



SAPIENZA
UNIVERSITÀ DI ROMA

Realizzazione di comunicazione all-to-all in sistemi multiprocessore attraverso reti Fat Tree

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Informatica

Candidato

Federica Magliocca

Matricola 1754777

Responsabile

Prof.ssa Annalisa Massini

Corresponsabile

Dr. Daniele Izzi

Anno Accademico 2020/2021

Tesi non ancora discussa

Realizzazione di comunicazione all-to-all in sistemi multiprocessore attraverso reti Fat Tree

Tesi di Laurea. Sapienza – Università di Roma

© 2021 Federica Magliocca. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Email dell'autore: magliocca.1754777@studenti.uniroma1.it

Ringraziamenti

Finalmente il giorno è arrivato: scrivere queste frasi di ringraziamento è il tocco finale della mia tesina. È stato un periodo di profondo apprendimento, non solo a livello scientifico, ma anche personale. Vorrei spendere due parole di ringraziamento nei confronti di tutte le persone che mi hanno sostenuto e aiutato durante questo periodo.

Un ringraziamento particolare va al mio responsabile di tirocinio, la professoressa Massini, per i suoi preziosi consigli. Mi ha fornito tutti gli strumenti di cui avevo bisogno per intraprendere la strada giusta e portare a compimento la mia tesina. Senza di lei questo lavoro non avrebbe preso vita! Ha visto il mio elaborato nascere, crescere e svilupparsi, rappresentando il mio punto di riferimento per ogni dubbio, domanda, consiglio o approfondimento.

Un sentito grazie al Dott. Daniele Izzi, corresponsabile, per il supporto costante e indispensabile nella realizzazione di ogni capitolo della mia tesina.

Ringrazio immensamente i miei genitori che mi hanno sempre sostenuto, appoggiando ogni mia decisione, fin dalla scelta del mio percorso di studi.

Un infinito grazie alla mia migliore amica, “la mia mamma”: sei la mia più fidata consigliera e il mio punto di riferimento. Prima di ogni esame mi dicevi sempre: “In bocca al lupo e, mi raccomando, stai tranquilla!” e al sentire quelle parole io rispondevo sempre “Crepi il lupo si, ma tranquilla proprio no!”. Che mi irritava quel tuo incoraggiamento, ma che l’ho sempre aspettato prima di ogni esame, perché anche se non l’ho mai ammesso mi tranquillizzava sentirselo dire. Mi hai sempre sostenuta nell'affrontare ogni difficoltà, mi hai consigliato nelle scelte più difficili, mi hai asciugato le lacrime durante le sconfitte, mi hai sgridata per spronarmi a dare il massimo, sempre!

Ringrazio mio fratello, sempre pronto ad ascoltarmi e a darmi consigli. A cercare di incoraggiarmi a camminare ogni giorno a testa alta senza aver paura dei giudizi degli altri. Ogni volta che ho avuto bisogno di te sei stato sempre presente. Ci siamo sempre sostenuti a vicenda, nella buona e nella cattiva sorte, sia durante le fatiche e lo sconforto che hanno caratterizzato i nostri percorsi che nei momenti di gioia e soddisfazione al raggiungimento del traguardo. Grazie perché senza di voi non sarei mai arrivata fino in fondo a questo difficile, lungo e tortuoso cammino. Questa tesina la dedico a voi che siete la mia famiglia, il mio più grande sostegno.

Vorrei ringraziare tutti i miei amici: da quelli che porto nel mio cuore dall’infanzia, Beatrice, Eleonora e Mariana, a quelli che ho incontrato nel mio cammino, Luciano, e che hanno deciso di camminare al mio fianco e di portarmi fino a qui. A tutti voi, così diversi ma così importanti, ognuno per ragioni uniche e speciali, voglio esprimere la mia più assoluta gratitudine.

Infine, vorrei dedicare questo piccolo traguardo a me stessa, che possa essere l'inizio di una lunga e brillante carriera professionale.

Un sentito grazie a tutti!

Sommario

I sistemi distribuiti rappresentano un'interessante architettura per applicazioni parallele. In questi sistemi la memoria è distribuita tra i processori che hanno la necessità di comunicare tra loro per scambiare dati attraverso una rete di interconnessione.

Il possibile sovraccarico di comunicazione tra processori limita le prestazioni di tali sistemi e ha reso necessario lo sviluppo di architetture di rete e algoritmi di instradamento sempre più efficienti.

Tra le possibili richieste di comunicazione vi sono: one-to-one (unicast), one-to-many (multicast), one-to-all (broadcast) e all-to-all, che a sua volta si distingue in all-to-all broadcast, in cui ogni processore invia lo stesso messaggio ad ogni altro processore, e all-to-all personalizzata, dove ogni processore invia un messaggio diverso ad ogni altro processore.

La comunicazione all-to-all personalizzata (chiamata anche complete exchange o totale exchange) è sicuramente la più onerosa in quanto a prestazioni e larghezza di banda, ma è molto utilizzata, perché rappresenta un passo intermedio in algoritmi scientifici paralleli, come la trasposizione di matrice e la trasformata di Fourier veloce (FFT).

La comunicazione all-to-all è stata molto studiata sulle topologie di reti più comuni, quali mesh, tori, hypercube, e su reti di interconnessione multistadio.

In questa relazione ci si concentrerà sulla comunicazione all-to-all personalizzata realizzata su una particolare topologia di rete multistadio, la rete Fat-Tree.

Inoltre si esploreranno reti Fat-Tree di dimensioni ridotte, chiamate Slimmed Fat-Tree, e si analizzerà come realizzare una comunicazione all-to-all personalizzata anche su quest'ultime.

La topologia Fat-Tree è una rete ad albero in cui le dimensioni dei collegamenti tra i nodi aumentano salendo verso la radice.

Nonostante ciò garantisca un'elevata connettività, nodi di così grandi dimensioni sono difficilmente realizzabili. Per questo motivo la radice e i nodi discendenti vengono replicati usando nodi di dimensione minore, mantenendo invariato il numero complessivo di archi.

Le reti Fat-tree così ottenute sono molto utilizzate nei sistemi di elaborazione ad alte prestazioni, grazie alla loro elevata scalabilità, simmetria e tolleranza ai guasti.

Nel capitolo 1 si introdurranno alcune topologie di rete molto comuni, come mesh, ring e hypercube, e le reti multistadio.

Il capitolo 2 si concentrerà sulle reti di Clos e su come da esse possano essere derivate le reti Fat-Tree. Sarà definita la topologia Fat-Tree attraverso il modello di Generalized Fat-Tree, con il quale si farà una distinzione tra le rete Fat-Tree, Slimmed Fat-Tree e Fattened Fat-Tree.

Nel capitolo 3 sarà illustrata la comunicazione all-to-all broadcast e all-to-all personalizzata su reti mesh, ring, hypercube e multistadio. In particolare sulle reti multistadio la comunicazione all-to-all personalizzata viene ottenuta realizzando insiemi di permutazioni che formano un Quadrato Latino.

Infine gli ultimi 2 capitoli descriveranno la parte principale del lavoro di tirocinio, cioè la realizzazione della comunicazione all-to-all personalizzata su Fat-Tree e Slimmed Fat Tree realizzata instradando N permutazioni che formano un Quadrato Latino.

Indice

Ringraziamenti	ii
Sommario	iii
1 Introduzione alle reti	1
1.1 Topologie di rete	1
1.1.1 Ring	2
1.1.2 Rete completamente connessa	2
1.1.3 Mesh	3
1.1.4 Torus	3
1.1.5 Hypercube	4
1.1.6 Reti a stadi	4
2 Dalle reti di Clos ai Fat-Tree	6
2.1 Reti di Clos	6
2.1.1 Rete bloccante	7
2.1.2 Rete non bloccante	7
2.1.3 Reti di Clos a 3 stadi	7
2.1.4 Conversione di una rete Clos a 3 stadi in 5 stadi	9
2.1.5 Da rete di Clos a Fat-Tree	9
2.2 Fat-Tree	10
2.2.1 Alberi	10
2.2.2 Fat tree multistage	11
2.2.3 Generalized Fat-Tree	12
2.2.4 Fattened e Slimmed Fat-Tree	13
3 Comunicazione all-to-all	15
3.1 All-to-all broadcast	15
3.1.1 Ring	15
3.1.2 Mesh	16
3.1.3 Hypercube	16
3.2 All-to-all personalizzata	17
3.2.1 Ring	17
3.2.2 Mesh	17
3.2.3 Hypercube	18
3.2.4 Reti a stadi	19

4 Comunicazione all-to-all personalizzata su Fat tree attraverso LLS	23
4.1 All-to-all personalizzata su Fat Tree	23
4.1.1 Configurazioni dei nodi	24
4.1.2 Rotazioni	25
4.1.3 Algoritmo di routing	27
4.2 All-to-all personalizzata su Slimmed Fat Tree	30
4.2.1 Configurazioni dei nodi	30
4.2.2 Rotazioni	31
4.2.3 Algoritmo di routing	35
5 Comunicazione all-to-all personalizzata su Fat-Tree attraverso CLS	40
5.1 All-to-all personalizzata su Fat-Tree	40
5.1.1 Configurazioni dei nodi	41
5.1.2 Canonical Latin Square	42
5.1.3 Partizione delle permutazioni in quadrati latini	44
5.2 All-to-all personalizzata su Slimmed Fat-Tree	47
5.2.1 Configurazione dei nodi	47
5.2.2 Canonical Latin Square	48
5.2.3 Partizione delle permutazioni in quadrati latini	51
Conclusioni	54
Bibliografia	55

Capitolo 1

Introduzione alle reti

Le reti di interconnessione sono usate per consentire scambi di dati o condivisione di risorse comuni in sistemi multiprocessore nei Data Center e in sistemi ad alte prestazioni (HPC systems). Una rete può essere a commutazione di circuito o a commutazione di pacchetto:

- In una rete a commutazione di circuito (circuit switching) viene messo a disposizione dei due processori che devono comunicare un canale; questa tecnica funziona come una comunicazione telefonica dove c'è una linea diretta fra gli interlocutori e il messaggio ha a sua disposizione un intero cammino da fare che lo porta direttamente al destinatario. Lo svantaggio di questa tecnica è dato dal fatto che se c'è un'altra richiesta che vuole usare quella linea deve prima aspettare che si liberi.
- In una rete a commutazione di pacchetto (packet switching) ogni messaggio è diviso in piccoli pacchetti e ogni pacchetto può seguire un instradamento diverso. Lo svantaggio di queste reti è dato dal fatto che la commutazione dei pacchetti è complicata, poiché è necessario controllare che i pacchetti arrivino nell'ordine in cui sono stati inviati. In questa relazione verranno considerate soltanto le reti a commutazione di circuito.

1.1 Topologie di rete

La topologia di rete è il modello geometrico (grafo) che rappresenta le relazioni di connettività tra gli elementi che costituiscono una rete. Il concetto di topologia si applica a qualsiasi tipo di rete di telecomunicazioni: telefonica, rete di computer, Internet.

Gli elementi fondamentali di una topologia di rete sono:

- i nodi, che rappresentano dispositivi hardware in grado di comunicare tra loro.
- i rami (o archi), che evidenziano le relazioni di connettività tra i nodi.

La topologia viene rappresentata quindi sotto forma di grafo in cui i nodi, in grado di scambiarsi direttamente l'informazione, sono collegati tra loro tramite uno o più archi.

Si distingue tra due tipi di topologie, logica e fisica:

- la topologia logica descrive il modo in cui i nodi della rete si scambiano i dati.

- la topologia fisica spiega come si distribuiscono i nodi nello spazio e come essi sono connessi tra loro.

Di seguito verranno illustrate alcune delle topologie più diffuse, come le topologie ad anello o ring, completamente connessa, mesh, torus e hypercube. Successivamente verranno introdotte le reti di interconnessione multistadio.

1.1.1 Ring

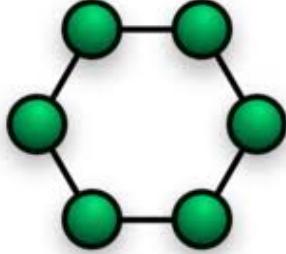


Figura 1.1. Rete ad anello

La topologia ring (o ad anello) mostrata in Figura 1.1, porta questo nome proprio per la sua forma circolare, infatti tutti i nodi sono connessi in un cerchio e ogni nodo ha due vicini. Tutti i nodi analizzano i dati che ricevono. Un nodo accetta un'informazione soltanto se è destinata ad esso, altrimenti la inoltra al nodo successivo. Le topologie ad anello sono molto diffuse per via dell'alta tolleranza ai guasti dato che l'informazione trasmessa può viaggiare in entrambi i versi dell'anello per raggiungere una certa destinazione, e non necessita di un nodo centrale per gestire la connessione tra gli elementi.

Tali reti consentono inoltre di ottimizzare l'utilizzo della banda disponibile, per esempio inviando alcuni pacchetti in un verso e altri pacchetti nel verso opposto, bilanciando così l'impiego delle risorse e limitando la possibilità che una parte dell'anello risulti congestionata mentre l'altra parte è scarica.

1.1.2 Rete completamente connessa

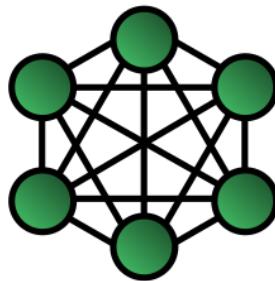


Figura 1.2. Rete completamente connessa

La topologia completamente connessa, mostrata in Figura 1.2, prevede che ogni nodo sia collegato direttamente con tutti gli altri nodi della rete. La relazione tra

numero di nodi e il numero di archi è data da: $R = N(N - 1)/2$. La caratteristica più importante di questa rete è che, dato un nodo qualsiasi, esiste sempre un arco che consente di collegarlo a un altro nodo qualsiasi della rete.

1.1.3 Mesh

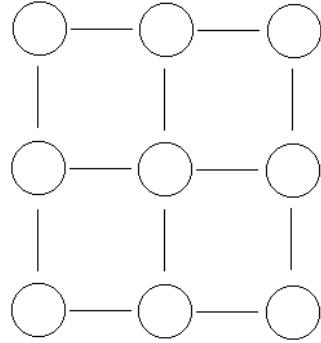


Figura 1.3. Mesh

Una topologia mesh con n nodi ha $2(n - \sqrt{n})$ archi ed è costruita in modo tale che la massima distanza tra i nodi sia $2\sqrt{n} - 1$. Questa rete ha la forma di una griglia dove i nodi sono collegati tra loro sia orizzontalmente che verticalmente. Le reti mesh sono molto utilizzate quando si hanno un gran numero di processori, in alternativa alle reti Hypercube. La Figura 1.3 mostra una rete mesh a 9 nodi.

1.1.4 Torus

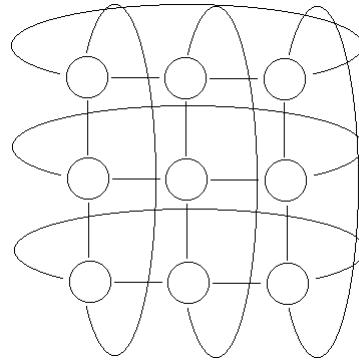


Figura 1.4. Torus

La topologia torus, in Figura 1.4, è costruita a partire da una rete mesh aggiungendo degli archi tra i nodi che si trovano all'estremità. In questo modo il cammino di un pacchetto nel caso peggiore viene migliorato di un fattore 2 rispetto a quello che si ha nella mesh.

1.1.5 Hypercube

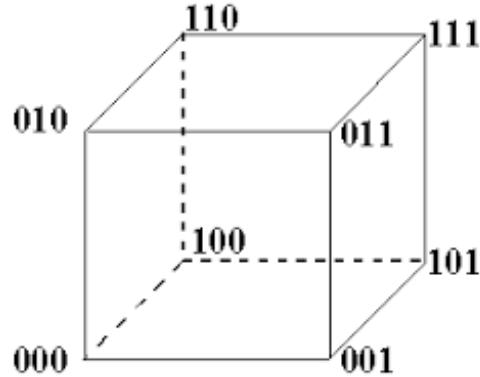


Figura 1.5. Hypercube

Un ipercubo o hypercube, in Figura 1.5, si ottiene collegando N nodi, dove N può essere espresso come: $N = 2^m$ dove 2^m , è il numero di bit necessari per etichettare i nodi della rete.

La rete è costruita collegando i nodi che differiscono solo di un bit nella loro rappresentazione binaria. Una rete ipercubo 3D può essere creata duplicando due reti 2D e aggiungendo un bit nella posizione più significativa. Il nuovo bit aggiunto è 0 per un ipercubo 2D e 1 per l'altro ipercubo 2D. Gli angoli dei rispettivi ipercubi 2D sono collegati per creare la rete dell'ipercubo 3D. Questo metodo può essere utilizzato per costruire qualsiasi ipercubo rappresentato da m bit con un ipercubo rappresentato da $m - 1$ bit.

1.1.6 Reti a stadi

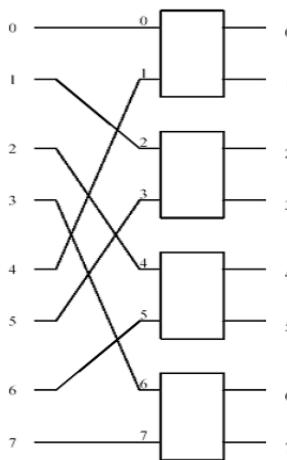


Figura 1.6. Rete a stadio singolo shuffle 8x8

In una rete di interconnessione a stadio singolo, i nodi di ingresso sono collegati all'uscita tramite un singolo stadio di interruptori o switch.

Come si può vedere in Figura 1.6, da un singolo shuffle, non tutti gli input possono raggiungere tutti gli output. Sono necessari più stadi shuffle per collegare tutti gli ingressi a tutte le uscite.

Una rete di interconnessione multistadio è formata collegando in cascata più reti a stadio singolo. Gli switch possono utilizzare il proprio algoritmo di routing o essere controllati da un router centralizzato per realizzare le richieste di connessione tra gli ingressi e le uscite.

Molto diffuse sono le reti costituite da $\log_2 N$ stadi, come Omega, Baseline, Butterfly in Figura 1.7, e il loro rovescio. Altre reti multistadio molto studiate ed utilizzate sono le reti di Clos, che hanno tipicamente 3 stadi, e la rete di Benes, che consiste di $2 \log N - 1$ stadi.

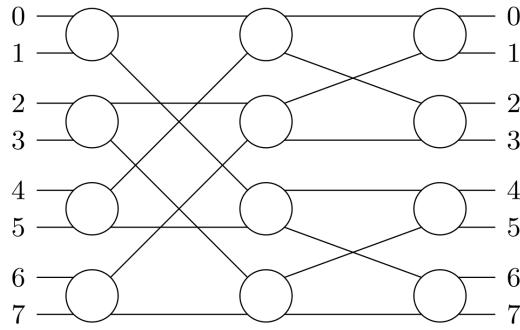


Figura 1.7. Rete Butterfly

In una MIN con $\log_2 N$ stadi, ogni nodo appartenente allo stadio j , con $0 < j < \log N - 1$, è connesso con due nodi dello stadio $j - 1$ e due nodi dello stadio $j + 1$, secondo una regola dipendente dalla topologia di rete. Ogni nodo nello stadio $j = 0$ è connesso con una coppia di ingressi e ogni nodo nello stadio $j = \log N - 1$ è connesso con una coppia di uscite [5].

Ogni nodo ha due porte di ingresso, due porte di uscita e può assumere due configurazioni: straight o cross come mostrato in Figura 1.8.

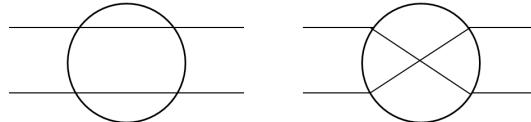


Figura 1.8. Esempio di switch con due ingressi e due uscite.

Capitolo 2

Dalle reti di Clos ai Fat-Tree

2.1 Reti di Clos

Questa topologia di rete è stata progettata da Edson Erwin nel 1938 e poi formalizzata da Charles Clos nel 1953 [3]. La rete di Clos, Figura 2.1, è una rete a commutazione di circuito multistadio.

Si tratta di una topologia di rete ampiamente utilizzata oggi nei Data Center.

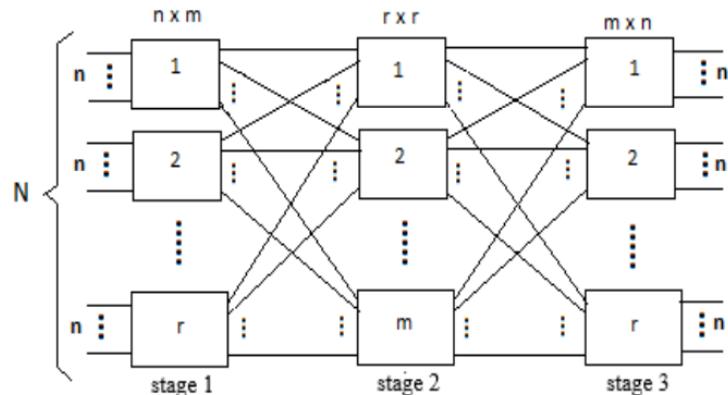


Figura 2.1. Rete di Clos [11]

Le reti di Clos hanno tre stadi:

- lo stadio di ingresso
- lo stadio intermedio
- lo stadio di uscita

Quando si attraversa uno degli switch dello stadio di ingresso, viene selezionato uno switch intermedio e il pacchetto viene instradato attraverso tale switch al relativo switch di uscita.

Per comprendere al meglio l'idea di Clos è necessario prima definire cosa si intende con rete bloccante e rete non bloccante.

2.1.1 Rete bloccante

Il concetto di rete bloccante si presenta soltanto nel caso di rete a commutazione di circuito, poiché la risorsa è vincolata durante la durata di una sessione e non è disponibile per altre sessioni.

Si supponga di avere un nodo di input con due sorgenti, A e B , collegato tramite un unico percorso ad un nodo di output con le destinazioni C e D come in Figura 2.2. Si supponga ora che tale percorso sia già impegnato in una comunicazione tra la sorgente A e la destinazione C . A questo punto se la sorgente B volesse comunicare con D non ci sarebbe nessun percorso libero disponibile per instradare la sua richiesta. Questo è un esempio di rete bloccante.



Figura 2.2. Rete bloccante [11]

2.1.2 Rete non bloccante

Una rete non bloccante è una rete in cui esistono abbastanza percorsi da poter soddisfare tutte le richieste delle sorgenti. Infatti, facendo riferimento all'esempio precedente, se esistesse un altro percorso tra gli interruttori input ed output, B potrebbe comunicare con D attraverso quest'ultimo percorso, come mostrato in Figura 2.3.

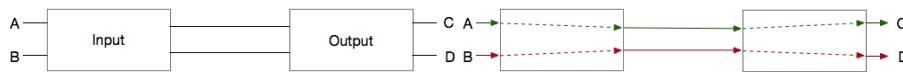


Figura 2.3. Rete non bloccante [11]

Tuttavia ottenere una rete non bloccante di questo tipo richiederebbe un costo elevato. Per questo motivo è necessario fare un ulteriore distinzione:

- Una rete è strettamente non bloccante (o non bloccante in senso stretto) se può instradare una richiesta dai percorsi disponibili senza modificare le connessioni esistenti.
- Invece si parla di rete non bloccante "rearrangeable" quando "dato qualsiasi insieme di richieste in corso e qualsiasi coppia di terminali inattivi, le richieste esistenti possono essere riassegnate su nuove rotte (se necessario) in modo da rendere possibile il collegamento della coppia inattiva" [2].

2.1.3 Reti di Clos a 3 stadi

Si supponga di avere una rete non bloccante a due stadi con n ingressi e r switch nei due stadi come in Figura 2.4.

Si supponga ora di aggiungere uno stadio intermedio tra gli switch di input e output, come in Figura 2.5, dove ogni switch nello stadio intermedio deve essere collegato a ogni switch di ingresso e uscita, ciò significa che le dimensioni degli switch dello stadio intermedio devono essere $r \times r$.

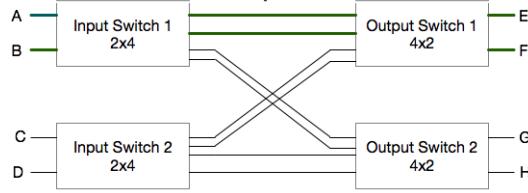


Figura 2.4. Rete a 2 stadi con $n=2$ e $r=2$ [11]

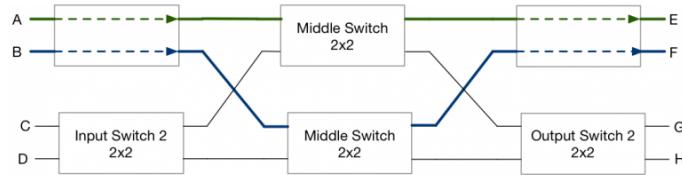


Figura 2.5. Rete di Clos a 3 stadi non bloccante "rearrangeable" [11]

In questo modo si ottiene una rete non bloccante a tre stadi in grado di instradare tutte le richieste dalle sorgenti modificando le connessioni, cioè una rete non bloccante "rearrangeable".

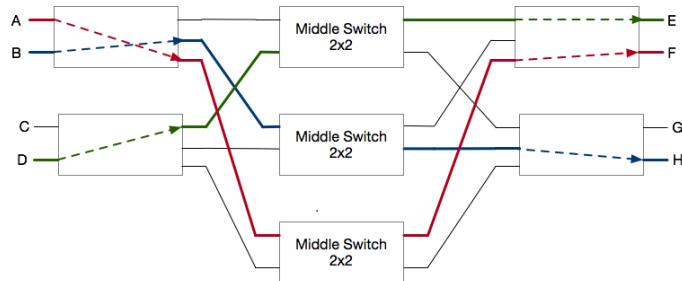


Figura 2.6. Rete di Clos a 3 stadi strettamente non bloccante [11]

Quindi se si inseriscono abbastanza switch nello stadio intermedio, come in Figura 2.6, la rete diventerebbe strettamente non bloccante.

È noto grazie a Clos [3] che per $m \geq 2n - 1$, dove n è il numero di ingressi degli switch del primo stadio e m il numero di switch nello stadio intermedio, la rete è strettamente non bloccante ed è invece non bloccante "rearrangeable" se $m \geq n$.

2.1.4 Conversione di una rete Clos a 3 stadi in 5 stadi

Charles Clos ha inoltre mostrato che, a partire da una rete a 3 stadi, si possono costruire reti con un numero dispari di stadi realizzando lo stadio intermedio con più stadi. Si immagini di avere una rete di Clos a 3 stadi come in Figura 2.7

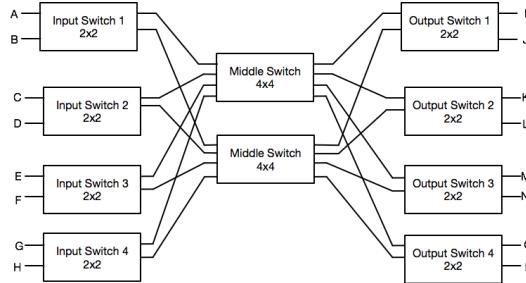


Figura 2.7. Rete di Clos a 3 stadi con $n=2$ e $r=4$ [11]

Si potrebbero rendere gli switch nello stadio intermedio 2×2 sostituendo ogni switch 4×4 con la rete di clos a 3 stadi 2×2 vista in Figura 2.5.

Così si otterebbe una rete di Clos a 5 stadi come in Figura 2.8. In questo caso si ottiene una rete di Benes.

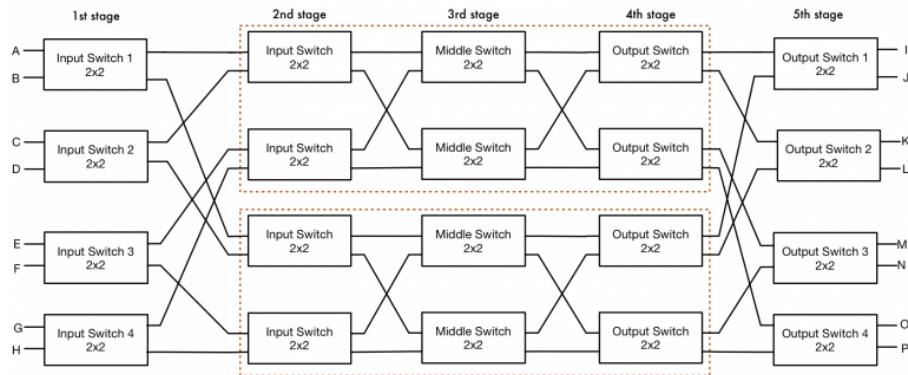


Figura 2.8. Rete di Clos a 5 stadi [11]

2.1.5 Da rete di Clos a Fat-Tree

Si consideri le rete di Clos a 3 stadi mostrata in precedenza e si immagini di piegare la topologia al centro come in Figura 2.9, concatenando gli switch del primo e terzo stadio. Quello che si ottiene è un'altra topologia di rete.

Questa topologia è nota con molti termini, come ad esempio "Folded Clos", tuttavia in questa relazione ci si riferirà ad essa con il nome, ormai comunemente usato, di Fat-Tree.

Lo stesso risultato si può ottenere con una rete di Clos di qualsiasi dimensione, con qualche semplice accorgimento.

Infatti con una rete di Clos a 5 stadi sarebbe sufficiente riorganizzare gli switch nel secondo e quarto stadio e piegare la rete al centro come mostrato in Figura 2.10.

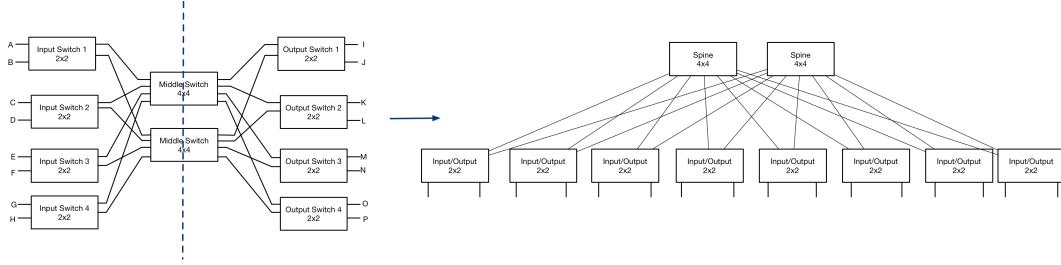


Figura 2.9. Rete di Clos a 3 stadi (sx), rete di Clos ripiegata (dx) [11]

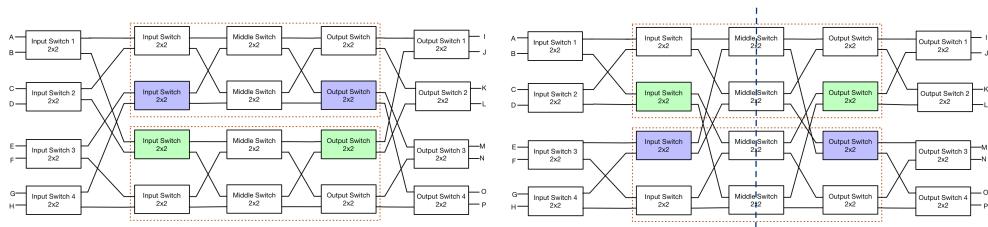


Figura 2.10. Rete di Clos a 5 stadi prima e dopo lo scambio [11]

In questo modo si ottiene il Fat-Tree in Figura 2.11.

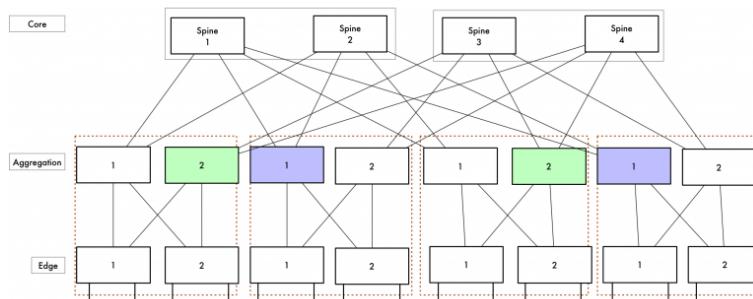


Figura 2.11. Fat-Tree con 3 livelli [11]

2.2 Fat-Tree

I Fat-Tree sono stati originariamente introdotti da Charles Leiserson nel 1985 [6]. L’idea si basa sull’albero binario completo al quale sono stati aggiunti dei collegamenti con l’obiettivo di ridurre la congestione che si crea vicino alla radice.

2.2.1 Alberi

Si immagini di avere un albero elementare binario (o in generale k -ario), come quello mostrato in Figura 2.12.

Questo albero avrà una sola radice collegata ai suoi k discendenti, i quali saranno a loro volta collegati ai loro discendenti, e così fino a raggiungere i nodi foglia.

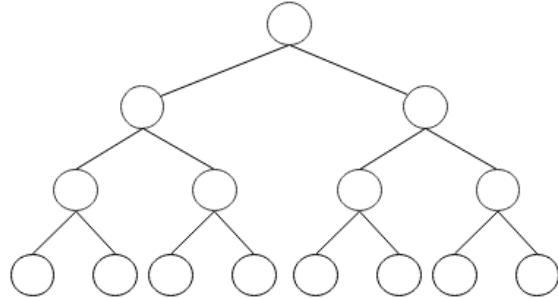


Figura 2.12. Albero binario di altezza 4

Una topologia di questo tipo presenta un difetto fondamentale. All'aumentare del numero di nodi foglia, tutti competono per i collegamenti più vicini al nodo radice. Questo crea il cosiddetto collo di bottiglia.

2.2.2 Fat tree multistage

I Fat-Tree risolvono il problema del collo di bottiglia aggiungendo collegamenti tra i nodi man mano che si sale verso la radice, come si può vedere in Figura 2.13.

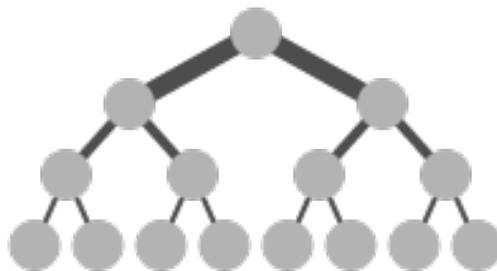


Figura 2.13. Fat-Tree con 4 livelli

Tuttavia questa soluzione aumenta la dimensione degli switch necessari al nodo radice e ai suoi figli per garantire che siano disponibili percorsi sufficienti per tutti i nodi nella rete. Perciò incrementando il numero di nodi cresce la dimensione del nodo radice e dei suoi discendenti.

Questo rende impossibile costruire reti che hanno migliaia di nodi.

Per questo motivo si utilizza maggiormente la topologia Fat-Tree in Figura 2.14, ovvero un Fat-Tree il cui nodo radice e i suoi discendenti vengono replicati usando nodi di dimensione minore e mantenendo invariato il numero complessivo di archi.

I vantaggi nell'utilizzo di topologie Fat-Tree sono molti. Essi "forniscono regolarità, simmetria, scalabilità ricorsiva, massima tolleranza ai guasti, scalabilità logaritmica della bisezione del diametro e consentono semplici algoritmi di self-routing e broadcasting tolleranti ai guasti". [10]

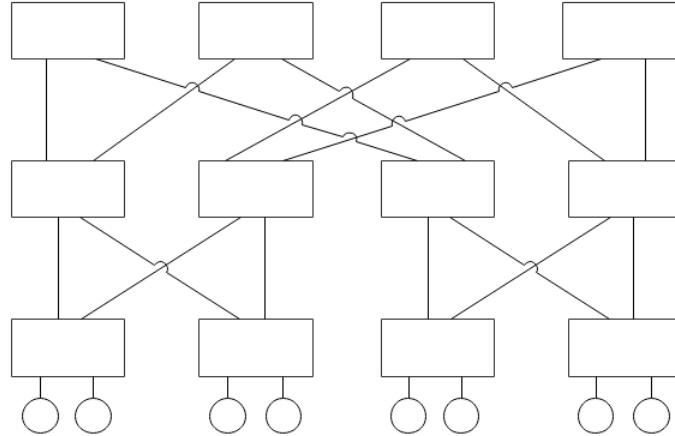


Figura 2.14. Multistage Fat-Tree

2.2.3 Generalized Fat-Tree

La topologia Generalized Fat-Tree (GFT) [9], [10] rappresenta un modello formale unificante per progettare e analizzare un'architettura basata su Fat-Tree [10].

Un Fat-Tree generalizzato si ottiene ricorsivamente con la funzione $GFT(h, m, w)$, dove h è l'altezza dell'albero, m il numero di sottoalberi di altezza $h - 1$ e w^h i nodi al livello top.

Per costruire un $GFT(h + 1, m, w)$ si creano m copie distinte di $GFT(h, m, w)$, indicate con $GFT^j(h, m, w)$ con $0 \leq j \leq m - 1$, e si aggiungono w^{h+1} nodi al livello $h + 1$.

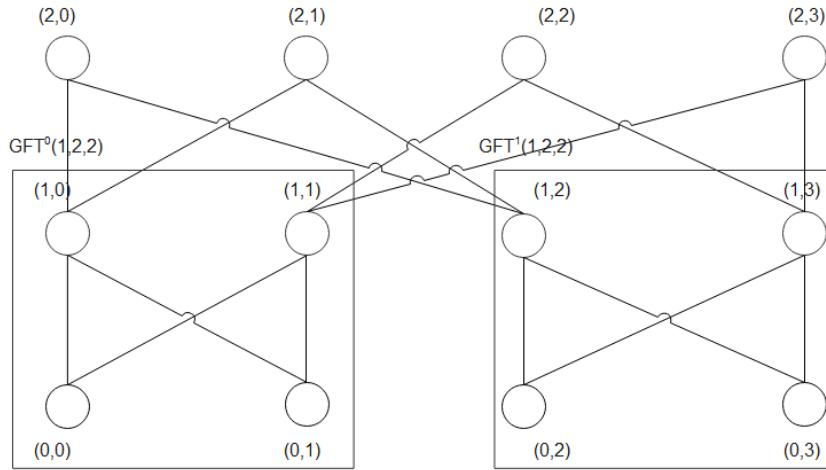
Ogni nodo al livello top, etichettato con $(h + 1, k)$ dove $0 \leq k \leq w^h - 1$, sarà collegato con un nodo al livello top, etichettato con $(h, k + jw^h)$, di ogni $GFT^j(h, m, w)$ con la seguente regola:

$$(h, a) \rightarrow (h + 1, b) \text{ se } a \bmod w^h = \lfloor \frac{b}{w} \rfloor.$$

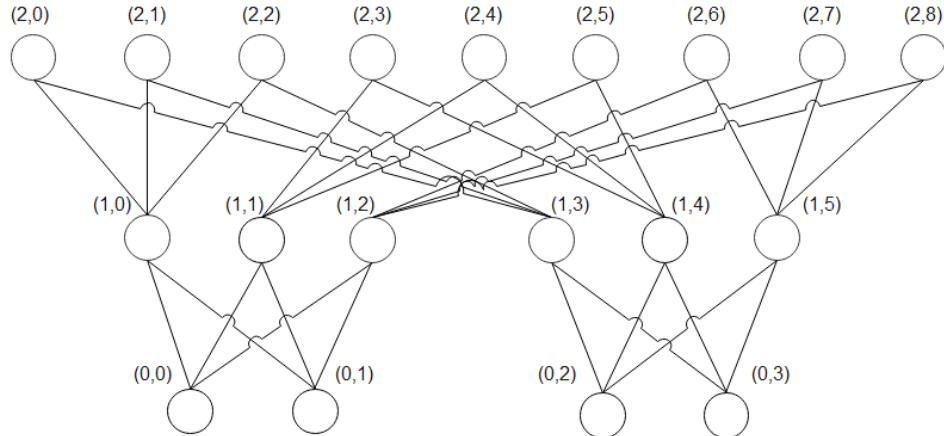
La Figura 2.15 mostra un $GFT(2, 2, 2)$ costruito nel seguente modo:

- Si creano $m = 2$ copie distinte di $GFT(1, 2, 2)$, ovvero $GFT^0(h, m, w)$ e $GFT^1(h, m, w)$, e si aggiungono $w^h = 4$ nodi al livello 2.
- Si etichettano i nodi al livello 2 con $(2, k)$ dove $0 \leq k \leq w^h - 1 = 3$.
- Si etichettano i nodi al livello 1 del $GFT^0(h, m, w)$ con $(1, k)$ e del $GFT^1(h, m, w)$ con $(1, k + 2)$ dove $0 \leq k \leq w^h - 1 = 1$.
- Si collegano i nodi al livello 2 con i nodi al livello 1.

Ad esempio il nodo $(1, 0)$ si collega con il nodo $(2, 0)$ poiché $0 \bmod 2 = 0$ e $\lfloor \frac{0}{2} \rfloor = 0$ e con il nodo $(2, 1)$, infatti $\lfloor \frac{1}{2} \rfloor = 0$

Figura 2.15. $GFT(2, 2, 2)$

2.2.4 Fattened e Slimmed Fat-Tree

Figura 2.16. $GFT(2, 2, 3)$: Fattened Fat-Tree

L’albero in Figura 2.15 rappresenta un Fat-Tree semplice in cui $m = w$. Tuttavia si possono ottenere diverse costruzioni di un Fat-Tree modificando i valori m e w .

Quando $m < w$ si ottiene un Fattened Fat Tree, cioè un Fat Tree "ingrassato".

Infatti come si può notare in Figura 2.16 il livello top appare più grande rispetto al resto dell’albero.

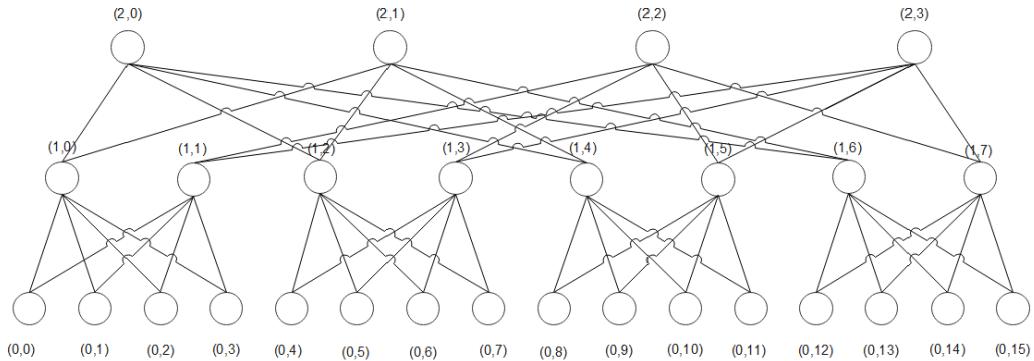


Figura 2.17. $GFT(2, 4, 2)$: Slimmed Fat-Tree

La topologia in Figura 2.17 rappresenta invece un Fat-Tree in cui $m > w$, il che trasforma l'albero in un Fat-Tree "snellito", Slimmed Fat-Tree, poichè diminuisce il numero di nodi usati ad ogni livello.

Capitolo 3

Comunicazione all-to-all

In un sistema di elaborazione distribuito, i processori devono poter comunicare gli uni con gli altri. Considerando il numero di processori coinvolti, possiamo dividere in one-to-one (unicast), one-to-many (multicast), one-to-all (broadcast) e all-to-all [8]. Tra le possibili richieste di comunicazione, uno schema particolarmente impegnativo in termini di prestazioni e scalabilità è rappresentato dalla comunicazione all-to-all. [5].

In base al tipo di messaggio da inviare, le comunicazioni all-to-all possono essere classificate in all-to-all broadcast e all-to-all personalizzate.

3.1 All-to-all broadcast

La comunicazione all-to-all prevede che ogni processore invii un messaggio individuale a ogni altro processore. Si tratta di una generalizzazione del broadcast one-to-all in cui tutti i nodi avviano simultaneamente un broadcast.

Il broadcast all-to-all viene utilizzato nelle operazioni tra matrici, inclusa la moltiplicazione tra matrici e matrice vettore.

A seconda della topologia della rete sono necessari algoritmi all-to-all appropriati.

3.1.1 Ring

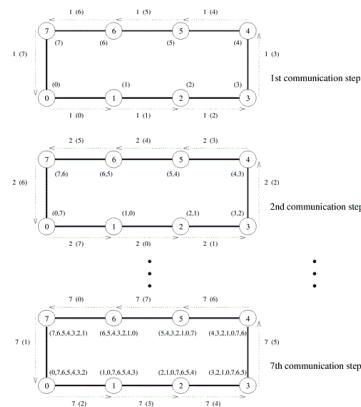


Figura 3.1. Comunicazione all-to-all broadcast su Ring [4]

Per realizzare la comunicazione all-to-all su una rete di tipo ring un nodo invia un messaggio di dimensione $m(n - 1)$, dove n è il numero di nodi e m la dimensione del messaggio, a uno dei suoi vicini. La comunicazione viene eseguita nella stessa direzione su tutti i nodi. Quando un nodo riceve un messaggio, estrae la parte che gli appartiene e inoltra il resto del messaggio al vicino successivo. Dopo $(n - 1)$ round di comunicazione, ogni messaggio viene distribuito alla sua destinazione, come mostrato in Figura 3.1.

3.1.2 Mesh

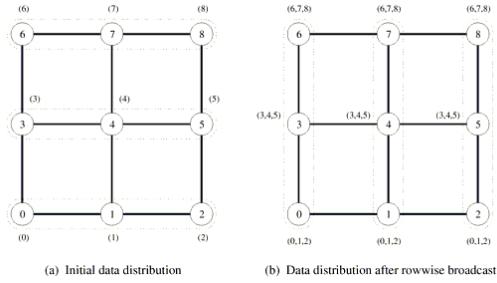


Figura 3.2. Comunicazione all-to-all broadcast su Mesh [4]

L'algoritmo all-to-all in una mesh 2D, descritta nel paragrafo 1.1.3, si basa su quello mostrato in precedenza per la topologia ad anello. Tale algoritmo è costituito da due fasi di comunicazione:

- Nella prima fase, proprio come mostrato nel paragrafo 3.1.1, ogni nodo raccoglie $i \sqrt{n}$ messaggi corrispondenti ai \sqrt{n} nodi delle rispettive righe. Ciascun nodo raccoglie queste informazioni in un unico messaggio di dimensioni $m\sqrt{n}$.
- Nella seconda fase i messaggi raccolti vengono inviati verso le rispettive colonne.

La Figura 3.2 mostra l'algoritmo in una mesh 3x3.

3.1.3 Hypercube

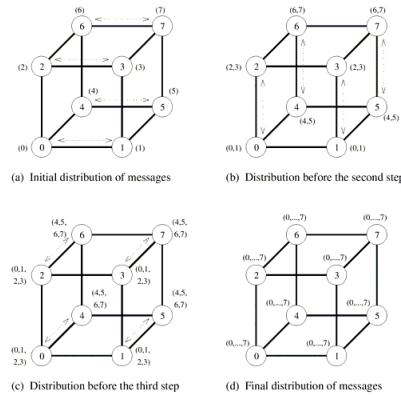


Figura 3.3. Comunicazione all-to-all broadcast su Hypercube [4]

L'algoritmo per il broadcast all-to-all sull'iper cubo descritto nel paragrafo 1.1.5, è un'estensione dell'algoritmo mesh. In ogni fase coppie di nodi vicini scambiano tra loro i dati e concatenano il messaggio ricevuto al proprio, per poi trasmettere il nuovo messaggio nella fase successiva. La procedura richiede $\log n$ passaggi per n nodi. La Figura 3.3 mostra questi passaggi per un ipercubo a 8 nodi con canali di comunicazione bidirezionali.

3.2 All-to-all personalizzata

Nella comunicazione all-to-all personalizzata ogni nodo ha informazioni diverse che deve inviare a tutti gli altri nodi. A differenza della corrispondente comunicazione broadcast, in cui ogni nodo invia lo stesso messaggio a tutti gli altri nodi, nella comunicazione personalizzata all-to-all (nota anche come complete exchange o total exchange) ogni nodo invia messaggi diversi ad ogni nodo.

Il problema della comunicazione personalizzata all-to-all è stato ampiamente studiato negli ultimi anni per molte topologie di rete, come mesh, tori e ipercubi, e più recentemente anche per reti gerarchiche, bus dati e reti di interconnessione multistadio, in breve MIN [5].

3.2.1 Ring

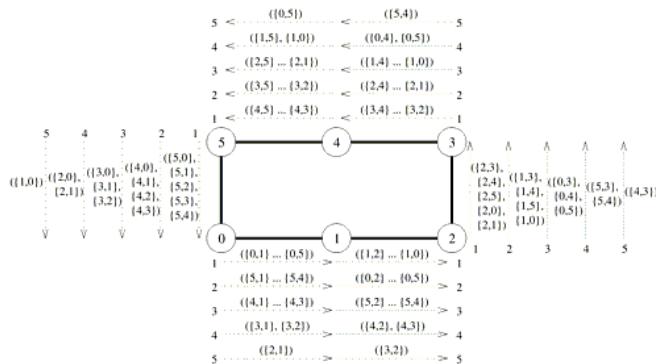


Figura 3.4. Comunicazione all-to-all personalizzata su Ring [4]

L'algoritmo per la comunicazione all-to-all personalizzata su una rete ad anello [4] si basa su quello della comunicazione all-to-all broadcast, cioè ogni nodo ha un messaggio di dimensione $m(n - 1)$ che invia ai suoi vicini. A differenza della comunicazione all-to-all broadcast ogni messaggio è etichettato da una coppia (i, j) dove i è la sorgente e j la destinazione. Quando un nodo riceve un messaggio estrae solo la parte a lui destinata e invia il resto ai suoi vicini. La Figura 3.4 mostra i passaggi di una comunicazione personalizzata all-to-all su un anello a 6 nodi.

3.2.2 Mesh

Nella comunicazione all-to-all personalizzata su una mesh [4], ogni nodo raggruppa gli n messaggi da inviare in base alle colonne dei nodi di destinazione. Ogni nodo ha inizialmente n messaggi di dimensione m , uno per ogni nodo, etichettati con (i, j) dove i è la sorgente e j la destinazione. Ciascun nodo assembla i propri dati in \sqrt{n} gruppi di \sqrt{n} messaggi ciascuno. Ogni gruppo contiene messaggi destinati a una

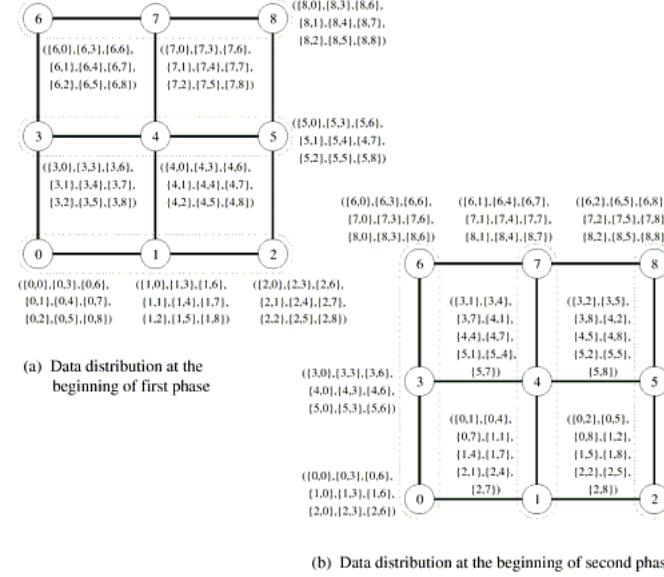


Figura 3.5. Comunicazione all-to-all personalizzata su Mesh [4]

colonna. Dopo che i messaggi sono stati raggruppati, la comunicazione viene eseguita in modo indipendente in ogni riga con messaggi raggruppati di dimensioni $m\sqrt{n}$. Nella seconda fase, dopo che i messaggi di ogni nodo sono stati nuovamente ordinati, questa volta in base alle righe, i nodi inviano i messaggi sulla propria colonna. La Figura 3.5 mostra una mesh 3×3 , in cui ogni nodo ha inizialmente 9 messaggi di dimensione m , e la mesh dopo la prima fase quando i messaggi sono stati inviati su ogni riga.

3.2.3 Hypercube

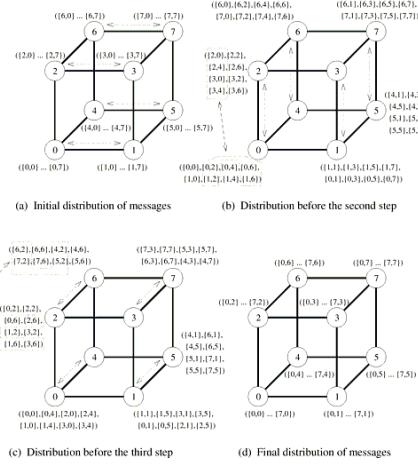


Figura 3.6. Comunicazione all-to-all personalizzata su Hypercube [4]

Un modo per eseguire una comunicazione all-to-all personalizzata su un ipercubo [4] ad n nodi prevede l'estensione dell'algoritmo della mesh. La comunicazione

termina in $\log n$ passi. In qualsiasi fase ogni nodo gestisce n pacchetti da m messaggi ciascuno. Si ricorda che in un ipercubo con n nodi, un insieme di $\frac{n}{2}$ collegamenti nella stessa dimensione collega due sottocubi di $\frac{n}{2}$ nodi ciascuno. Un nodo scambia un insieme di $\frac{n}{2}$ messaggi con un nodo vicino, tali messaggi sono quelli destinati ai nodi dell'altro sottocubo. La Figura 3.6 mostra i passaggi di comunicazione necessari per eseguire questa operazione su un ipercubo tridimensionale.

3.2.4 Reti a stadi

Su reti di interconnessione multistadio la comunicazione all-to-all personalizzata può essere realizzata instradando un insieme di N permutazioni che formano un Quadrato Latino [5].

Un Quadrato Latino può essere visto come una matrice $N \times N$ con N elementi distinti, dove ogni elemento compare una sola volta in ogni riga e ogni colonna, come in Figura 3.7.

A	B	C	D	E
B	C	E	A	D
C	E	D	B	A
D	A	B	E	C
E	D	A	C	B

Figura 3.7. Quadrato Latino [1]

3.2.4.1 Reti con $\log_2 N$ stadi

Le MIN con $\log_2 N$ stadi sono state ampiamente studiate. Purtroppo queste reti non sono "rearrangeable", il che significa che possono realizzare solo un sottoinsieme delle $N!$ possibili permutazioni. Possono comunque realizzare una comunicazione personalizzata all-to-all, se non è richiesta una capacità di permutazione completa.

Yang e Wang hanno sviluppato un metodo per realizzare la comunicazione personalizzata all-to-all [12], ATAPE, con complessità temporale $O(N)$, tuttavia l'algoritmo dipende dalle permutazioni interstadio, cioè dalla topologia delle reti.

In [8] invece viene descritto un metodo che realizza una comunicazione all-to-all personalizzata su una MIN con $\log N$ stadi indipendentemente dalla topologia della rete, attraverso l'operazione XOR.

Viene mostrato che, considerando tutte le permutazioni ammissibili per una MIN, queste si possono partizionare in insiemi P^l dove $l = 0, \dots, N^{(\frac{N}{2}-1)} - 1$, e tali insiemi sono quadrati latini. In particolare, si può definire un partizionamento canonico.

Si consideri una matrice M^l di dimensione $\frac{N}{2} \times \log_2 N$ tale che la sequenza delle sue righe, $r_0, r_1, \dots, r_{\frac{N}{2}-1}$, è la rappresentazione binaria di l , dove $l = 0, \dots, N^{(\frac{N}{2}-1)} - 1$. Tale matrice rappresenta il valore degli switch (0 o 1) della rete ed è associata ad una permutazione ammissibile. Per definizione la prima riga di switch è posta a straight.

Partendo da M^l è possibile ottenere tutte le matrici binarie che producono le permutazioni del quadrato latino associato attraverso $N - 1$ passi (passi XOR)

ognuno dei quali produce una matrice $M^{l,x}$, dove $1 \leq x \leq N - 1$ e $x_{\log N - 1} \dots x_1 x_0$ è la rappresentazione binaria di x .

Il passo XOR x produce la matrice $M^{l,x}$ nel seguente modo: ogni riga i di $M^{l,x}$ cioè $r_i^{l,x}$, si ricava dalla riga i di $M^{l,0}$ cioè da $r_i^{l,0}$, eseguendo lo XOR bit a bit tra $r_i^{l,0}$ e x cioè:

$$r_i^{l,x} = r_i^{l,0} \oplus x_{\log N - 1} \dots x_1 x_0$$

3.2.4.2 Reti con $2 \log_2 N - 1$ stadi

Una doppia MIN è una rete di interconnessione multistadio ottenuta concatenando due MIN con $\log_2 N$ stadi, sovrapponendo l'ultimo stadio della prima MIN con il primo stadio della seconda MIN. Quindi, una doppia MIN ha $2 \log_2 N - 1$ stadi, numerati da 0 a $2 \log_2 N - 2$, e il suo stadio intermedio, dove le due MIN si sovrappongono, è lo stadio $\log_2 N - 1$.

L'articolo [5] illustra come realizzare una comunicazione personalizzata all-to-all in tempo $O(N + \log_2 N) = O(N)$ su una rete Butterfly-Butterfly, mostrata in Figura 3.8, ottenuta concatenando due Butterfly.

In particolare viene mostrato come realizzare il quadrato latino che consiste della permutazione identità e delle sue $N - 1$ rotazioni, denotato Right Latin Square (RLS).

In [5] si mostra come ottenere la permutazione identità e, a partire da essa, tutte le rotazioni di ordine pari e come ottenere la prima rotazione e, a partire da essa, tutte le permutazioni di ordine dispari.

- Permutazione identità

L'idea per instradare la permutazione dell'identità attraverso la Butterfly-Butterfly è utilizzare una permutazione, π , che, applicata due volte, restituisce l'identità: verrà applicata una volta sulla prima Butterfly e una volta sulla seconda.

- Applicazione sulla prima Butterfly

- * Gli ingressi pari della prima metà sono mappati sulle uscite pari della prima metà in ordine crescente, ovvero $0, 2, \dots, N/2 - 2$.
 - * Gli ingressi pari della seconda metà sono mappati sulle uscite dispari della prima metà in ordine inverso, cioè $N - 2, N - 4, \dots, N/2$.
 - * Gli ingressi dispari della prima metà sono mappati sulle uscite pari della seconda metà in ordine inverso, ovvero $N/2 - 1, N/2 - 3, \dots, 1$.
 - * Gli ingressi dispari della seconda metà sono mappati sulle uscite dispari della seconda metà in ordine crescente, ovvero $N/2 + 1, N/2 + 3, \dots, N - 1$.

- Applicazione sulla seconda Butterfly

- * Gli ingressi in posizione pari della prima metà, che dopo la prima applicazione di π sono ancora $0, 2, \dots, N/2 - 2$, mantengono di nuovo la loro posizione.

- * Gli ingressi in posizione pari della seconda metà dopo la prima applicazione sono $N/2 - 1, N/2 - 3, \dots, 1$ e sono mappati sulle uscite dispari della prima metà in ordine inverso, divenendo così $1, 3, \dots, N/2 - 3, N/2 - 1$.
- * Gli ingressi in posizione dispari della prima metà dopo la prima applicazione sono $N - 2, N - 4, \dots, N/2$ e sono mappati sulle uscite pari della seconda metà, in ordine inverso, divenendo così $N/2, \dots, N - 2$.
- * Gli ingressi in posizione dispari della seconda metà, che dopo la prima applicazione di π sono ancora $N/2 + 1, N/2 + 3, \dots, N - 1$, mantengono di nuovo la loro posizione.

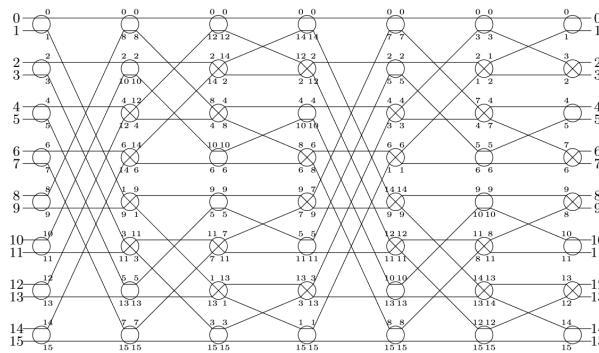


Figura 3.8. Permutazione identità su una Butterfly-Butterfly

- Prima rotazione

L'idea è di modificare in modo opportuno la permutazione dell'identità, secondo i seguenti punti:

- L'impostazione dei nodi del primo stadio rimane su straight, quindi si instradano ancora tutti gli ingressi pari nella prima metà e tutti ingressi dispari nella seconda metà nello stadio intermedio.
- Le configurazioni dei nodi della prima Butterfly vengono modificate per realizzare una permutazione intermedia, indicata come π_{1-int}^r , ottenuta mettendo in relazione le etichette di input di π e quelle di π_{1-int}^r . Quindi, la permutazione intermedia è: $\pi_{1-int}^r(i) = \pi(N/2 - i)$, per $i = 0, \dots, N - 1$.
- Per la seconda Butterfly gli switch nello stadio log N , lo stadio dopo lo stadio intermedio, devono essere modificati in modo tale che quelli a straight diventino cross e viceversa.

In generale la configurazione degli switch della seconda Butterfly è la stessa utilizzata per instradare l'identità per tutte le rotazioni pari, mentre per tutte le rotazioni dispari il primo stadio della seconda Butterfly (stadio log N nella Butterfly-Butterfly) è complementato rispetto all'impostazione degli switch per l'identità.

3.2.4.3 Reti con $\log_d N$ stadi

In [7], si propone un algoritmo di comunicazione personalizzata all-to-all per una d -MIN, cioè per MIN con $\log_d N$ stadi e switch $d \times d$, basato sull'operazione XOR.

L'operazione d -XOR(indicata con \oplus_d) per due cifre d -arie x e $y \in \{0, 1, \dots, d - 1\}$ è definita da $x \oplus_d y = (x + y) \bmod d$.

L'operazione d -XOR è una generalizzazione dell'operazione XOR ordinaria poiché $0 \oplus_2 0 = 0$, $0 \oplus_2 1 = 1$, $1 \oplus_2 0 = 1$, $1 \oplus_2 1 = 0$.

Per realizzare la comunicazione personalizzata all-to-all su una d -MIN, non sono necessari tutti i $d!$ possibili stati di un interruttore $d \times d$, è sufficiente utilizzare i d stati nell'insieme $S = 0\text{-shift}, 1\text{-shift}, \dots, (d - 1)\text{-shift}$, mostrati in Figura 3.9, dove un interruttore $d \times d$ ha lo stato k -shift se i_u è connesso a $o_{(u+k)} \bmod d$ per $u = 0, 1, \dots, d - 1$. Quando $d = 2$, lo stato 0-shift è lo stato straight e lo stato 1-shift è lo stato cross.

Una configurazione di rete di una d -MIN può essere rappresentata da una matrice $M = (m_{i,j})$ di dimensione $\frac{N}{d} \times \log_d N$ dove $m_{i,j}$ rappresenta lo stato dell' i -esimo interruttore dello stadio j e $m_{i,j} = k$ se l' i -esimo interruttore dello stadio j ha lo stato k -shift.

La matrice $M_0 = (m_{i,j}^0)$ si ottiene ponendo $m_{i,j}^0$ uguale a 0 per ogni i, j se $d > 2$, poichè quando $d = 2$, l'insieme S contiene tutti i possibili stati di un interruttore, cioè 2, invece quando $d > 2$, l'insieme S non contiene tutti i possibili stati di un interruttore.

Considerando un intero d -ario $x \in \{1, 2, \dots, N - 1\}$ rappresentato con $x_{(\log_d N)-1} x_{(\log_d N)-2} \dots x_0$, la matrice $M_x = (m_{i,j}^x)$ si ottiene dalla matrice M_0 ponendo

$$m_{i,j}^x = m_{i,j}^0 \oplus_d x_{(\log_d N)-1-j}$$

per ogni i, j .

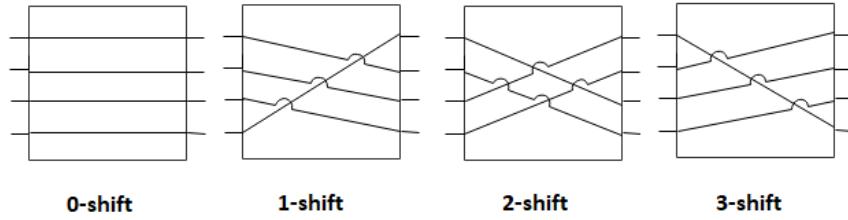


Figura 3.9. Stati d -shift con $d=4$

Capitolo 4

Comunicazione all-to-all personalizzata su Fat tree attraverso LLS

Come detto nel capitolo 3 una comunicazione all-to-all personalizzata può essere richiesta in molte applicazioni dell'elaborazione parallela. In questo lavoro di tirocinio si mostra come sia possibile realizzare tale comunicazione sulle reti Fat-Tree e Slimmed Fat-Tree, descritte nel capitolo 1.

4.1 All-to-all personalizzata su Fat Tree

La comunicazione personalizzata all-to-all su reti Fat Tree può essere realizzata attraverso un insieme di permutazioni che formano un Quadrato Latino, seguendo la linea della all-to-all personalizzata sulla Butterfly-Butterfly mostrata in [5] e descritta nel paragrafo 3.2.4.

In particolare ci si concentrerà sul Left Latin Square, LLS, composto dalla permutazione identità e dalle sue $N - 1$ rotazioni verso sinistra. La Figura 4.1 rappresenta un Left Latin Square con $N = 8$.

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	0
2	3	4	5	6	7	0	1
3	4	5	6	7	0	1	2
4	5	6	7	0	1	2	3
5	6	7	0	1	2	3	4
6	7	0	1	2	3	4	5
7	0	1	2	3	4	5	6

Figura 4.1. Left Latin Square con $N = 8$.

Come mostrato nel paragrafo 2.2.3, un Fat Tree è definito con $GFT(h, m, w)$ dove h rappresenta l'altezza dell'albero, m il numero di $GFT(h - 1, m, w)$ e w^h il numero di nodi da aggiungere al livello h .

Considerando la costruzione del $GFT(h, m, w)$ descritta nel paragrafo 2.2.3 si può notare che la dimensione dei nodi è dipendente dai parametri m e w . Per tale

motivo è necessario prima determinare quali sono gli stati ammissibili dei nodi, tra tutti quelli possibili, per realizzare le permutazioni del LLS.

4.1.1 Configurazioni dei nodi

Dati i parametri del Fat-Tree h, m e w è possibile determinare la dimensione degli switch considerando il livello a cui appartengono.

Quindi i nodi al livello top avranno dimensione $m \times m$, quelli al livello 0 avranno dimensione $w \times w$ mentre gli switch nei livelli intermedi avranno dimensione $m \times w$.

Si ricorda che nei Fat-Tree semplici $m = w$, quindi tutti i nodi della rete avranno la stessa dimensione, m .

Un nodo di dimensione m ha m ingressi, i_k , e m uscite, o_k , dove $0 \leq k < m - 1$, e quindi ha $m!$ possibili stati, di cui soltanto m sono gli stati necessari per realizzare il Left Latin Square, e saranno etichettati da 0 a $m - 1$.

In particolare uno stato i è costruito in modo tale che ogni ingresso, i_k , è collegato all'uscita $o_{(i-k) \bmod m}$, con $0 \leq k < m$. La Figura 4.2 mostra gli stati ammissibili per $m = 2$, $m = 4$ e $m = 8$.

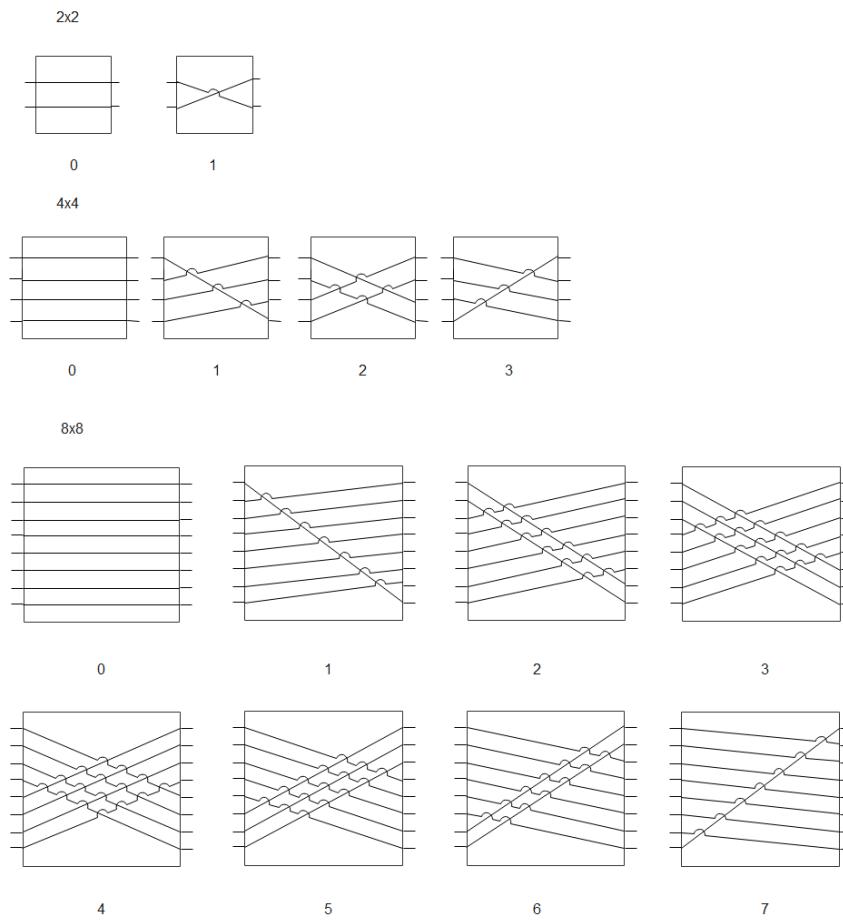


Figura 4.2. Stati dei nodi per $m = 2$, $m = 4$ e $m = 8$

4.1.2 Rotazioni

L'idea per realizzare la comunicazione all-to-all personalizzata su Fat-Tree è quella di partire dalla permutazione identità, realizzata banalmente impostando lo stato di tutti i nodi a straight, e modificare lo stato di un solo nodo alla volta del livello top per generare le rotazioni successive.

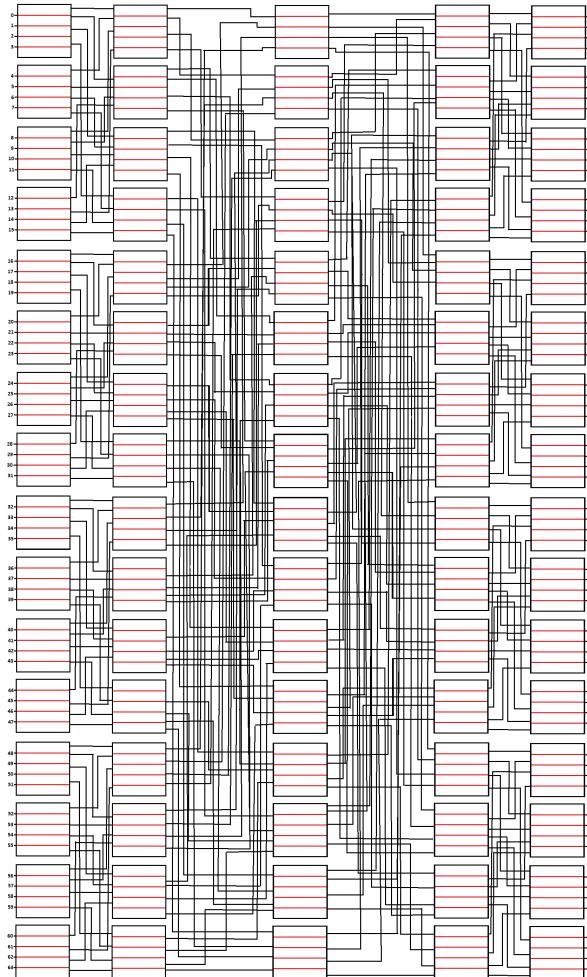


Figura 4.3. $GFT(2,4,4)$ con tutti i nodi a straight

I nodi negli altri livelli invece saranno modificati nella seconda metà del Fat-tree, cioè solo in discesa usando il self routing. La Figura 4.3 mostra un $GFT(2,4,4)$ "aperto", ovvero dove la salita e la discesa sono rappresentate separatamente da sinistra verso destra, che realizza la permutazione identità.

Per realizzare la rotazione 1 nel $GFT(2,4,4)$ si imposta il primo nodo del livello top, il nodo $(2, 0)$, nello stato 1, come descritto nel paragrafo 4.1.1.

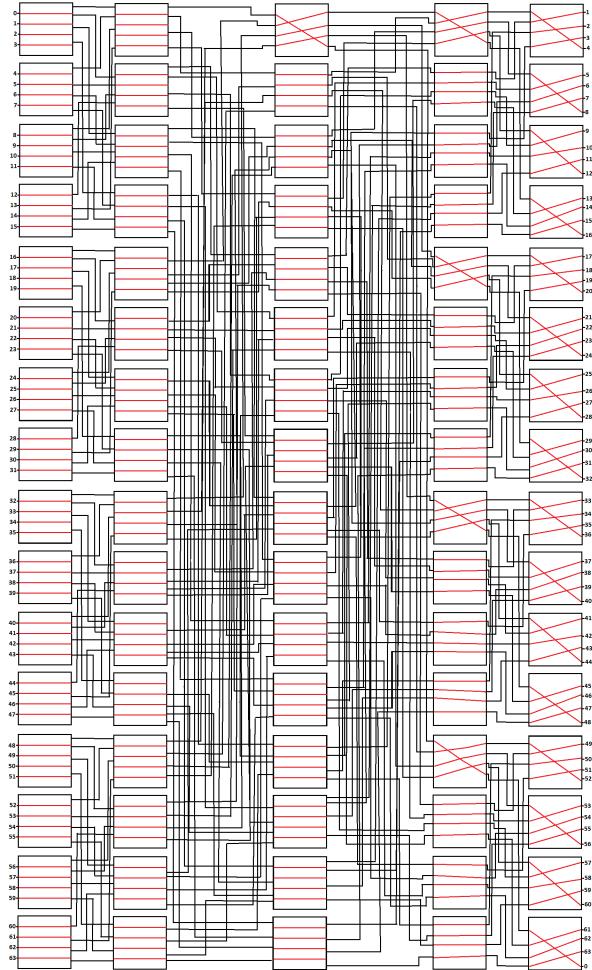


Figura 4.4. $GFT(2, 4, 4)$ che realizza la rotazione 1

Considerando la rotazione 1 rappresentata dagli output in Figura 4.4 e modificando il nodo (2, 1) del livello top si ottiene la rotazione 2, mostrata in Figura 4.5. Una volta che tutti i nodi del livello 2 saranno nello stato 1 si prosegue impostando il primo nodo del livello 2 nello stato 2 e così finché ogni nodo del livello top non si troverà nello stato m .

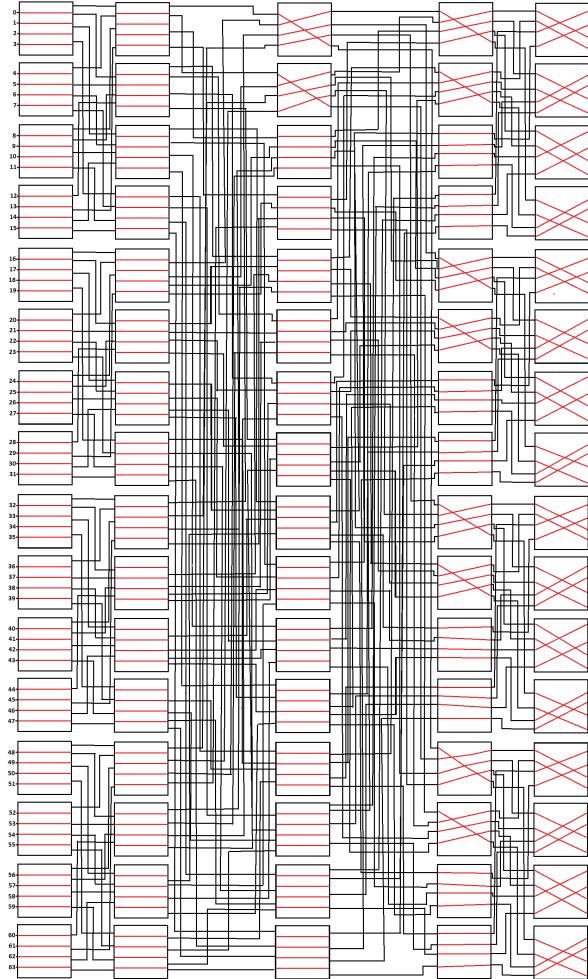


Figura 4.5. $GFT(2, 4, 4)$ che realizza la rotazione 2

4.1.3 Algoritmo di routing

Come già detto, si può ottenere una rotazione modificando lo stato di un nodo del livello top e riorganizzando gli stati degli altri nodi.

In questo paragrafo viene illustrato l'algoritmo di routing che permette di realizzare la rotazione k su un qualsiasi $GFT(h, m, w)$.

In particolare, l'algoritmo per determinare lo stato dei nodi viene applicato ricorsivamente su ogni livello e calcola lo stato dei nodi radice di ogni $GFT(j, m, w)$ dove $0 \leq j \leq h - 1$.

L'algoritmo determina lo stato dei nodi in discesa poiché gli stati dei nodi in salita saranno sempre straight.

Algoritmo 1: Algoritmo che calcola le configurazioni dei nodi di un $GFT(h, m, w)$ per ottenere la rotazione k

Input: h : altezza del Fat-Tree, m : copie di $GFT(h - 1, m, w)$ durante la ricorsione, w : nodi da aggiungere al livello top, k : rotazione, x : altezza soluzione parziale

Result: configurazioni dei nodi di un $GFT(h, m, w)$ alla rotazione k

Variabili globali:

$S[0, \dots, m^h * h] \leftarrow$ dizionario che mappa dalle etichette di tutti i nodi (x, j) alle loro configurazioni;

Function $Routing(h: int, m: int, w: int, k: int, x: int): void$

if $x == h + 1$ **then**

 Print S;

return

end

$c = 0$;

for Every i , s.t. $0 \leq i \leq w^{h-x}$ **do**

for Every j , s.t. $c \leq j \leq k \bmod w^x + c$ **do**

$| S[(x, j)] \leftarrow ((\lfloor \frac{k}{w^x} \rfloor \bmod m) + 1) \bmod m;$

end

$c = j + 1$;

for Every j , s.t. $k \bmod w^x + c \leq j \leq w^x + c$ **do**

$| S[(x, j)] \leftarrow \lfloor \frac{k}{w^x} \rfloor \bmod m;$

end

$c = j + 1$;

end

$Routing(h, m, w, k, x + 1);$

end

La prima chiamata è $Routing(h, m, w, k, 0)$

- Esempio 1.

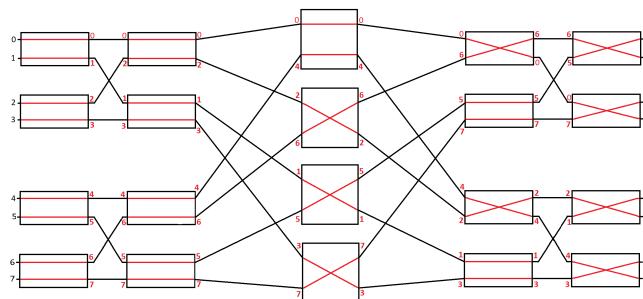


Figura 4.6. Rotazione 5 su $GFT(2, 2, 2)$

Dati $GFT(2, 2, 2)$ e $k = 5$

Si applica l'algoritmo su ogni livello:

– per $h = 2$ si ha:

$$k \bmod w^h = 5 \bmod 4 = 1$$

Quindi il primo nodo del livello 2 è nello stato

$$((\lfloor \frac{k}{w^h} \rfloor \bmod m) + 1) \bmod m = (\lfloor \frac{5}{4} \rfloor \bmod 2 + 1) \bmod 2 = 0.$$

I restanti $w^h - (k \bmod w^h) = 4 - 1 = 3$ nodi sono nello stato

$$\lfloor \frac{k}{w^h} \rfloor \bmod m = \lfloor \frac{5}{4} \rfloor \bmod 2 = 1.$$

- per $h = 1$ si ha:

$$k \bmod w^h = 5 \bmod 2 = 1$$

Quindi il primo nodo di ogni $GFT(1, 2, 2)$ è nello stato

$$((\lfloor \frac{k}{w^h} \rfloor \bmod m) + 1) \bmod m = (\lfloor \frac{5}{2} \rfloor \bmod 2 + 1) \bmod 2 = 1.$$

I restanti $w^h - (k \bmod w^h) = 2 - 1 = 1$ nodi sono nello stato

$$\lfloor \frac{k}{w^h} \rfloor \bmod m = \lfloor \frac{5}{2} \rfloor \bmod 2 = 0.$$

- per $h = 0$ si ha:

$$k \bmod w^h = 5 \bmod 1 = 0$$

Tutti i nodi sono nello stato

$$\lfloor \frac{k}{w^h} \rfloor \bmod m = \lfloor \frac{5}{1} \rfloor \bmod 1 = 1.$$

- Esempio 2.

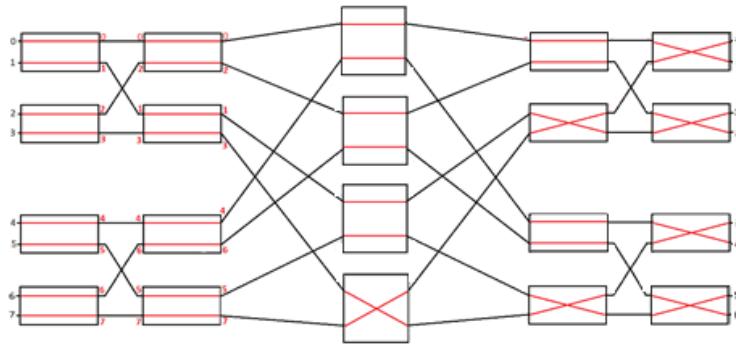


Figura 4.7. Rotazione 7 su $GFT(2, 2, 2)$

Dati $GFT(2, 2, 2)$ e $k = 7$

Si applica l'algoritmo su ogni livello:

– per $h = 2$ si ha:

I primi $k \bmod w^h = 7 \bmod 4 = 3$ nodi del livello 2 sono nello stato

$$((\lfloor \frac{k}{w^h} \rfloor \bmod m) + 1) \bmod m = (\lfloor \frac{7}{4} \rfloor \bmod 2 + 1) \bmod 2 = 0.$$

I restanti $w^h - (k \bmod w^h) = 4 - 3 = 1$ nodi sono nello stato

$$\lfloor \frac{k}{w^h} \rfloor \bmod m = \lfloor \frac{7}{4} \rfloor \bmod 2 = 1.$$

– per $h = 1$ si ha:

$$k \bmod w^h = 7 \bmod 2 = 1$$

Il primo nodo di ogni $GFT(1, 2, 2)$ è nello stato

$$((\lfloor \frac{k}{w^h} \rfloor \bmod m) + 1) \bmod m = (\lfloor \frac{7}{2} \rfloor \bmod 2 + 1) \bmod 2 = 0$$

Il secondo è nello stato 1.

– per $h = 0$ si ha:

$$k \bmod w^h = 7 \bmod 1 = 0$$

Tutti i nodi sono nello stato

$$\lfloor \frac{k}{w^h} \rfloor \bmod m = \lfloor \frac{7}{1} \rfloor \bmod 2 = 1.$$

4.2 All-to-all personalizzata su Slimmed Fat Tree

Usando la stessa idea, ma con le opportune modifiche, è possibile realizzare una comunicazione all-to-all personalizzata anche su Slimmed Fat-Tree, descritti nel paragrafo 2.2.4. Tuttavia poiché questi ultimi hanno la caratteristica che $m > w$, il numero di archi e nodi di queste reti è minore di quello dei Fat-Tree appena visti.

Questo fa sì che siano necessari più passi per instradare tutte le informazioni dai nodi sorgente ai nodi destinazione.

Il numero di passi necessari dipende appunto dai parametri m e w , in particolare è dato da $\lceil \frac{m}{w} \rceil$.

Inoltre, ricordando che la dimensione dei nodi negli stadi intermedi è $m \times w$, e che in questo caso $m > w$, è necessario definire quali sono gli stati di questi nodi necessari per realizzare la comunicazione all-to-all personalizzata.

Per i nodi del livello top e livello 0 invece ci si riferirà ancora alla definizione descritta per i Fat-Tree nel paragrafo 4.1.1.

4.2.1 Configurazioni dei nodi

Un nodo con m ingressi e w uscite dove $m > w$ non può instradare tutti gli input verso gli output, quindi ci saranno stati differenti a seconda delle combinazioni dei w ingressi che vengono instradati.

Quindi, un nodo $m \times w$ ha $\frac{m!}{(m-w)!}$ possibili stati diversi, ma di questi ne saranno sufficienti m , per realizzare ogni rotazione del LLS.

Tuttavia verranno definiti soltanto gli stati dei nodi in discesa, cioè nodi di dimensione $w \times m$, simmetrici rispetto ai nodi $m \times w$

Uno stato i ammissibile di un nodo $w \times m$ è definito nel seguente modo:

- se $0 \leq i \leq w - 1$, si realizza instradando gli ingressi, i_k con $0 \leq k \leq i - 1$, verso le uscite $o_{(m-i)+k}$ e gli ingressi, i_k con $i \leq k \leq w - 1$, nelle uscite o_{k-i} .
- se $w \leq i \leq m - 1$, si costruisce instradando ogni ingresso i_k dove $0 \leq i \leq w - 1$ verso l'uscita $o_k + w - (i \bmod w)$.

Si noti che gli stati i tali che $i \bmod w = 0$ sono a straight. Questi stati sono utilizzati per impostare lo stato dei nodi in salita in ogni passo per poter instradare tutte le informazioni.

La Figura 4.8 mostra gli stati dei nodi 2×4 e 4×8 .

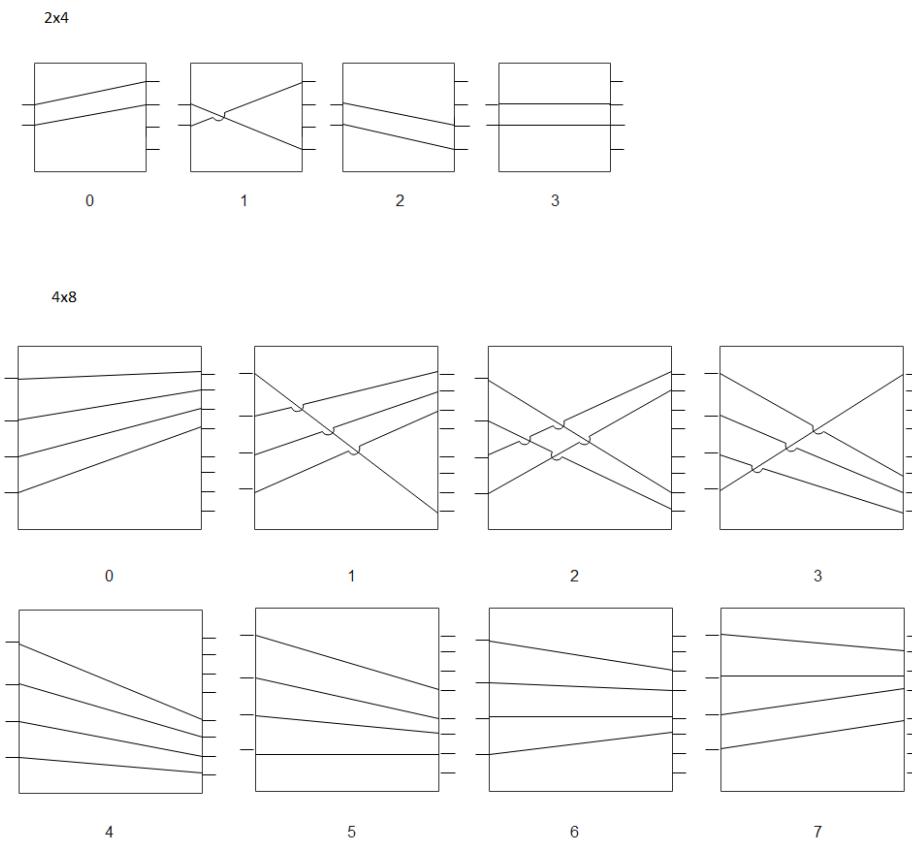


Figura 4.8. Stati dei nodi 2×4 e 4×8

4.2.2 Rotazioni

Esattamente come descritto per i Fat-Tree nella Sezione 4.1.2, la comunicazione all-to-all personalizzata su Slimmed Fat-Tree si realizza considerando la permutazione identità e modificando lo stato di un nodo del livello top per generare ogni rotazione.

Tuttavia si ricorda che per instradare tutte le informazioni sono necessari $\lceil \frac{m}{w} \rceil$ passi, perciò ogni rotazione sarà realizzata instradando $\frac{wm^h}{\lceil \frac{m}{w} \rceil}$ informazioni in ogni passo, dove wm^h rappresenta il numero totale di ingressi nella rete.

In particolare nell' i -esimo passo ogni nodo dello stato intermedio in salita invierà l' i -esimo gruppo di w ingressi nelle rispettive uscite, mentre in discesa gli stati dei nodi, allo stesso modo dei Fat-Tree, saranno modificati usando il self routing.

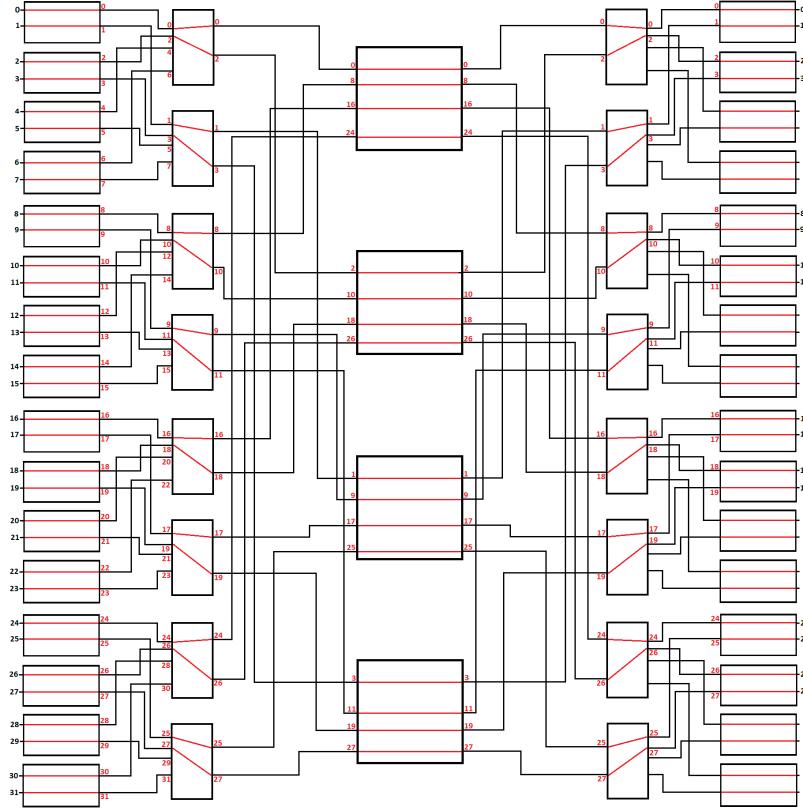


Figura 4.9. $GFT(2, 4, 2)$ che realizza la permutazione identità al passo 1

La Figura 4.9 e la Figura 4.10 mostrano un $GFT(2, 4, 2)$ che realizza la permutazione identità in 2 passi, ovvero ogni nodo dello stato intermedio in salita (di dimensione 4×2) instrada soltanto le prime 2 informazioni al primo passo e le altre 2 nel secondo.

I nodi in discesa, invece, instradano gli ingressi sulle prime due metà durante il primo passo e sulle seconde due metà durante il secondo passo.

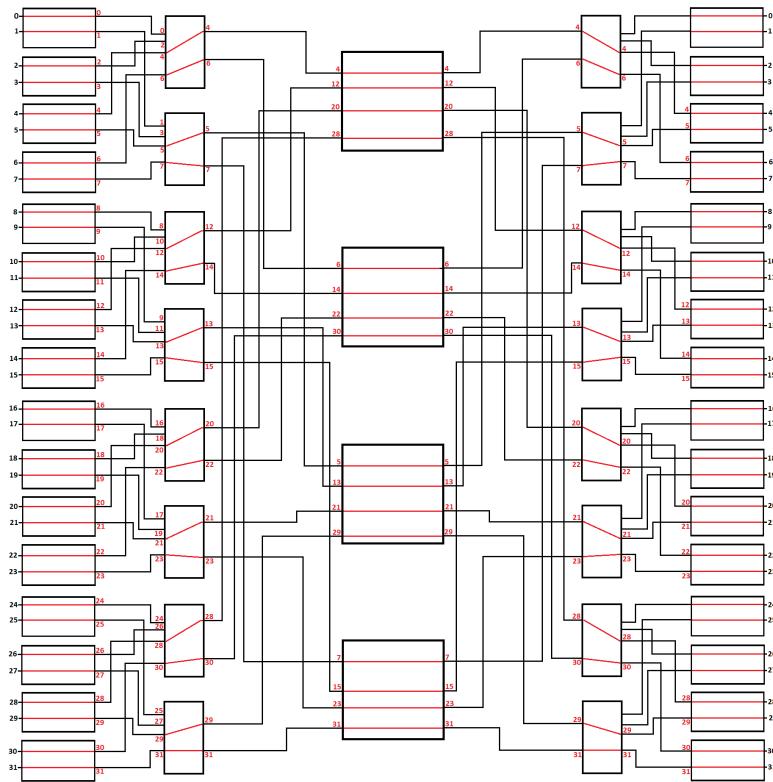


Figura 4.10. $GFT(2, 4, 2)$ che realizza la permutazione identità al passo 2

Si noti che nel primo passo i nodi intermedi, sia in salita che in discesa, sono nello stato 0, mentre nel secondo si trovano nello stato 2.

Per realizzare la rotazione 1 il primo nodo del livello top sarà settato allo stato 1 nel primo passo, lasciando invariati i passi successivi. Quando ogni nodo del livello top avrà cambiato stato, verranno modificati i nodi nel passo successivo.

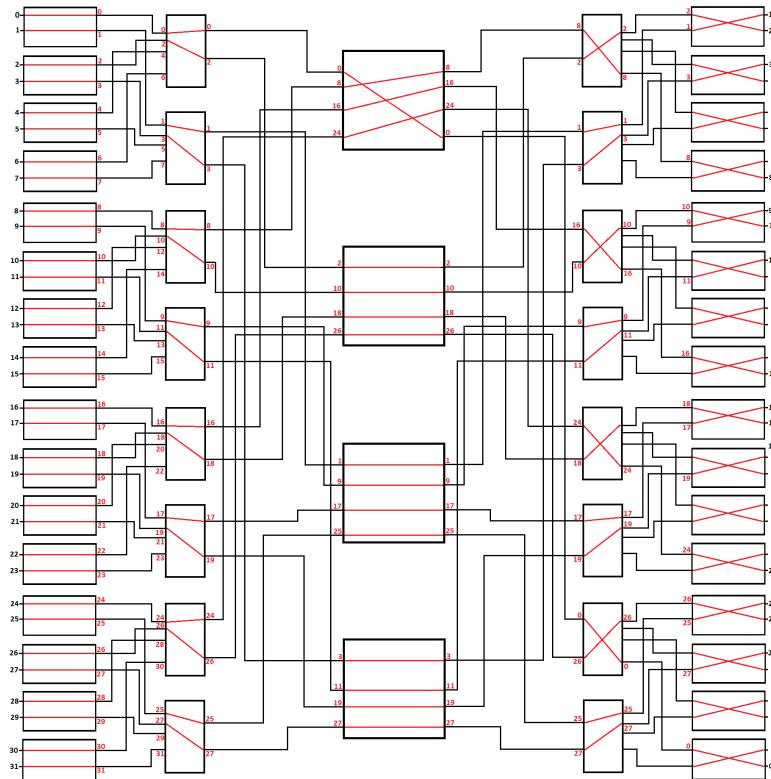


Figura 4.11. $GFT(2, 4, 2)$ che realizza la rotazione 1 al passo 1

La Figura 4.11 e la Figura 4.12 mostrano la realizzazione della rotazione 1 su un $GFT(2, 4, 2)$.

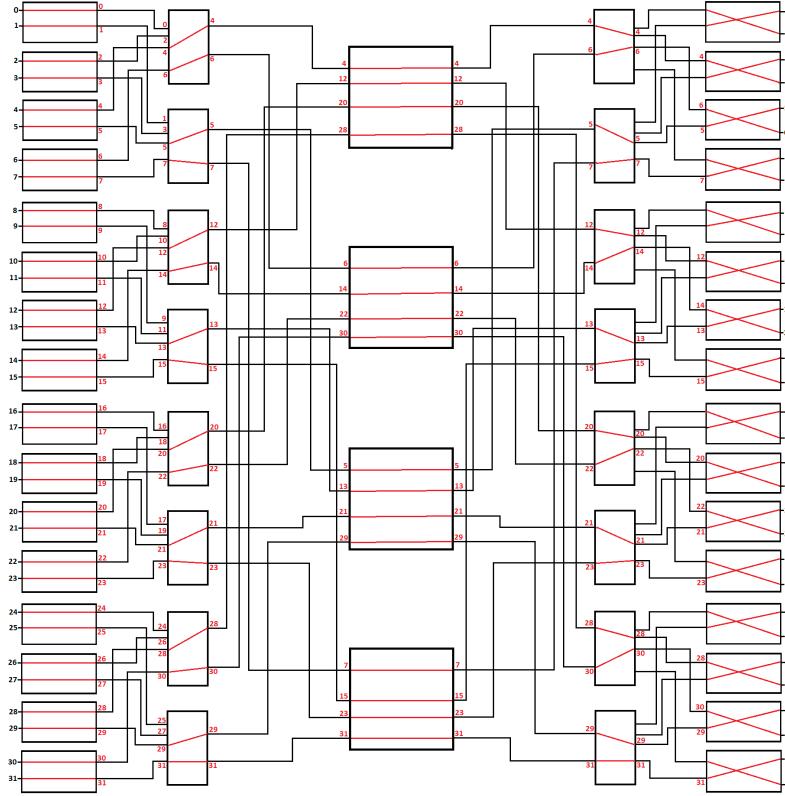


Figura 4.12. $GFT(2, 4, 2)$ che realizza la rotazione 1 al passo 2

4.2.3 Algoritmo di routing

In questo paragrafo si mostra l'algoritmo di routing che realizza la rotazione k al passo p , dove $1 \leq p \leq \lfloor \frac{m}{w} \rfloor$.

Come nel paragrafo 4.1.3 l'algoritmo viene applicato ricorsivamente su ogni livello e calcola lo stato dei nodi radice di ogni $GFT(h, m, w)$.

Tuttavia è necessario distinguere tra i vari livelli, poiché gli stati dei nodi nei livelli intermedi dipendono dal passo p , mentre per il livello top è necessario determinare quali sono i passi che devono essere modificati per realizzare la rotazione k .

L'algoritmo determina lo stato dei nodi lungo la discesa, poiché i nodi in salita sono sempre a straight.

Algoritmo 2: Algoritmo che calcola le configurazioni dei nodi di un $GFT(h, m, w)$ per ottenere la rotazione k al passo p

Input: h : altezza dello Slimmed Fat-Tree, m : copie di $GFT(h - 1, m, w)$ durante la ricorsione, w : nodi da aggiungere al livello top, k : rotazione, x : altezza soluzione parziale, p : passo

Result: configurazioni dei nodi di un $GFT(h, m, w)$ al passo p

Variabili globali:

$S_p[0, \dots, m^{h-x} * w^x] \leftarrow$ dizionario che mappa dalle etichette di tutti i nodi (x, j) con $0 \leq x \leq h$ alle loro configurazioni al passo p ;

Function $Routing(h: int, m: int, w: int, k: int, x: int, p: int): void$

```

if  $x == h + 1$  then
    Print  $S_p$ ;
    return
end
if  $x == h$  then
    if  $1 \leq p \leq \frac{k \bmod \lceil \frac{m}{w} \rceil w^h}{w^x}$  then
        for Every  $j$ , s.t.  $0 \leq j \leq w^x$  do
             $S_p[(x, j)] \leftarrow (((\lfloor \frac{k}{w^x} \rfloor \bmod m) + 1) \bmod m;$ 
        end
    end
    if  $p == \frac{k \bmod \lceil \frac{m}{w} \rceil w^x}{w^x} + 1$  then
        for Every  $j$ , s.t.  $0 \leq j \leq (k \bmod \lceil \frac{m}{w} \rceil w^x) \bmod w^x$  do
             $S_p[(x, j)] \leftarrow (((\lfloor \frac{k}{w^x} \rfloor \bmod m) + 1) \bmod m;$ 
        end
        for Every  $j$ , s.t.  $(k \bmod \lceil \frac{m}{w} \rceil w^x) \bmod w^x \leq j \leq w^x$  do
             $S_p[(x, j)] \leftarrow \lfloor \frac{k}{w^x} \rfloor \bmod m;$ 
        end
    end
    else
        for Every  $j$ , s.t.  $0 \leq j \leq w^x$  do
             $S_p[(x, j)] \leftarrow \lfloor \frac{k}{w^x} \rfloor \bmod m;$ 
        end
    end
end
if  $x == 0$  then
    for Every  $j$ , s.t.  $0 \leq j \leq m^h$  do
         $S_p[(x, j)] \leftarrow k \bmod w;$ 
    end
end
else
     $c = 0;$ 
    for Every  $i$ , s.t.  $0 \leq i \leq w^{h-x}$  do
        for Every  $j$ ,  $c \leq j \leq k \bmod w^x + c$  do
             $S_p[(x, j)] \leftarrow (((\lfloor \frac{k}{w^x} \rfloor \bmod m) + 1) \bmod m + w(p - 1)) \bmod m;$ 
        end
         $c = j + 1;$ 
        for Every  $j$ , s.t.  $k \bmod w^x + c \leq j \leq w^x + c$  do
             $S_p[(x, j)] \leftarrow (\lfloor \frac{k}{w^x} \rfloor \bmod m + w(p - 1)) \bmod m;$ 
        end
         $c = j + 1;$ 
    end
end
     $Routing(h, m, w, k, x + 1, p);$ 
end

```

- Esempio 1.

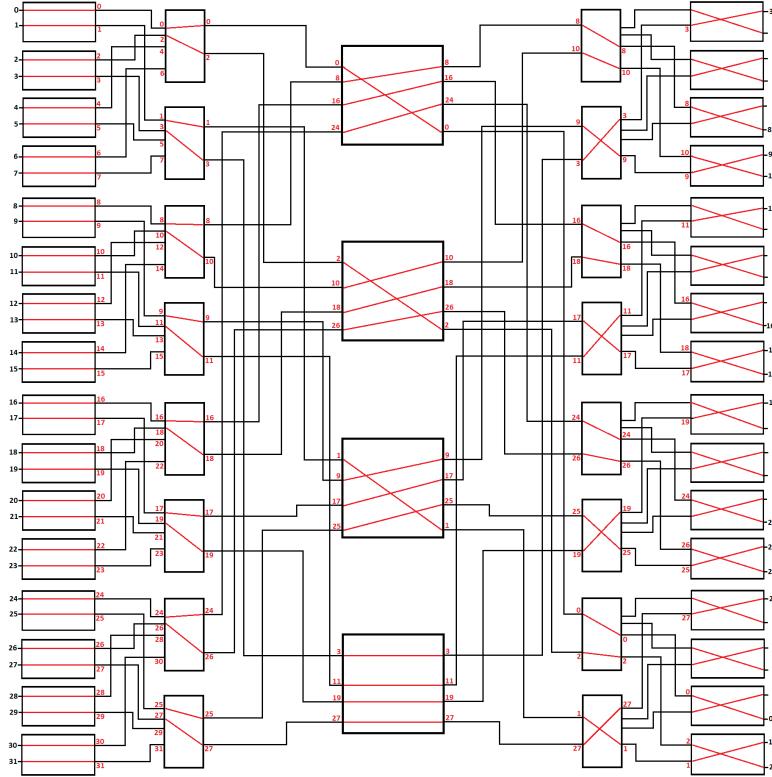


Figura 4.13. Rotazione 3 su $GFT(2, 4, 2)$ al passo 1

Dati $GFT(2, 4, 2)$ e $k = 3$, si hanno $\lceil \frac{m}{w} \rceil = \lceil \frac{8}{4} \rceil = 2$ passi.

– per $h = 2$ si ha:

$$\frac{k \mod \lceil \frac{m}{w} \rceil w^h}{w^h} = \frac{3 \mod \lceil \frac{4}{2} \rceil 4}{4} = \frac{3 \mod 8}{4} = 0$$

$$\begin{aligned} (k \mod \lceil \frac{m}{w} \rceil w^h) \mod w^h &= (3 \mod \lceil \frac{4}{2} \rceil 4) \mod 4 = \\ &= (3 \mod 8) \mod 4 = 3 \end{aligned}$$

I primi 3 nodi del passo 1 sono nello stato

$$((\lfloor \frac{k}{\lceil \frac{m}{w} \rceil w^h} \rfloor \mod m) + 1) \mod m = (\frac{3}{8} \mod 4) + 1 = 1.$$

Mentre tutti i nodi di tutti gli altri passi sono nello stato 0.

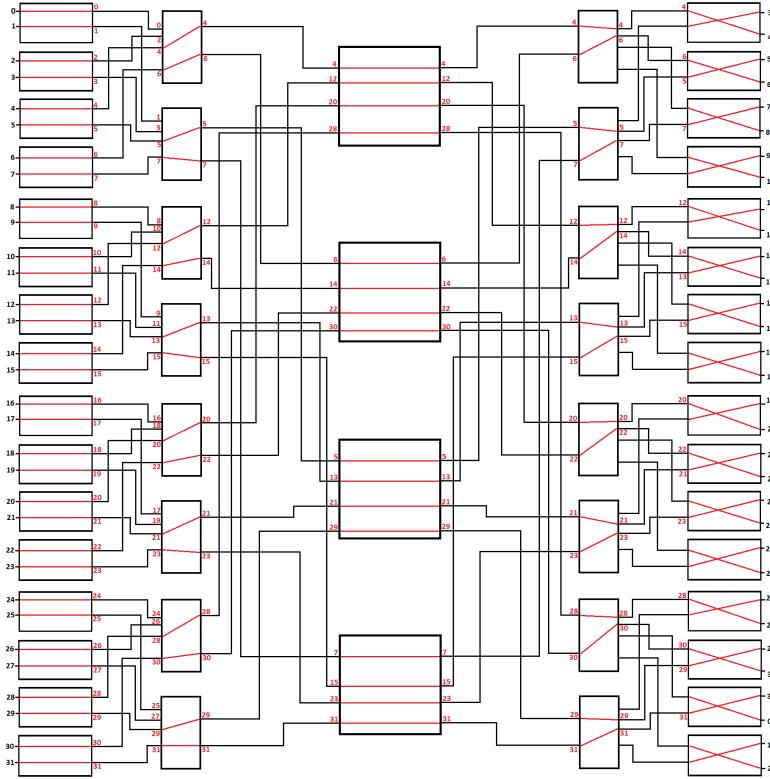


Figura 4.14. Rotazione 3 su $GFT(2, 4, 2)$ al passo 2

– per $h = 1$ si ha:

* Passo 1

$$k \mod w^h = 3 \mod 2 = 1$$

Il primo nodo di ogni $GFT(1, 4, 2)$ è nello stato

$$\begin{aligned} & \left(\left\lfloor \frac{k}{w^h} \right\rfloor \mod m + 1 \right) \mod m + w(i-1) \mod m = \\ & = \left(\left\lfloor \frac{3}{2} \right\rfloor \mod 4 + 1 \right) \mod 4 = 2 \end{aligned}$$

I restanti

$$w^h - (k \mod w^h) = 2 - 1 = 1$$

nodi sono nello stato

$$\left(\left\lfloor \frac{k}{w^h} \right\rfloor \mod m + w(i-1) \right) \mod m = \left\lfloor \frac{3}{2} \right\rfloor \mod 4 = 1.$$

* Passo 2

Il primo nodo di ogni $GFT(1, 4, 2)$ è nello stato

$$(((\lfloor \frac{k}{w^h} \rfloor \mod m) + 1) \mod m + w(i-1)) \mod m =$$

$$\lfloor \frac{3}{2} \rfloor \mod 4 + 1 \mod 4 + 1 = 3$$

I restanti

$$w^h - (k \mod w^h) = 2 - 1 = 1$$

nodi sono nello stato

$$(\lfloor \frac{k}{w^h} \rfloor \mod m + w(i-1)) \mod m = \lfloor \frac{3}{2} \rfloor \mod 4 + 1 = 3.$$

– per $h = 0$ si hanno tutti i nodi nello stato

$$k \mod w = 3 \mod 2 = 1$$

Capitolo 5

Comunicazione all-to-all personalizzata su Fat-Tree attraverso CLS

In [8] e in [7] si dimostra che si può ottenere un comunicazione all-to-all personalizzata su reti di interconnessione multistadio attraverso l'operazione di XOR, applicata alla configurazione dei nodi della rete, sia nel caso di switch 2×2 e $\log_2 N$ stadi [8] che nel caso di switch $d \times d$ e $\log_d N$ stadi [7].

L'obiettivo di questo capitolo è quello di mostrare che l'operazione di XOR può essere utilizzata per realizzare la comunicazione all-to-all personalizzata anche su reti Fat-Tree e Slimmed Fat-Tree. In particolare sarà illustrato come tale operazione permette di instradare le permutazioni che formano il quadrato latino canonico, e come utilizzando lo stesso metodo si possono ottenere altri quadrati latini. In realtà si dimostrerà che tutte le permutazioni ottenibili, senza modificare le configurazioni dei nodi in salita, su un $GFT(h, m, w)$ possono essere raggruppate in quadrati latini.

5.1 All-to-all personalizzata su Fat-Tree

Si ricorda che per realizzare una comunicazione all-to-all personalizzata su un $GFT(h, m, w)$ è necessario instradare wm^h (numero di ingressi del Fat-Tree) permutazioni che formano un quadrato latino.

In questo caso ci si concentrerà sul Canonical Latin Square, CLS, cioè un quadrato latino che ha la caratteristica di avere la prima riga e la prima colonna in ordine crescente e l'ultima riga e l'ultima colonna in ordine decrescente. La Figura 5.1 mostra un quadrato latino canonico con $N=8$.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\ 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

Figura 5.1. Canonical Latin Square N=8

Tuttavia per realizzare tale quadrato latino si devono considerare configurazioni dei nodi diverse rispetto a quelle viste nel paragrafo 4.2.1.

5.1.1 Configurazioni dei nodi

Come già affermato, in un Fat-Tree i nodi hanno dimensione $m \times m$ nel livello top, $m \times w$ negli stadi intermedi e $w \times w$ al livello 0. Si ricorda però che in un Fat-Tree semplice $m = w$, quindi i nodi avranno tutti dimensione $m \times m$.

Considerando i nodi $m \times m$, che hanno cioè m ingressi e m uscite, verranno definiti gli m stati ammissibili per realizzare il quadrato latino canonico tra tutti gli $m!$ stati possibili.

Uno stato i è definito in modo tale che ogni ingresso, i_k con $0 \leq k \leq m - 1$, è collegato con l'uscita, $o_{(k \oplus i)}$.

La Figura 5.2 mostra gli stati per $m = 2$, $m = 4$ e $m = 8$.

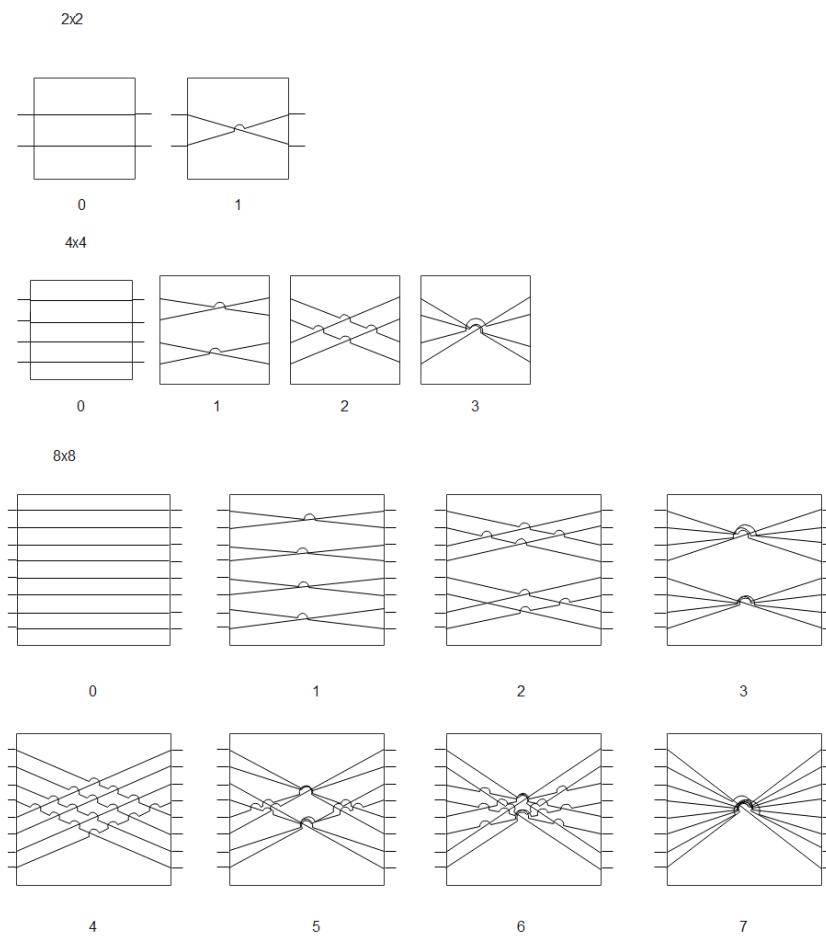


Figura 5.2. Configurazioni dei nodi di dimensione $m = 2$, $m = 4$ e $m = 8$

5.1.2 Canonical Latin Square

Si consideri una matrice binaria M che rappresenti la configurazione del $GFT(h, m, w)$ "aperto", cioè dove la discesa e la salita sono rappresentate separatamente, che avrà quindi $2h + 1$ livelli. In questa matrice in ogni riga il j -esimo gruppo di $\log_2 m$ elementi, con $0 \leq j \leq 2h$, indica lo stato del j -esimo nodo ottenuto usando la rappresentazione binaria.

La dimensione di tale matrice sarà quindi $m^h \times (2h + 1) \log_2 m$.

Si indicherà con M^i la matrice in cui ogni riga è la rappresentazione binaria di i di lunghezza $(2h + 1) \log_2 m$ e con C_i la configurazione della rete rappresentata dalla matrice binaria M^i . La matrice M^0 sarà quindi una matrice composta da tutti 0 che rappresenta la configurazione delle reti C^0 in cui tutti i nodi sono nello stato straight.

Poichè lo stato dei nodi in salita non deve essere modificato, le prime $h \log_2 m$ colonne a partire da destra rimarranno sempre a 0, il che significa che saranno generate tante matrici quante sono le cifre che si possono rappresentare con $(h + 1) \log_2 m = \log_2 m^{h+1}$ bit, ovvero $2^{\log_2 m^{h+1}} = m^{h+1}$ matrici.

Perciò per formare il quadrato latino canonico si costruiranno le matrici M^i con $0 \leq i \leq m^{h+1} - 1$, partendo dalla matrice M^0 attraverso m^{h+1} passi XOR.

Dato l'intero $i \in \{0, 1, \dots, m^{h+1}-1\}$ si consideri la rappresentazione binaria di i di lunghezza $i_{(2h+1)\log_2 m-1} \dots i_1 i_0$.

È possibile generare ogni riga, r_k^i con $0 \leq k \leq m^h$, della matrice M^i nel seguente modo:

$$r_k^i = r_k^0 \oplus i_{(2h+1)\log_2 m-1} \dots i_1 i_0$$

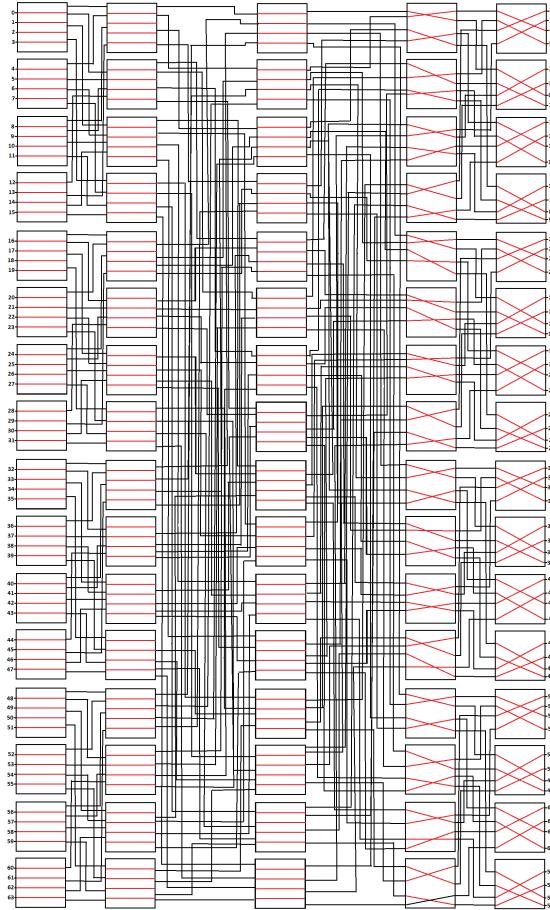


Figura 5.3. Configurazione C^6 di un $GFT(2,4,4)$

Le Figure 5.3 e 5.4 mostrano le configurazioni C^6 e C^7 di un $GFT(2,4,4)$.

Infatti ogni riga della rete relativa a C^6 si ottiene dallo XOR tra la generica riga di M^0 che è 0000000000 e la generica riga di $i = 6$ in base 2 con 10 cifre 0000000110. Lo XOR è eseguito cifra per cifra usando la rappresentazione binaria delle cifre. Lo stesso vale per $i = 7$ la cui rappresentazione in base 2 è 0000000111.

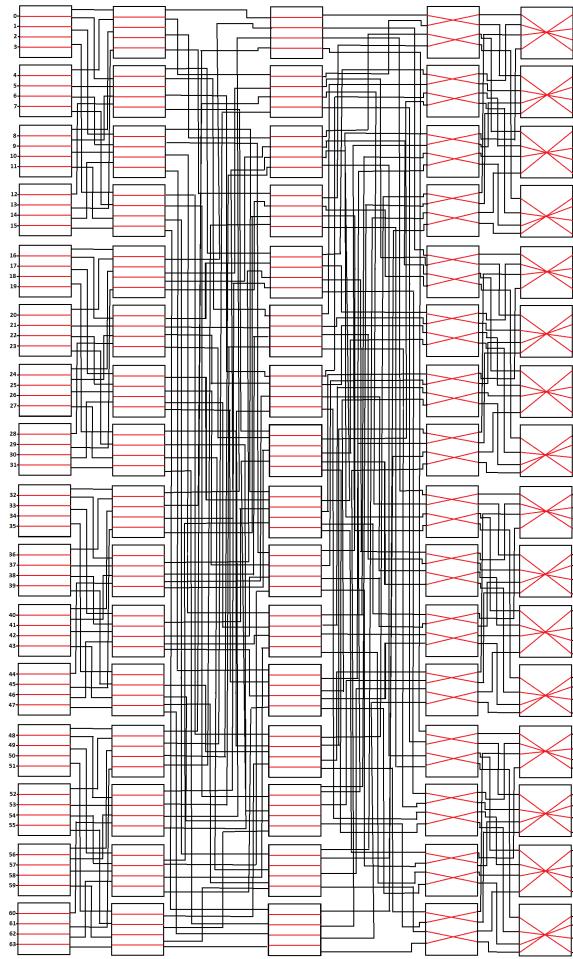


Figura 5.4. Configurazione C^7 di un $GFT(2, 4, 4)$

5.1.3 Partizione delle permutazioni in quadrati latini

Si rappresenti la configurazione di un $GFT(h, m, w)$ attraverso una matrice binaria M di dimensioni $m^h \times (2h + 1) \log_2 m$ come detto nel paragrafo 5.1.2. Tale matrice può avere $2^{m^h(2h+1)\log_2 m}$ configurazioni diverse tuttavia non si vogliono modificare gli stati dei nodi in salita perciò possono essere generate soltanto $2^{m^h(h+1)\log_2 m} = m^{(h+1)m^h}$ configurazioni che instradano $m^{(h+1)m^h}$ permutazioni.

Tutte le permutazioni possono essere quindi raggruppate in $\frac{m^{(h+1)m^h}}{m^{h+1}} = m^{(h+1)m^h - (h+1)}$ quadrati latini. Sarà illustrato uno dei possibili modi di partizionare tutte le permutazioni in quadrati latini.

Nel paragrafo 5.1.2 viene mostrato come realizzare il quadrato latino canonico in passi XOR a partire da una matrice in cui tutti i nodi sono nello stato straight. Allo stesso modo si vedrà come a partire da determinate matrici si costruisce un insieme di quadrati latini, di cui fa parte lo stesso quadrato latino canonico.

Si consideri $M^{(0,i)}$ la matrice binaria in cui la sequenza delle colonne è la rappresentazione binaria di i con $0 \leq i \leq m^{(h+1)m^h - (h+1)}$.

In questo modo le prime $h \log_2 m$ colonne che rappresentano i nodi in salita saranno sempre a 0.

A partire dalla matrice $M^{(0,i)}$ saranno generate tutte le matrici $M^{(x,i)}$ con $x \in \{0, \dots, m^{h+1} - 1\}$, attraverso m^{h+1} passi XOR.

Data $x_{(2h+1)\log_2 m-1} \dots x_1 x_0$ la rappresentazione binaria di x (dove i primi $h \log_2 m$ bit sono 0) si può ottenere ogni riga, $r_k^{(x,i)}$ con $0 \leq k \leq m^h$, della matrice $M^{(x,i)}$ nel seguente modo:

$$r_k^{(x,i)} = r_k^{(0,i)} \oplus x_{(2h+1)\log_2 m-1} \dots x_1 x_0$$

- Esempio

Dato un $GFT(2, 2, 2)$ la cui configurazione è rappresentata dalla matrice $M^{(0,6)}$ si ottengono tutte le matrici $M^{(x,6)}$ con $0 \leq x \leq 7$ come mostrato in Figura 5.5.

01325467	00000 00001 00001 00000
10234576	00001 00000 00000 00001
23107645	00010 00011 00011 00010
32016754	00011 00010 00010 00011
45761023	00100 00101 00101 00100
54670132	00101 00100 00100 00101
67543201	00110 00111 00111 00110
76452310	00111 00110 00110 00111

Figura 5.5. Generazione di un quadrato latino su un $GFT(2, 2, 2)$ da $M^{(0,2)}$

```

import Fat_Tree_utility as ftu
import math

# GFT(h, m, w)
# h - altezza
# m - copie di GFT(h-1,m,w) durante la ricorsione
# w - nodi da aggiungere al livello h
# l -
# M0k - matrice la cui sequenza di colonne è rappresentazione binaria di k = 0,...,m^((h+1)m^h-(h+1))

def xor(h, m, w, l, M0k):
    # m ingressi per ogni nodo livello h
    # w*h numero nodi livello h
    # M = [m * w ** h][h * 3] matrice che rappresenta le permutazioni intermedie in tutti gli stadi
    M = []
    # Mlk - generata da M0k con w*m**h passi XOR con la rappresentazione binaria di l
    Mlk = []

    #inizializzazione matrici
    for i in range(m * w ** h):
        M.append([i-1])
    for i in range(m*w):
        Mlk.append([])

    #settiamo la colonna 0 della matrice con gli ingressi della rete al livello top
    posElem = 0
    for j in range(w ** h):
        for i in range(m):
            M[posElem][0] = j + (m * w) * i
            posElem = posElem + 1

    #settiamo la matrice generata da M0k con w*m**h passi XOR con la rappresentazione binaria di l
    lbin = ftu.de2bi(l, (h + 1) * int(math.log(m, 2)))
    for i in range(m ** h):
        for j in range((h+1)*int(math.log(m,2))):
            Mlk[i].append(M0k[i][j] ^ int(lbin[j]))

    # settiamo le altre colonne della matrice dividendo tra le colonne che rappresentano
    # l'instradamento all'interno di un nodo e quelle che rappresentano l'instradamento
    # da un livello a un altro della rete
    livello = 0
    for colonna in range(0, (h+1)*2-1):
        nodo = 0
        #colonne che rappresentano l'instradamento all'interno di un nodo
        if (colonna % 2 == 0):
            #leggiamo ogni riga della matrice Mlk
            for i in range(m ** h):
                #generiamo la stringa di lunghezza log m confnodo che rappresenta la configurazione del nodo nodo
                confnodo = ""
                for j in range(livello, livello+int(math.log(m,2))):
                    confnodo = confnodo + ftu.de2bi(Mlk[i][j],1)
                #utilizziamo lo xor per determinare come gli ingressi vengono instradati verso le uscite del nodo
                #poichè le configurazioni ammissibili dei nodi si ottengono con l'operazione xor
                for posElem in range(nodo, nodo + m):
                    nuovaPos = posElem ^ int(confnodo,2)
                    M[posElem].append(M[nuovaPos][colonna])
                nodo = nodo + m
        livello = livello + int(math.log(m,2))

        #colonne che rappresentano l'instradamento da un livello a un altro della rete
    else:
        posElem = 0
        for numSubFatTree in range(w**h):
            nuovaPos = numSubFatTree
            for j in range(posElem, posElem + m):
                M[j].append(M[nuovaPos][colonna])
            nuovaPos = nuovaPos + m*w
            posElem = j + 1

    #restituiamo la permutazione finale
    permut = []
    for i in range(m * w ** h):
        permut.append(M[i][2*(h+1)-1])
    return permut

```

Figura 5.6. Funzione $xor(h, m, w, l, M0k)$ che restituisce la permutazione generata dalla matrice $M^{(l,k)}$ su un $GFT(h, m, w)$

La Figura 5.6 mostra la funzione $xor(h, m, w, l, M0k)$ che restituisce la permutazione generata dalla matrice $M^{(l,k)}$, che rappresenta la configurazione $C^{(l,k)}$ di un $GFT(h, m, w)$, costruita a partire dalla matrice $M^{(0,k)}$ e l'operazione di XOR con l .

5.2 All-to-all personalizzata su Slimmed Fat-Tree

È possibile utilizzare lo stesso metodo visto per i Fat-Tree, per instradare le permutazioni che formano il quadrato latino canonico e di conseguenza gli altri quadrati latini, anche in uno Slimmed Fat-Tree.

Ovvero si dimostra che tutte le permutazioni ottenibili, senza modificare le configurazioni dei nodi in salita, su uno Slimmed Fat-Tree possono essere partizionate in quadrati latini.

Si ricorda nuovamente che sono necessari più passi per portare a termine la comunicazione all-to-all, in particolare in un $GFT(h, m, w)$ con $m > w$ servono $\lceil \frac{m}{w} \rceil$ passi.

Anche questa volta, prima di proseguire, è necessario considerare i nodi di dimensione $w \times m$ dove $m > w$ e le loro configurazioni ammissibili.

5.2.1 Configurazione dei nodi

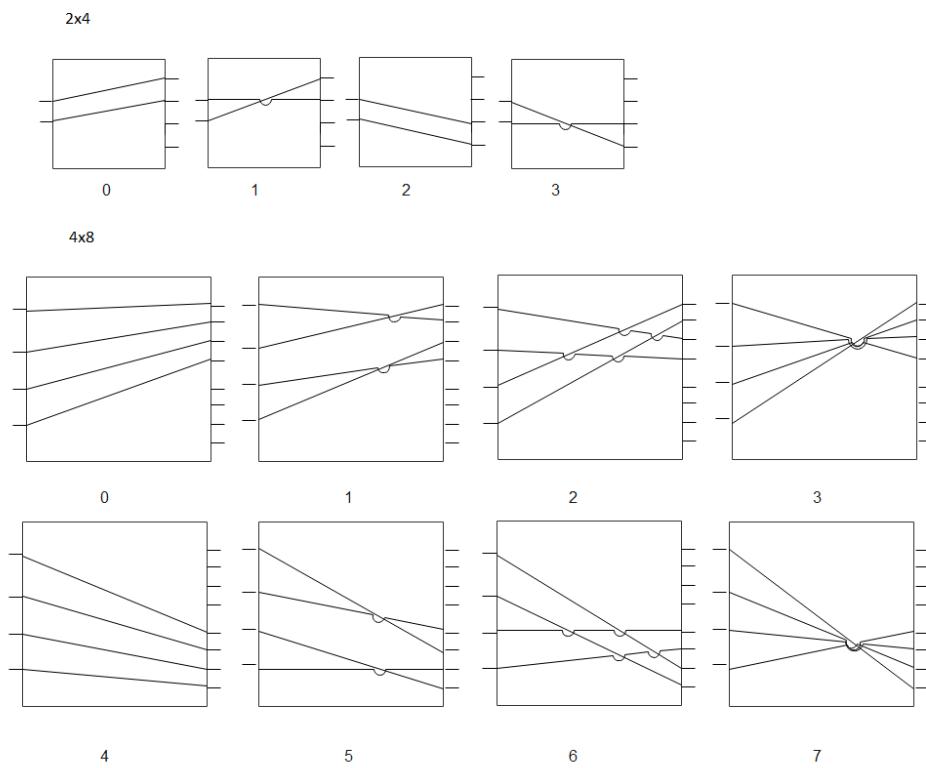


Figura 5.7. Configurazioni dei nodi di dimensione 2×4 e 4×8

Uno stato i ammissibile è costruito in modo tale che ogni ingresso, i_k con $0 \leq k \leq w - 1$, è collegato con l'uscita, $o_{(k \oplus i)}$.

Si noti che gli stati i tali che $i \bmod w = 0$ sono a straight. Questi sono necessari per impostare lo stato dei nodi in salita in ogni passo per instradare tutte le informazioni.

La Figura 5.7 mostra gli stati dei nodi 2×4 e 4×8 .

5.2.2 Canonical Latin Square

Nel paragrafo 5.1.2 è stato discusso un metodo per realizzare una comunicazione all-to-all personalizzata attraverso un Canonical Latin Square su un Fat-Tree sfruttando una matrice binaria M che rappresenta la configurazione di rete.

Per rappresentare la configurazione di rete di uno Slimmed Fat-Tree è però necessario utilizzare tante matrici quante sono i passi necessari per permettere lo scambio di tutti i dati.

Inoltre si ricorda che i nodi hanno dimensione $m \times m$ nel livello top, $m \times w$ negli stadi intermedi e $w \times w$ al livello 0, questo significa che per rappresentare lo stato dei nodi al livello top e nei livelli intermedi sono necessari $\log_2 m$ bit mentre per rappresentare lo stato dei nodi al livello 0 sono richiesti $\log_2 w$ bit.

Poichè la matrice M rappresenta la configurazione di un Fat-Tree aperto, avremo due livelli 0, quello in salita e quello in discesa.

Quindi è necessario costruire $\lceil \frac{m}{w} \rceil$ matrici di dimensione

$$m^h \times (2h - 1) \log_2 m + 2 \log_2 w.$$

Tuttavia il numero di nodi in ogni livello in uno Slimmed Fat-Tree diminuisce salendo verso il livello top. Per questo motivo non tutti i bit di ogni colonna dovranno essere considerati. Si sceglie di utilizzare i primi $w^x m^{h-x}$ bit, con $0 \leq x \leq h$, per ogni colonna che rappresenta il livello x (sia in salita che in discesa), mentre i restanti saranno impostati a -1 e non saranno considerati.

La matrice binaria, $M^{(i,l)}$, che rappresenta la configurazione, $C^{(i,l)}$, della rete al passo i sarà generata partendo dalla matrice $M^{(1,l)}$.

Si costruisca $M^{(1,l)}$ in modo tale che ogni riga è la rappresentazione binaria di lunghezza $(2h - 1) \log_2 m + 2 \log_2 w$ di un intero $l \in \{0, \dots, w^h - 1\}$, poichè con $h \log_2 m + \log_2 w = \log_2 w^h m^h$ bit si possono rappresentare $2^{\log_2 w^h m^h} = w^h m^h$ interi, esattamente il numero di permutazioni da instradare sulla rete per realizzare un quadrato latino.

In questo modo le prime $(h - 1) \log_2 m + \log_2 w$ colonne della matrice $M^{(1,l)}$ saranno formate da tutti 0, quindi tutti i nodi dei livelli in salita saranno a straight.

Tuttavia si ricorda che i nodi dei livelli intermedi devono essere modificati in ogni passo per determinare quali sono le informazioni da instradare, perciò le matrici $M^{(i,l)}$ con $1 \leq i \leq \lceil \frac{m}{w} \rceil$ saranno costruite a partire dalla matrice $M^{(1,l)}$ modificando soltanto i bit che rappresentano lo stato di un nodo nel livello intermedio.

Quindi ogni elemento $m_j^{(i,l)}$ della colonna j , dove $\log_2 w \leq j \leq (h - 1) \log_2 m$ oppure $h \log_2 m + 1 \leq j \leq (2h - 1) \log_2 m$, della matrice $M^{(i,l)}$ si ottiene con:

$$m_j^{(i,l)} = (m_j^{(1,l)} + w(i - 1)) \bmod m$$

Analogamente dato l'intero $l \in \{0, 1, \dots, w m^h - 1\}$ la cui rappresentazione binaria di lunghezza $(2h - 1) \log_2 m + 2 \log_2 w$ è $l_{((2h-1) \log_2 m + 2 \log_2 w)-1} \dots l_1 l_0$, dove i primi $(h - 1) \log_2 m + \log_2 w$ bit sono 0, è possibile generare ogni elemento, $m_j^{(i,l)}$, della colonna j della matrice $M^{(i,l)}$ dalla matrice $M^{(1,0)}$ nel seguente modo:

- se $\log_2 w \leq j \leq (h - 1) \log_2 m$

$$m_j^{(i,l)} = (m_j^{(1,0)} \oplus l_j + w(i - 1)) \mod m$$

- altrimenti

$$m_j^{(i,l)} = m_j^{(1,0)} \oplus l_j$$

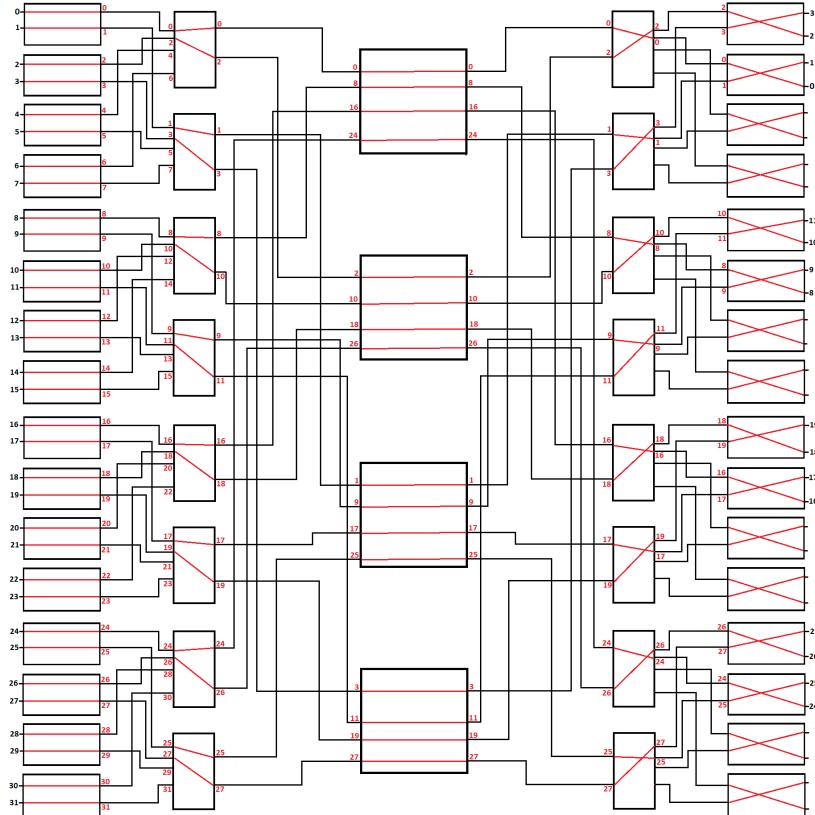


Figura 5.8. Configurazione $C^{(1,3)}$ di un $GFT(2, 4, 2)$

Le Figure 5.8 e 5.9 mostrano le configurazioni $C^{(1,3)}$ e $C^{(2,3)}$ di un $GFT(2, 4, 2)$

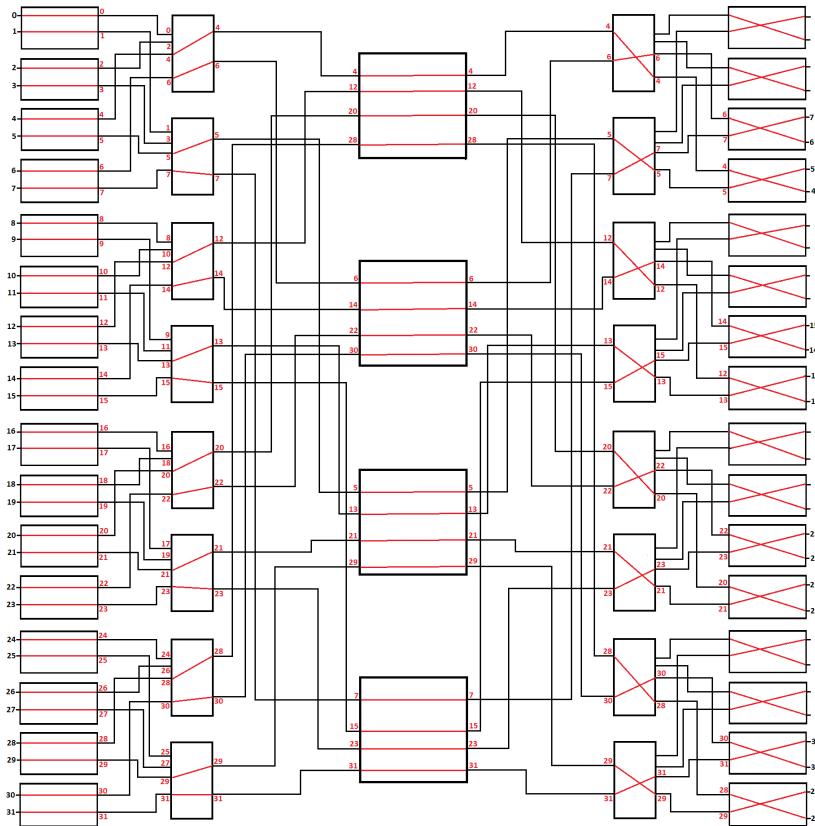


Figura 5.9. Configurazione $C^{(2,3)}$ di un $GFT(2, 4, 2)$

5.2.3 Partizione delle permutazioni in quadrati latini

Si vuole dimostrare che le permutazioni che possono essere instradate su uno Slimmed Fat-Tree, con i nodi in salita fissati, possono essere partizionate in quadrati latini, come illustrato nel paragrafo 5.1.3 per i Fat-Tree.

Una configurazione di uno Slimmed Fat-Tree verrà rappresentata attraverso $\lceil \frac{m}{w} \rceil$ matrici binarie di dimensione $m^h \times (2h - 1) \log_2 m + 2 \log_2 w$, poiché sono necessari $\lceil \frac{m}{w} \rceil$ passi per realizzare una comunicazione all-to-all personalizzata su un $GFT(h, m, w)$ con $m > w$.

Ogni matrice M ha dimensione $m^h \times (2h - 1) \log_2 m + 2 \log_2 w$ e può generare $2^{m^h(2h-1)\log_2 m+2\log_2 w}$ configurazioni diverse, tuttavia non si vogliono modificare gli stati dei nodi in salita perciò possono essere generate in realtà $2^{m^h h \log_2 m + \log_2 w}$ configurazioni.

Inoltre come discusso nel paragrafo 5.2.2, si considerano soltanto i primi $w^x m^{h-x}$ bit, con $0 \leq x \leq h$, di ogni colonna perciò si avranno $2^{(m^{h-x} w^x) \log_2 m + m^h \log_2 w}$ configurazioni, e poiché $\lceil \frac{m}{w} \rceil$ di queste servono a realizzare una sola configurazione si potranno instradare soltanto $p = \frac{2^{(m^{h-x} w^x) \log_2 m + m^h \log_2 w}}{\lceil \frac{m}{w} \rceil}$ permutazioni, con $0 \leq x \leq h$.

Sarà mostrato un metodo per raggruppare tutte le permutazioni di uno Slimmed Fat-Tree in $\frac{p}{wm^h}$ quadrati latini.

Si consideri la matrice $M^{(1,0,k)}$ che rappresenta la configurazione della rete al passo 1 e la cui sequenza di colonne è la rappresentazione binaria di k con $0 \leq k \leq \frac{p}{wm^h} - 1$. In questo modo le prime $(h - 1) \log_2 m + \log_2 w$ colonne della matrice $M^{(1,0,k)}$ sono formate da 0.

Tutte le matrici $M^{(i,0,k)}$ con $1 \leq i \leq \lceil \frac{m}{w} \rceil$ saranno costruite a partire dalla matrice $M^{(1,0,k)}$ modificando soltanto i nodi nei livelli intermedi come descritto nel paragrafo 5.2.2, e da queste saranno generate tutte le matrici $M^{(i,l,k)}$ con $l \in \{0, 1, \dots, wm^h - 1\}$, attraverso wm^h passi XOR.

Data $l_{((2h-1)\log_2 m+2\log_2 w)-1} \dots l_1 l_0$ la rappresentazione binaria di l (dove i primi $(h - 1) \log_2 m + \log_2 w$ bit sono 0) si può ottenere ogni riga, $r_j^{(i,l,k)}$, della matrice $M^{(i,l,k)}$ dalla riga $r_j^{(i,0,k)}$ della matrice $M^{(i,0,k)}$ con $0 \leq j \leq m^h$:

$$r_j^{(i,l,k)} = r_j^{(i,0,k)} \oplus l_{((2h-1)\log_2 m+2\log_2 w)-1} \dots l_1 l_0$$

- Esempio

Dato un $GFT(2, 4, 2)$ la cui configurazione è rappresentata dalle matrici $M^{(1,0,6)}$ e $M^{(2,0,6)}$ si ottengono tutte le matrici $M^{(1,x,6)}$ e $M^{(2,x,6)}$ con $0 \leq x \leq 63$ che formano un quadrato latino.

La Figura 5.10 rappresenta soltanto le matrici $M^{(1,x,6)}$ e $M^{(2,x,6)}$ con $0 \leq x \leq 2$.

Figura 5.10. Configurazioni per la generazione delle prime 2 permutazioni di un quadrato latino su un $GFT(2, 4, 2)$ a partire da $M^{(1,0,6)}$ e $M^{(2,0,6)}$

```

import Fat_Tree_utility as ftu
import math

# GFT(h, m, w)
# h - altezza
# m - copie di GFT(h-1,m,w) durante la ricorsione
# w - nodi da aggiungere al livello h
# l -
# MiOk - matrice la cui sequenza di colonne è rappresentazione binaria di k = 0,...,m^((h+1)m^h-(h+1)) al passo i
#p- passo

def xor(h, m, w, l, MiOk,p):
    # m ingressi per ogni nodo livello h
    # w*h numero nodi livello h
    # Milk - generata da MiOk con w*m**h passi XOR con la rappresentazione binaria di l
    Milk = []
    Mi = [m * w ** h][2*(h+1)] matrice che rappresenta le permutazioni intermedie in tutti gli stadi al passo i
    Mi = []

    # settiamo la matrice generata da MiOk con w*m**h passi XOR con la rappresentazione binaria di l
    lbin = ftu.de2bi(l, h * int(math.log(m, 2))+int(math.log(w, 2)))
    for x in range(m ** h):
        Milk.append([])
        for y in range(h * int(math.log(m, 2)) + int(math.log(w, 2))):
            if MiOk[x][y] != -1:
                Milk[x].append(MiOk[x][y] ^ int(lbin[y]))
            else:
                Milk[x].append(MiOk[x][y])

    #inizializziamo la matrice
    for x in range(w * m ** h):
        Mi.append([-1])

    # settiamo la colonna 0 della matrice con gli ingressi della rete al livello top
    posElem = 0
    for x in range(m ** h):
        for y in range(m):
            Mi[posElem][0] = x + (m * w) * y + w**2*(p-1)
            posElem = posElem + 1

    # settiamo le altre colonne della matrice dividendo tra le colonne che rappresentano
    # l'instradamento all'interno di un nodo a seconda del livello e quelle che rappresentano l'instradamento
    # da un livello a un altro della rete
    livello = int(math.log(m, 2))
    for colonna in range(0, (h + 1) * 2 - 1):
        nodo = 0

        # livello top
        if (colonna == 0):
            # leggiamo ogni riga della matrice Milk
            for i in range(w ** h):
                # generiamo la stringa di lunghezza log m confnodo che rappresenta la configurazione del nodo nodo
                confnodo = ""
                for j in range(0, int(math.log(m, 2))):
                    confnodo = confnodo + ftu.de2bi[Milk[i][j], 1]
                # utilizziamo lo xor per determinare come gli ingessi vengono instradati verso le uscite del nodo
                # poiché le configurazioni ammissibili dei nodi si ottengono con l'operazione xor

                for posElem in range(nodo, nodo + m):
                    nuovaPos = posElem ^ int(confnodo, 2)
                    Mi[posElem].append(Mi[nuovaPos][colonna])
                nodo = nodo + m

        #livello 0
        elif (colonna == 2 * (h + 1) - 2):
            k = (p-1)*w
            for i in range((p-1)*w,m * w+(p-1)*w):
                # generiamo la stringa di lunghezza log w confnodo che rappresenta la configurazione del nodo nodo
                confnodo = ""
                for j in range(h*int(math.log(m, 2)),h*int(math.log(m, 2))+int(math.log(w, 2))):
                    confnodo = confnodo + ftu.de2bi[Milk[k][j], 1]
                if (i % w != 0):
                    k = k + (w-1)
                else:
                    k = k+1
                # utilizziamo lo xor per determinare come gli ingessi vengono instradati verso le uscite del nodo
                # poiché le configurazioni ammissibili dei nodi si ottengono con l'operazione xor

                for posElem in range(nodo, nodo + w):
                    nuovaPos = posElem ^ int(confnodo, 2)
                    Mi[posElem].append(Mi[nuovaPos][colonna])
                nodo = nodo + w

        # colonne che rappresentano l'instradamento da un livello a un altro della rete
        elif ( colonna % 2 == 1):
            posElem = 0
            for numSubFatTree in range(m * w):
                nuovaPos = numSubFatTree
                for j in range(posElem, posElem + w):
                    Mi[j].append(Mi[nuovaPos][colonna])
                nuovaPos = nuovaPos + m * w
                posElem = j + 1
        #livelli intermedi
        else:
            for i in range(m * w):
                # generiamo la stringa di lunghezza log m confnodo che rappresenta la configurazione del nodo nodo
                confnodo = ""
                for j in range(livello, livello + int(math.log(m, 2))):
                    confnodo = confnodo + ftu.de2bi[Milk[i][j], 1]
                # utilizziamo lo xor per determinare come gli ingessi vengono instradati verso le uscite del nodo
                # poiché le configurazioni ammissibili dei nodi si ottengono con l'operazione xor

                for posElem in range(nodo, nodo + w):
                    nuovaPos = posElem ^ int(confnodo, 2)
                    Mi[posElem].append(Mi[nuovaPos][colonna])
                nodo = nodo + w

    #restituiamo la permutazione finale
    permut = []
    for i in range(m ** h):
        permut.append(Mi[i][2*(h+1)-1])
    return permut

```

Figura 5.11. Funzione $xor(h, m, w, l, MiOk, p)$ che restituisce la permutazione parziale del passo i generata dalla matrice $M^{(i,l,k)}$ su un $GFT(h, m, w)$ con $m > w$

Conclusioni

In questa relazione è stato affrontato il problema di realizzare una comunicazione personalizzata all-to-all su architetture di calcolo che utilizzano reti Fat-Tree e Slimmed Fat-Tree per la comunicazione.

Si è deciso di considerare le reti Slimmed Fat-Tree, poichè le dimensioni dei loro nodi sono inferiori a quelle di un Fat-Tree semplice, e ciò le rende meno costose.

Tuttavia impedisce che tutte le informazioni dalle sorgenti vengano instradate nello stesso momento, pertanto è necessario eseguire più passi per realizzare completamente la comunicazione.

La comunicazione all-to-all su reti Fat-Tree e Slimmed Fat-tree, costruite attraverso il modello Generalized Fat Tree, $GFT(h, m, w)$, è stata realizzata instradando le wm^h permutazioni che formano un quadrato latino.

Sono stati presentati due metodi per ottenere la comunicazione all-to-all:

- il primo metodo segue lo schema fornito dal Left Latin Square, LLS, che consiste nella permutazione dell'identità e di tutte le sue $wm^h - 1$ rotazioni verso sinistra;
- il secondo metodo si basa sul Canonical Latin Square, CLS, un quadrato latino le cui permutazioni si ottengono scambiando di volta in volta sottoinsiemi di informazioni di dimensioni diverse.

È stato mostrato che ogni permutazione del Left Latin Square può essere ottenuta semplicemente conoscendo l'indice della rotazione che si vuole andare a realizzare. Una rotazione del LLS si ottiene dalla configurazione dei nodi del Fat-Tree usata per la rotazione precedente, modificando lo stato di un solo nodo al livello top e impostando gli stati dei nodi negli altri livelli attraverso l'algoritmo di self-routing.

Per il Canonical Latin Square è sufficiente utilizzare una matrice binaria che rappresenti la configurazione della rete, in cui ogni nodo è impostato a straight, e eseguire un'operazione di XOR tra ogni riga della matrice e la rappresentazione binaria di un intero $l \in \{0, 1 \dots wm^h - 1\}$.

Inoltre nel capitolo 5 si dimostra che tutte le permutazioni che si possono realizzare su una rete Fat-Tree, fissando i nodi dei livelli in salita a straight, possono essere partizionate in quadrati latini. Uno di questi partizionamenti è stato realizzato costruendo in modo opportuno matrici binarie che rappresentano la rete e utilizzando lo stesso metodo presentato per il Canonical Latin Square, ovvero eseguendo l'operazione di XOR tra ognuna di queste matrici e la rappresentazione binaria dei numeri da 0 a $wm^h - 1$.

Tuttavia è stato presentato un solo modo di partizionare le permutazioni in quadrati latini, nei quali rientra lo stesso Canonical Latin Square. Sarebbe sicuramente interessante esplorare altri metodi per costruire le configurazioni di un Fat-Tree che realizzino quadrati latini diversi da quelli ottenuti, mantenendoli indipendenti dai parametri m e w del Generalized Fat-Tree.

Bibliografia

- [1] M.F. Barozzi. La letteratura combinatoria (2). <https://keespopinga.blogspot.com/2009/02/la-letteratura-combinatoria-ii.html>, febbraio 2009.
- [2] V. E. Beneš. On rearrangeable three-stage connecting networks. *Bell Syst. Tech. J.*, XLI:1481–1492, 1962.
- [3] C. Clos. A study of non-blocking switching networks. *B.S.T.J.*, pages 406–424, 1953.
- [4] A. Grama, V. Kumar, G. Karypis, and A. Gupta. *Introduction to Parallel Computing, 2nd edition*. Pearson, 2003.
- [5] D. Izzi and A. Massini. Optimal all-to-all personalized communication on butterfly networks through a reduced latin square. In *IEEE 22nd International Conference on High Performance Computing and Communications (HPCC)*, pages 1065–1072, 2020.
- [6] C. E. Leiserson. Fat-trees: Universal network for hardware-efficient supercomputing. *Transazioni IEEE sui computer*, pages 892–901, 1985.
- [7] V.W. Liu, C. Chen, and R.B. Chen. Optimal all-to-all personalized exchange in d-nary banyan multistage interconnection networks. *Journal of Combinatorial Optimization*, 14:131–142, 2007.
- [8] A. Massini. All-to-all personalized communication on multistage interconnection networks. *Discrete Applied Mathematics*, 128(2-3):435–446, 2003.
- [9] W. Nienaber. *Effective Routing on Fat-Tree Topologies*. PhD thesis, Florida State University, 2014.
- [10] S.R. Ohring, M. Ibel, S.K. Das, and M.J. Kumar. On generalized fat trees. *Proceedings of 9th International Parallel Processing Symposium*, pages 37–44, 1995.
- [11] D. Singh. Demystifying DCN Topologies: Clos/Fat Trees – Part1. <https://packetpushers.net/demystifying-dcn-topologies-clos-fat-trees-part1/>, November 2018. consultato: Novembre 2021.
- [12] Y. Yang and J. Wang. Optimal all-to-all personalized exchange in self-routable multistage networks. *IEEE Trans. Parallel Distrib. Systems*, 11(3):261–274, 2000.