ son necesarios $2^i$ pasos de comunicación. En general, si consideramos solamente el tiempo de transmisión y suponemos que el tamaño de los mensajes es $N$, podemos calcular el tiempo de comunicación del algoritmo como:

$$t_1 = \sum_{i=0}^{d-1} 2^i N T_e = (2^d - 1) N T_e \approx 2^d N T_e \qquad (1)$$

donde $T_e$ es el tiempo de transmisión por cada elemento del mensaje.

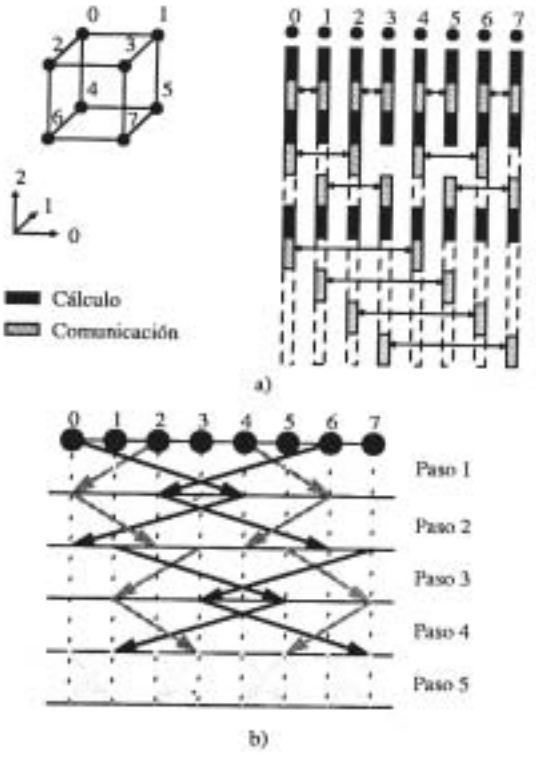


Figura 6: a) Cálculo y comunicación de un CC-cubo sobre una línea de 8 nodos; b) Ejecución de una iteración del núcleo del algoritmo CC-cubo segmentado sobre una línea

## 6.2 Embedding de un CC-cubo Segmentado

De nuevo utilizaremos el estándar embedding para ejecutar un algoritmo CC-cubo segmentado sobre una línea. Tomemos por ejemplo una iteración cualquiera del núcleo del algoritmo CC-cubo segmentado. Al final de la iteración todos y cada uno de los procesos envían un mensaje a cada uno de sus tres vecinos en el hipercubo. Según muestra la figura 6.b, cada proceso tiene a sus vecinos a distancia 1, 2 y 4. El eje vertical de la figura representa el tiempo. Cada una de las flechas representa la comunicación entre dos nodos cuyos procesos son vecinos y el diferente tono de gris de las flechas indica la dimensión del CC-cubo que se utiliza en la comunicación. La figura muestra que durante los dos primeros pasos de comunicación hay una serie de nodos que realizan la comunicación por las dimensiones 1 y 2 y en los dos siguientes pasos los nodos que hasta ahora permanecían inactivos realizan la comunicación también por las dimensiones 1 y 2. Una vez que todos los nodos han realizado la comunicación por estas dos dimensiones comienza la comunicación por la dimensión 0. En el caso de la dimensión 0 solamente es necesario un paso para que todos los nodos realicen la comunicación. En el caso un número cualquiera de dimensiones tomaríamos éstas de dos en dos comenzando por aquellas de mayor distancia y en orden descendente. Nótese en la figura que, en cualquier paso de comunicación, la mayoría de los enlaces son atravesados por dos mensajes (uno en cada dirección) es decir, que el uso de la red de interconexión es mucho más eficiente. En general, para comunicar todos los procesos de un CC-cubo $d$-dimensional con sus respectivos vecinos en las dimensiones $i, i+1, ..., i+M-1$ serán necesarios $(2^{i+M+1} - 2^{i+1})/3$ pasos de comunicación si $M$ es par o $(2^{i+M+1} - 2^{i})/3$ pasos de comunicación si $M$ es impar.

Si contabilizamos los pasos de comunicación del prólogo, del núcleo y del epílogo y suponemos que $Q < d$, estos resultan ser en total:

$$\frac{(3Q+1)2^{d+2} - 2^{d-Q+2} - 9z}{18} \tag{2}$$

donde $z = Q$ si $Q$ es par y $z = (Q+1)$ si $Q$ es impar.

Si suponemos que $Q \geq d$, el número total de pasos de comunicación es:

$$\frac{(3Q+1)2^{d+2} + 3d - 12Q - z}{18} \tag{3}$$

donde $z = 4$ si $d$ es par y $z = 5$ si $d$ es impar.

Suponiendo ahora que $Q$ es lo suficientemente grande ($Q \gg d$) podemos suponer que el coste debido al prólogo y al epílogo es despreciable respecto al coste del núcleo. Teniendo en cuenta solamente el tiempo de transmisión y sabiendo que el tamaño de los mensajes es $N/Q$ podemos expresar el tiempo de comunicación del algoritmo CC-cubo segmentado como:

$$t_2 \approx (Q-d+1)\frac{2^{d+1}N}{3Q}T_e \approx \frac{2^{d+1}}{3}NT_e \tag{4}$$

Si comparamos este resultado con el obtenido anteriormente para el caso del algoritmo CC-cubo sin segmentación resulta que podemos expresar la mejora del rendimiento obtenido al aplicar el método CALMANT como $r = t_1/t_2 = 1.5$, lo cual significa una importante reducción en el tiempo de comunicación. En general, para mallas de c dimensiones resulta que la mejora de rendimiento es del orden de $r = 1.5 \cdot c$.

Cuanto mayor es el parámetro $Q$ mejor uso se hace de la red de interconexión, en cambio al aumentar el número de paquetes a transmitir el coste debido a la inicialización de los mensajes también aumenta. Existe un compromiso que deberá ser resuelto mediante la elección de un $Q$ óptimo y minimizar así el coste de comunicación.

El ejemplo aquí presentado es un caso simplificado que nos muestra cómo el encaminamiento de los mensajes tiene una relevancia importante a la hora de especificar los algoritmos. El estudio completo del encaminamiento para mallas y toros de varias dimensiones es mucho mas extenso y puede encontrarse en la tesis doctoral de Díaz de Cerio (1998).

# 7 Figuras de Rendimiento

El método CALMANT ha sido aplicado a problemas tales como la *FFT*, el *complete exchange* y el *cálculo de valores y vectores propios de una matriz*, todos ellos sobre diferentes tipos de arquitectura (Díaz de Cerio *et al.*, 1995, 1996, 1998; Royo *et al.*, 1998). En anteriores trabajos se ha presentado la metodología CALMANT en una versión inicial para toros, sin aplicar la segmentación de las comunicaciones. Posteriormente se aplicó la segmentación a hipercubos síncronos y asíncronos (estos últimos con posibilidad de solapar cálculo y comunicación). En esta sección presentaremos algunas de las principales contribuciones, relacionadas con la aplicación de CALMANT a mallas y haciendo uso de la segmentación. Las gráficas están obtenidas a partir de modelos analíticos del tiempo de ejecución.

En primer lugar consideraremos la aplicación del método CALMANT al problema de la FFT. Se ha comparado el rendimiento de nuestro método con el método propuesto por Aykanat y Dervis (1991) para la resolución de la FFT sobre un hipercubo de dimensión $d = 10$ con capacidad para solapar cálculo y comunicaciones. La gráfica de la figura 7.a muestra el rendimiento de ambos métodos respecto al tiempo de ejecución del algoritmo CC-cubo en función del tamaño del problema ($2^n$). En el eje horizontal se representa el valor de $n$ y en el eje vertical se muestra el rendimiento que resulta de dividir el tiempo del algoritmo CC-cubo ($T_{CC}$) por el tiempo de las diferentes propuestas ($T_{CALMANT}$, $T_{Aykanat-Dervis}$). Además la gráfica presenta una cota que no puede ser superada ($T_{CC}/T_{cota-inferior}$) para mostrar lo próximos que nos encontramos al rendimiento óptimo. Suponemos un tiempo de inicialización de los mensajes ($T_{sup}$) 10 veces superior al tiempo de transmisión por elemento y suponemos que este tiempo es a su vez 100 veces el tiempo de cálculo ($T_a$) de una operación aritmética (con el fin de obtener una relación de solapamiento cálculo-comunicación). La propuesta de Aykanat y Dervis, a parte de ser una propuesta específica para la FFT, explota el solapamiento pero no la segmentación de las comunicaciones. Es por esta razón que nuestro método resulta ser más eficiente.

Al aplicar el método CALMANT al algoritmo de comunicación complete exchange sobre mallas y toros también se han obtenido resultados interesantes. En la figura 7.b podemos encontrar un gráfica para toros de tres dimensiones donde se compara nuestra propuesta con las propuestas por Takkella y Seidel (1994) y por Tseng y Gupta (1996). La gráfica muestra el rendimiento de los algoritmos resultantes en función del volumen de datos ($2^m$) que cada nodo envía a otro. De nuevo, en el eje horizontal se presenta el tamaño del problema ($m$) y en el eje vertical aparece el rendimiento, como el tiempo de ejecución del algoritmo CC-cubo ($T_{CC}$), partido por el tiempo de las diferentes propuestas ($T_{CALMANT}$, $T_{Takkella-Seidel}$, $T_{TsengyGupta}$). En este caso estamos suponiendo una relación de 1000 entre el

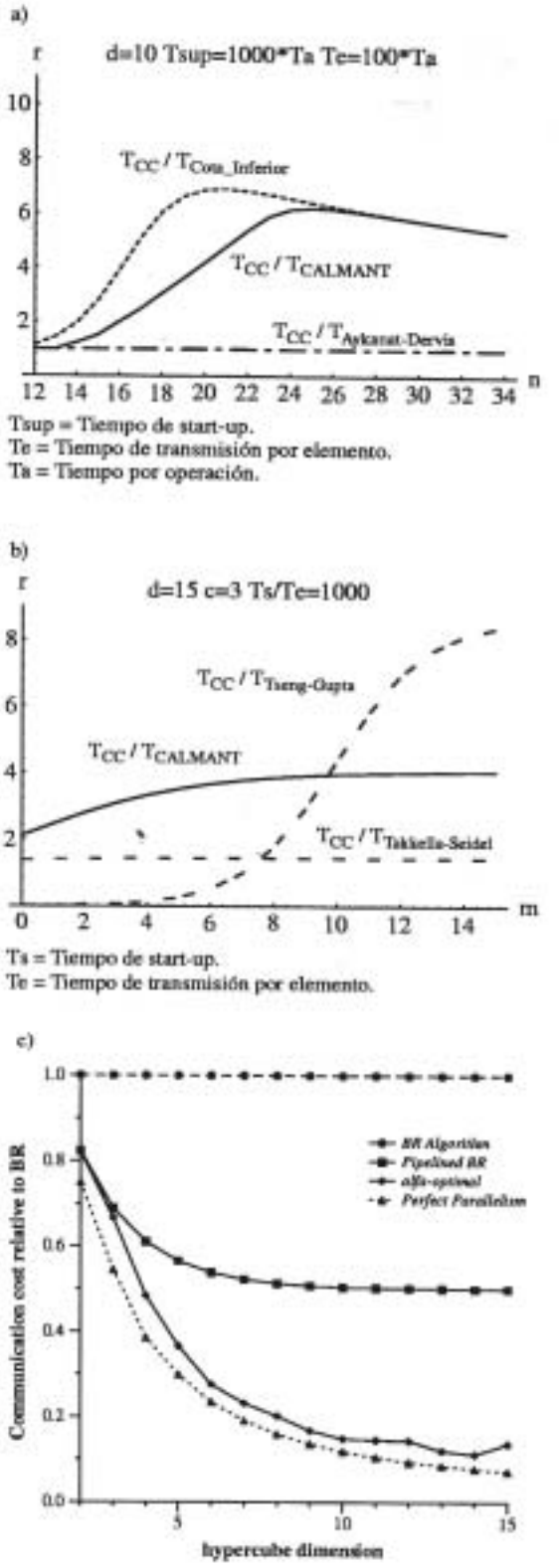


Figura 7: a) Algoritmo FFT sobre un hipercubo; b) Algoritmo complete exchange sobre un toro; c) Algoritmo BR sobre un hipercubo

tiempo de inicialización de los mensajes ($T_s$) y el tiempo de transmisión de un elemento ($T_e$). Hay que destacar que la aplicación de nuestra propuesta se realiza de forma sistemática sobre diferentes arquitecturas y consigue, en muchas ocasiones, mejores resultados que las propuestas específicas para arquitecturas concretas que otros autores hacen.

Por último, el método propuesto también ha sido aplicado al algoritmo de Jacobi para el cálculo de valores y vectores propios sobre un hipercubo. La figura 7.c muestra el coste de comunicación relativo cuando se aplica el método CALMANT (*pipelined BR*) y cuando aplicamos dicho método sobre una modificación del algoritmo también propuesta por nuestro grupo (*alfa-optimal*). En dicha gráfica aparecen otras dos curvas que representan el algoritmo de Jacobi sin segmentación (*BR algorithm*) y una cota mínima (*perfect parallelism*). Todo ello en función de la dimensionalidad del hipercubo.

## 8 Conclusiones

En este trabajo se ha presentado el método CALMANT para la planificación sistemática de algoritmos con topología en hipercubo en sistemas multiprocesador. Como ejemplos de aplicabilidad del método y de resultados obtenidos, hemos considerado tres problemas diferentes (*FFT, complete exchange* y *cálculo de valores y vectores propios de una matriz*). Al comparar con las propuestas de otros autores para la solución de estos problemas, se puede comprobar que CALMANT ofrece mejores resultados para muchos casos. No hay que olvidar que nuestra propuesta es general y aplicable a muchos problemas diferentes y que las propuestas de los otros autores con los que nos hemos comparado son específicas para un problema y un tipo de arquitectura concretos. A lo largo del trabajo, se ha tratado de dar una idea intuitiva, evitando al máximo posible grandes formulismos en la obtención de los modelos analíticos del tiempo de ejecución. Para aquellos lectores que deseen un detalle completo, pueden dirigirse a la tesis de Díaz de Cerio (1998).

## Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Educación y Ciencia (CICYT TIC-98/0511) y por el Centro Europeo de Paralelismo de Barcelona (CEPBA).

## Referencias

**Aykanat C.** and **Dervis A.**, "An Overlapped FFT Algorithm for Hypercube Multicomputers", in proceedings of *International Conference on Parallel Processing*, 1991, pp. II-316 - III-317.

**Díaz de Cerio L.**, "CALMANT: Un Método Sistemático para la Ejecución de Algoritmos con Topología Hipercubo en Multicomputadores", Tesis doctoral, ftp: // ftp.ac.upc.es / pub / reports / DAC / 2000 / UPC-DAC-2000-13.ps.Z, Diciembre, 1998.

**Díaz de Cerio L.**, **González A.** and **Valero-García M.**, "Communication Pipelining in Hypercubes", *Parallel Processing Letters*, Vol. 6, No. 4, December, 1996, pp. 507-523.

**Díaz de Cerio L.**, **Valero-García M.** and **González A.**, "A Study of the Communication Cost of the FFT on Torus Multicomputers", in proceedings of the *IEEE First International Conference on Algorithms And Architectures for Parallel Processing (ICA3PP)*, April, 1995, pp. 131-140.

**Díaz de Cerio L.**, **Valero-García M.** and **González A.**, "A Method For Exploiting Communication/Computation Overlap in Hypercubes", *Parallel Computing*, Vol.24, No.2, June, 1998, pp.221-245.

**Royo D.**, **González A.** and **Valero-García M.**, "Jacobi Orderings for Multi-Port Hypercubes", in proceedings of the *12th. Int. Parallel Processign Symposium (IPPS'98)* and *9th Symposium on Parallel and Distributed Processing*, March, 1998, pp. 88-98.

**González A.**, **Valero-García M.** and **Díaz de Cerio L.**, "Executing Algorithms with Hypercube Topology on Torus Multicomputers", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, No. 8, August 1995, pp. 803-814.

**Johnsson S.L.** and **Krawitz R.L.**, "Cooley-Tukey FFT on the Connection Machine", *Parallel Computing*, 18, 1992, pp. 1201-1221.

**Lam M.**, "Software Pipelining: An Effective Scheduling Technique for VLIW machines", in proceedings of the *Conf. on Programming Language Design and Implementation*, June, 1988, pp. 318-328.

**Matic S.**, "Emulation of Hypercube Architecture on Nearest-Neighbor Mesh-Connected Processing Elements", *IEEE Trans. on Computers*, Vol. 39, No. 5, May, 1990, pp. 698-700.

**Takkella S.** and **Seidel S.**, "Complete Exchange and Broadcast Algorithms for Meshes", in proceedings of the *IEEE Scalable High Performance Computing Conf.*, 1994, pp. 422-428.

**Tseng Y.** and **Gupta S.K.S.**, "All-to-All Personalized Communication in a Wormhole-Routed Torus", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 5, May, 1996, pp. 498-505.

***Luis Díaz de Cerio*** *recibió el título de ingeniero en Telecomunicación en 1993 y el título de doctor en Ingeniería de Telecomunicación en 1998, ambos títulos concedidos por la Universitat Politècnica de Catalunya en Barcelona (Spain). En estos momentos es profesor asociado en el Departamento de Arquitectura de Computadores en la Universidad Politècnica de Catalunya.*
*Actualmente dedica su investigación a temas relacionados con algoritmos para sistemas ultiprocesador, sistemas de computación distribuidos, el Grid y técnicas para el diseño de procesadores de bajo consumo.*

***Miguel Valero-García*** *recibió el título de ingeniero en Informática en 1986 y el título de doctor en Informática en 1989, ambos títulos concedidos por la Universitat Politècnica de Catalunya en Barcelona (Spain). En estos momentos es profesor titular en el Departamento de Arquitectura de Computadores en la Universitat Politècnica de Catalunya. Actualmente dedica su investigación a temas relacionados con algoritmos para sistemas multiprocesador y técnicas para la mejora de la calidad docente en la universidad.*

***Antonio González*** *recibió el título de ingeniero en Informática en 1986 y el título de doctor en Informática en 1989, ambos títulos concedidos por la Universitat Politècnica de Catalunya en Barcelona (Spain). En estos momentos es profesor titular en el Departamento de Arquitectura de Computadores en la Universitat Politècnica de Catalunya. Actualmente dedica su investigación a temas relacionados con algoritmos para sistemas multiprocesador, jerarquía de memoria, ejecución especulativa y técnicas para el diseño de procesadores de bajo consumo.*

***Dolors Royo*** *recibió el título de ingeniera en Informática en 1989 y el título de doctora en Informática en 1999, ambos títulos concedidos por la Universitat Politècnica de Catalunya en Barcelona (Spain). En estos momentos es profesora titular en el Departamento de Arquitectura de Computadores en la Universitat Politècnica de Catalunya. Actualmente dedica su investigación a temas relacionados con algoritmos para sistemas multiprocesador, sistemas distribuidos de computación y el Grid.*