Facoltà di Scienze MM.FF.NN.

INSTRADAMENTO DI MESSAGGI SULLA RETE MULTIBUTTERFLY

Relatore: Ch.Mo Prof. Livio Colussi

Correlatore: Dott. Andrea Pietracaprina

Laureando: Alessio Gianelle

25/02/1997

Sommario

Lo scambio di dati tra i vari processori è un problema fondamentale del calcolo parallelo. Questo problema, noto col nome di routing, è stato a lungo studiato il letteratura. Esistono vari problemi di routing a seconda di come sono distribuiti i messaggi e del loro numero. Se abbiamo N messaggi che devono essere instradati su una N-input rete, ciascuno con destinazione diversa, parleremo di routing di una permutazione. Se invece, sempre nella stessa rete, abbiamo hN messaggi e ogni nodo di input contiene h messaggi e ogni nodo di output è destinazione di h messaggi, allora parleremo di h-relation. In particolare osserviamo che il primo problema è un caso particolare del secondo, in cui si è posto $h = 1$. Le h-relation hanno acquistato importanza soprattutto con la formulazione dei nuovi modelli computazionali, in quanto molti di questi prevedono, come primitiva di comunicazione, l'h-relation. Nella tesi abbiamo analizzato il problema del routing di h-relation sulla rete multibutterfly [22]. Questa si ottiene come generalizzazione di una rete della famiglia del cubo, la rete butterfly. Infatti possiamo vedere una multibutterfly come la sovrapposizione di più reti butterfly, nelle quali gli archi sono stati opportunamente permutati.

Per prima cosa analizzeremo in dettaglio i principali risultati noti in letteratura, relativi all'instradamento di messaggi sulla multibutterfly. In particolare descriveremo due algoritmi che realizzano in tempo asintoticamente ottimo il routing di una permutazione [22, 3]. Non si conoscono ancora degli algoritmi che instradino le h-relation, ad eccezione di quello di Maggs e Vöcking (pubblicato recentemente in [13], mentre questo lavoro era in fase di elaborazione), che però utilizza una estensione della multibutterfly con $N(\log N + h)$ nodi. Quello che descriveremo nella tesi sarà quindi il primo algoritmo deterministico che realizza l'instradamento di una h-relation su una N-input multibutterfly con soli $N(\log N + 1)$ nodi e con buffer di taglia $\Theta(h)$. Il tempo di routing del nostro algoritmo è $O(\log N(\log N + h))$, ossia peggiore rispetto al tempo ottimo solo di un fattore logaritmico. Contando

solo il contributo dovuto alla comunicazione, il tempo dell'algoritmo scende a $O(\log^2 N + h)$. Oltre a un'analisi sulla complessità dell'algoritmo, abbiamo studiato delle scelte ottimali per i parametri coinvolti nella definizione della rete e nella descrizione dell'algoritmo.

Capitolo 1

Introduzione

Nei computer sequenziali il singolo processore esegue le varie istruzioni una di seguito all'altra secondo un dato ordine cronologico. In una macchina parallela le istruzioni possono essere eseguite simultaneamente da più processori, incrementando così notevolmente il numero di operazioni eseguibili dalla macchina in un secondo. I numerosi progressi che si sono ottenuti in campo tecnologico, hanno permesso di aumentare notevolmente la potenza di un singolo processore. Negli ultimi vent'anni, infatti, questa cresceva circa di un fattore 2 ogni 2 anni. Questa crescita non può essere infinita, in quanto esistono dei limiti fisici, quali la velocità della luce e la limitatezza dei materiali, che segneranno un asintoto per questa crescita. Dall'altra parte, però, il rapido decrescere dei costi per la realizzazione di processori e banchi di memoria, e la forte richiesta di prestazioni sempre più veloci da parte di nuove tecnologie (come l'elaborazione delle immagini o la realizzazione di modelli per lo studio di problemi di fluidodinamica), fanno apparire inevitabile che il predominio, finora incontrastato, dei computer sequenziali, sia preso da quelli paralleli. A differenza dalle aspettative questo passaggio non è ancora avvenuto. Una delle cause principali di questo fatto è dovuta alla mancanza di un modello universale per il calcolo parallelo. Infatti il successo della programmazione sequenziale è dovuto anche all'esistenza del modello di von Neumann. Quest'ultimo riesce ad astrarre le architetture

dei computer sequenziali esistenti, consentendo di realizzare algoritmi che possono essere eseguiti efficientemente su un qualsiasi computer reale senza conoscerne la particolare struttura. Ciò permette di progettare algoritmi basandosi solo su questo modello, sapendo che questi possono poi essere eseguiti efficientemente su una qualsiasi macchina sequenziale. In altri termini il modello di von Neumann è un ponte che connette il mondo caotico del software con quello altrettanto caotico dell'hardware, permettendo al primo di essere eseguito efficientemente nel secondo. È chiara quindi la necessità di trovare un modello per la programmazione parallela che esibisca le stesse caratteristiche di universalità del modello di von Neumann. Un modello cioè sufficientemente semplice da assicurare delle opportune basi per lo sviluppo del software, sufficientemente accurato da garantire previsioni realistiche sulle performance, e sufficientemente generale da permettere al software di essere eseguito efficientemente su un ampio raggio di architetture. Uno dei maggiori candidati a questo ruolo è il *bulk-synchronous parallel (BSP) model* introdotto da Valiant in [23]. Una delle caratteristiche principali di questo modello è il fatto che tiene divisa la parte che riguarda la computazione dei dati (parte lasciata al programmatore) da quella che riguarda la gestione della memoria e la comunicazione tra i processori per lo scambio dei valori calcolati. Vediamo quindi in dettaglio come Valiant definisce il modello BSP. Un BSP è caratterizzato da tre attributi (si veda la Figura 1.1).

- Un numero di *componenti* in grado di compiere operazioni di calcolo e/o memoria.
- Un *router* che si preoccupa di scambiare i messaggi tra i vari componenti.
- Una primitiva che permette di *sincronizzare* tutti o una parte dei componenti ad intervalli regolari.

Un programma consiste in una sequenza di superstep. Ogni *superstep* si divide in tre fasi consecutive: la prima è dedicata alla *computazione locale*

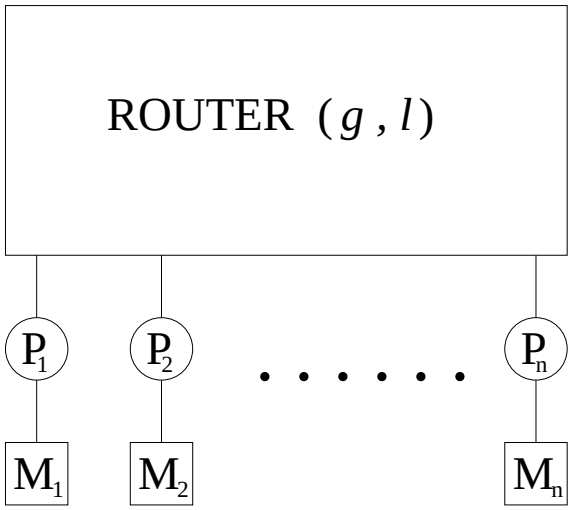

Figura 1.1: Il modello BSP. Il router è l'elemento dedicato allo scambio dei messaggi ed è caratterizzato dai due parametri g e ℓ. I P_i sono i vari processori, ciascuno dotato di una propria memoria M_i.

(all'interno cioè di ogni singolo componente); la seconda alla *comunicazione globale* (che interessa cioè tutti i componenti); la terza alla *sincronizzazione* tra i vari componenti. Ogni componente i, può essere visto come una coppia (P_i, M_i) dove P_i è un processore e M_i è un modulo di memoria. Il processore P_i, è inoltre dotato di un'*unità di output*, in cui vengono inseriti i messaggi che devono essere spediti, e di un'*unità di input*, da cui vengono prelevati i messaggi che sono arrivati. Durante la prima fase di un superstep ogni processore estrae messaggi dall'unità di input, compie delle operazioni di calcolo su questi e sui dati presenti nel modulo locale, e genera dei messaggi, destinati ad altri processori, che vengono inseriti nell'unità di output. Durante la fase di comunicazione globale, ogni messaggio contenuto nell'unità di output di un processore è trasferito nell'unità di input del processore di destinazione. Il superstep, infine, si conclude con la fase di sincronizzazione in cui viene comunicato a ogni processore che tutte le computazioni locali sono terminate e che ogni messaggio generato nel superstep è arrivato alla sua destinazione. Il tempo di esecuzione di un superstep si esprime in funzione di due parametri,

g e ℓ, e vale

$$T_{superstep} = w + gh + \ell$$

dove w è il numero massimo di operazioni locali eseguibili da un processore e h è il massimo numero di messaggi spediti o ricevuti da un singolo processore durante il superstep. Il parametro ℓ può essere visto come un limite superiore al tempo richiesto per una sincronizzazione globale ($w = 0$ e $h = 0$). Invece $g + \ell$ è un limite superiore al tempo necessario per instradare una permutazione parziale ($w = 0$ e $h = 1$). Più in generale il BSP si basa sull'assunzione che il router esegua una h-relation e una sincronizzazione globale in tempo $gh + \ell$ per una qualche coppia di parametri g, ℓ che caratterizzano il router. Per maggiori dettagli si può vedere anche [8] dove il BSP viene messo a confronto con un altro modello per la programmazione parallela, il LogP.

A questo punto diventa interessante vedere come una macchina reale possa realizzare l'astrazione BSP. Essenzialmente questa deve poter essere in grado di compiere una sincronizzazione globale e il routing di una h-relation, per un qualsiasi valore di h non noto a priori, in tempo soddisfacente, cioè pari a $gh + \ell$ per degli opportuni parametri fissati g e ℓ. In questa tesi ci soffermeremo ad analizzare il secondo aspetto, in quanto più rilevante. In particolare studieremo una rete a elevate prestazioni, la multibutterfly, e descriveremo un algoritmo deterministico per questa rete che consente di realizzare una h-relation in tempo $O(\log^2 N + h \log N)$. Cominceremo la nostra analisi da un caso particolare di h-relation, ossia il routing di una permutazione (o 1-relation), in quanto questo è stato a lungo studiato in letteratura.

1.1 Il problema del routing

Abbiamo visto che una differenza sostanziale tra i computer sequenziali (dotati cioè di un singolo processore) e i multiprocessori, riguarda il fatto che in quest'ultimi un grosso peso è assunto dalle comunicazioni tra i processori. In questa sezione analizzeremo in dettaglio il problema del routing, ossia

quello di instradare dei messaggi o dei dati attraverso la rete, riassumendo i principali risultati noti in letteratura.

Prendendo spunto da [24] possiamo dire che la soluzione di un *problema di routing* può essere formalizzata attraverso una tripla, (G, R, Q) dove $G = (V, E)$ è un *grafo diretto*, R è un *algoritmo di routing* e Q è una *disciplina di coda*. In particolare diremo che questo è un problema *inizializzato* se consiste di una quintupla, (G, R, Q, I, D), dove I è la *specificazione dell'input* e D è la *specificazione dell'output*. Ossia, detto X un qualsiasi insieme di messaggi, I è una mappa da $X \longrightarrow V$ che specifica, per ogni messaggio dell'insieme X, il nodo in cui questo si trova inizialmente, e D è una mappa da $X \longrightarrow V$ che assegna ad ogni messaggio di X l'indirizzo della destinazione desiderata. In ciascun passo l'algoritmo di routing, R, decide, per ogni messaggio e per ogni nodo, in quale nodo vicino spedire il messaggio, se possibile, durante il passo successivo. Ogni arco diretto $e \in E$ ha una coda q_e, dove vengono posti in attesa tutti i messaggi che nel prossimo passo devono usufruire dell'arco e. La disciplina di coda Q determina quale di questi messaggi può effettivamente attraversare l'arco durante il passo successivo. Si assume infine che ogni messaggio richieda un passo dell'algoritmo per attraversare un arco.

In base al valore della coppia (I, D) possiamo avere tre tipi di problemi. Se $|X| = |V|$ e, I e D, sono entrambe delle funzioni biiettive, allora parleremo del routing di una *permutazione*. In particolare se $|X| < |V|$ avremo il routing di una *permutazione parziale*. Infine se $|X| \leq h|V|$ e per ogni $v \in V$ ci sono al più h messaggi che partono da v e che sono destinati a v, allora si parlerà di una *h-relation*. Esistono anche altri tipi di problemi di routing, come ad esempio i problemi *many-to-one*, in cui più di un messaggio può avere la stessa destinazione, e quelli *one-to-many*, dove un singolo messaggio può avere più di una destinazione. Questi possono però rientrare nella prima suddivisione con piccoli accorgimenti. I problemi many-to-one possiamo vederli come delle permutazioni (o al limite delle permutazioni parziali) se diamo la possibilità a più messaggi, aventi la stessa destinazione, di combinarsi in uno solo con la stessa destinazione. Ovviamente quest'ultimo sarà

più grande; ossia ammettere questa possibilità significa che un nodo può contenere più messaggi nello stesso istante. I problemi one-to-many, invece possono essere visti come delle h-relation. Infatti se un messaggio è destinato ad h nodi, possiamo crearci h copie di quel messaggio e assegnare a ognuna una destinazione diversa. Per un'analisi più dettagliata su questi problemi si veda [9].

Un problema di routing può essere risolto sostanzialmente in due modi: tramite un algoritmo on-line, oppure off-line. Negli *algoritmi on-line* il routing è eseguito dalla rete in modo distribuito; ossia ogni singolo nodo decide dove spedire il messaggio in base alle informazioni locali (contenute cioè nel nodo) e a quelle trasportate dal messaggio stesso. Invece negli *algoritmi off-line* il problema è risolto da un controllore centrale, eventualmente esterno alla rete, che conosce l'intera struttura della rete e le destinazioni di tutti i messaggi. Il controllore centrale stabilisce i cammini per i messaggi e invia le informazioni neccessarie, per eseguire effettivamente l'instradamento, ai singoli nodi della rete. Nel caso in cui un'informazione globale sia disponibile a tutti i nodi, il problema del routing di una permutazione diventa abbastanza semplice. Ad esempio tramite la rete di Beneš [7], costituita da due reti butterfly collegate opportunamente, è possibile realizzare una qualsiasi permutazione in $2 \log N$ passi, senza code. Osserviamo che se lo stesso schema di routing deve essere usato per spedire molti messaggi, allora risulta conveniente calcolare prima i cammini, e una volta distribuite queste informazioni sulla rete, instradare i messaggi. Algoritmi paralleli per eseguire questa precomputazione si possono trovare in [12, 14, 19]. Nel seguito riporteremo invece i principali risultati che riguardano il routing on-line, che è anche l'oggetto di questa tesi.

Definiamo innanzitutto due modelli di routing. Nel *modello store-and-forward* ogni messaggio è visto come un'entità atomica che i nodi della rete possono conservare in un buffer o scambiarsi tramite gli archi. Nel *modello di circuit-switching*, l'intero cammino dal nodo sorgente al nodo di destinazione, è dedicato al messaggio, per permettere una trasmissione continua dei dati

del messaggio stesso. Questo secondo modello è più chiaro se pensiamo a una rete telefonica; ogni linea, una volta stabilita, è completamente dedicata ai due utenti per il periodo in cui essi desiderano comunicare. La differenza sostanziale tra i due modelli è che nel secondo si richiede praticamente di costruire dei cammini che siano edge-disjoint; ciò rende inutile l'utilizzo di buffer sugli archi della rete. Il modello store-and-forward risulta invece più semplice, in quanto non pone delle restrizioni su come devono essere fatti i cammini, consentendo anche a più messaggi di trovarsi nello stesso nodo nello stesso tempo. In altri termini un algoritmo di circuit-switching funziona anche per il modello store-and-forward, mentre il viceversa non è quasi mai vero.

Vediamo ora un risultato generale che fissa un limite inferiore per un tipo particolare di algoritmi deterministici di routing: ossia gli algoritmi di routing del tipo oblivious. Ricordiamo che un algoritmo si dice *oblivious* se il cammino che un messaggio deve seguire può essere determinato usando esclusivamente il nodo di origine e quello di destinazione. Riportiamo quindi il Teorema 3.23 che si trova in [9]:

TEOREMA 1. *Sia $G = (V, E)$ una rete di N nodi di grado d. Sia A un algoritmo di routing oblivious per G. Allora esiste una permutazione per cui A richiede almeno $\sqrt{N}/2d$ passi per instradarla sulla rete.*

Prima di passare a una rassegna dei maggiori risultati noti in letteratura definiamo una caratteristica della rete che ci permetterà di dare un limite inferiore per un qualsiasi algoritmo di routing. Definiamo come *diametro* di una rete la massima lunghezza tra tutti i cammini di lunghezza minima che collegano due nodi. Ovviamente sono desiderabili reti con diametro piccolo, in quanto queste permettono ai nodi di scambiarsi più velocemente i messaggi. In particolare sussiste il seguente teorema:

TEOREMA 2. *Il diametro di una rete costituisce un limite inferiore naturale per il tempo di esecuzione, al caso pessimo, di un qualsiasi algoritmo di routing per la rete.*

1.2 Risultati noti in letteratura

Ci proponiamo ora di fare una breve rassegna dei principali risultati che ci sono in letteratura riguardanti i due modelli suddetti. Ci soffermeremo maggiormente sul modello store-and-forward, non solo perché più semplice, ma anche perché è il modello che abbiamo usato per realizzare il nostro algoritmo.

1.2.1 Il modello store-and-forward

Cominceremo la nostra analisi dalle reti più semplici. Nei linear-array, il routing di una permutazione risulta molto semplice. Infatti attraverso un algoritmo greedy si può risolvere il problema in $N-1$ passi, senza usare code. Poiché il diametro di un linear array è $N-1$, dal Teorema 2 si deduce che questo tempo è il migliore possibile. Le cose però non funzionano altrettanto bene nelle altre reti, e questo perché, negli algoritmi greedy, non si cerca in alcun modo di evitare collisioni tra i messaggi. Ad esempio se permettiamo alle code di crescere illimitatamente, un algoritmo greedy su un $\sqrt{N} \times \sqrt{N}$ array, richiede $2\sqrt{N}-2$ passi per eseguire il routing. Anche in questo caso il tempo è il migliore possibile, ma nel caso peggiore possiamo avere code di taglia $\frac{2}{3}\sqrt{N}-1 \in \Theta(\sqrt{N})$. Fortunatamente le cose funzionano molto meglio al caso medio. Ad esempio se consideriamo delle permutazioni casuali (in cui cioè le destinazioni dei messaggi sono scelte casualmente) si può dimostrare che con probabilità molto vicina a 1, si formano code con al più $O(1)$ messaggi. Prestazioni molto simili si possono ottenere anche in modo deterministico; esiste infatti un algoritmo deterministico che riesce ad eseguire il routing in $2\sqrt{N}+\frac{4\sqrt{N}}{q}+o(\frac{\sqrt{N}}{q})$ passi usando code di taglia $2q-1$, per ogni q tale che $1 \leq q \leq \sqrt{N}$. Ricordiamo infine che esiste anche un altro algoritmo, più complesso di quelli citati prima, che richiede tempo $2\sqrt{N}-2$ con code di taglia $O(1)$, in cui però la costante risulta abbastanza alta. Sempre per quanto riguarda gli array $\sqrt{N} \times \sqrt{N}$, esiste un algoritmo off-line, molto semplice, che risolve il routing di una permutazione in $3\sqrt{N}-3$ passi,

con code di taglia 1. Per una descrizione più dettagliata di questo algoritmo e degli altri risultati, rimandiamo a [9].

Passiamo ora ad alcune reti più complesse; in particolare analizzeremo la classe delle reti ipercubiche, in quanto, non solo è stata a lungo studiata, ma anche perché assicura delle buone prestazioni per i problemi di routing. Il rappresentante più importante per questa classe di reti è l'N-cubo. L'*N-cubo* è una coppia (V, E) dove $V = \{0, 1, \ldots, N-1\}$ ed $E = \{(w, w_i) \mid w \in V,\ i \in \{0, 1, \ldots, \log N - 1\}\}$[1]. Questa rete è conosciuta anche come l'ipercubo binario di dimensione $\log N$. Per ogni i l'arco (w, w_i) è detto arco di dimensione i. Osserviamo che l'N-cubo ha N nodi, e ha sia grado che diametro uguali a $\log N$. Dal cubo deriviamo le altre reti della stessa classe, come ad esempio, l'*N-Cube-Connected-Cycle* (N-CCC), la *N-butterfly* e l'*N-suffle-exchange*. Per quanto riguarda gli algoritmi deterministici di routing possiamo dire che la migliore strategia, finora conosciuta, sfrutta un'importante proprietà di queste reti. Infatti si può dimostrare che il routing di una permutazione, su una qualsiasi rete ipercubica, richiede $T + O(\log N)$ passi, dove T è il tempo necessario per ordinare N elementi sulla rete. Ora poiché $T = \Omega(\log N)$, allora ogni algoritmo di routing su una rete ipercubica può essere eseguito in $O(T)$ passi. Usando questa strategia l'algoritmo più veloce, tra quelli pratici, che si è riusciti a ottenere, è quello che si basa sulla rete di ordinamento di Batcher [5], che richiede tempo $O(\log N)^2$ per instradare una permutazione. Più recentemente Cypler e Plaxton hanno descritto in [18] un algoritmo piuttosto complesso, su una rete ipercubica, che richiede tempo $O(\log N(\log\log N)^2)$ per il sorting e quindi, per quanto appena visto, per il routing. Questi due risultati costituiscono i migliori limiti superiori conosciuti per il routing deterministico di una permutazione su una rete ipercubica. Sempre per quanto riguarda gli algoritmi deterministici, in particolare quelli di tipo oblivious, risultano molto interessanti quelli che sfruttano una strategia *greedy*. Questi algoritmi sono molto semplici da

[1]Con w_i denotiamo l'intero che si ottiene da w cambiando l'i-esimo bit della rappresentazione binaria di w.

realizzare, infatti ad ogni passo compiono la loro scelta analizzando un singolo bit, e consentono di calcolare per ogni messaggio il cammino più breve. Sfortunatamente, però, è semplice, per questi algoritmi, trovare una permutazione che richieda tempo N^α per un qualche $\alpha > 0$. Ad esempio, nel caso della butterfly, basta considerare la permutazione *bit-reversal*, ossia quella che spedisce nel nodo $(x', \log N)$ il messaggio contenuto nel nodo $(x, 0)$, dove x' si ottiene da x rovesciandone la rappresentazione binaria, per ottenere un tempo di esecuzione pari a $\sqrt{N/2} + \log N - 1$ [9]. Anche se esistono molti casi in cui questi algoritmi richiedono un tempo $O(\sqrt{N})$, il loro uso è molto diffuso. Infatti, esistono delle particolari permutazioni, molto usate per certi problemi, che questi algoritmi possono instradare in tempo $\log N$, ossia il migliore possibile. In particolare gli algoritmi greedy consentono di risolvere efficientemente problemi di routing monotono, problemi cioè in cui l'ordine relativi dei messaggi è preservato. A questo punto, risulta ovvio ciò che avevamo detto in precedenza; una volta ordinati i messaggi secondo la loro destinazione in tempo T, un algoritmo greedy può completare il routing in tempo $O(\log N)$.

Un modo per migliorare questo limite, sempre stando all'interno delle reti ipercubiche, è quello di usare la tecnica della randomizzazione. Nel 1981 si è scoperto infatti un algoritmo probabilistico che risolve il problema del routing, on-line, in tempo $O(\log N)$ con alta probabilità. Si veda a tal proposito [25]. In [25] Valiant e Brebner definiscono un algoritmo di routing, a due fasi, che inizialmente era stato ideato per l'N-cubo, ma successivamente, anche ad opera di altri autori, è stato esteso, con piccoli accorgimenti, alle altre reti ipercubiche [2, 21]. Nella prima fase, ogni messaggio sceglie, indipendentemente e casualmente, una destinazione, e tramite un algoritmo greedy la raggiunge. Nella seconda fase, una nuova esecuzione dell'algoritmo greedy porta i messaggi dalla destinazione intermedia alla destinazione corretta. In questo modo una permutazione arbitraria è scomposta in due permutazioni casuali che, si dimostra, possono essere instradate in tempo ottimo con alta probabilità. Osserviamo che questo algoritmo, almeno come era stato ideato

inizialmente, richiede code illimitate. Modifiche successive hanno permesso di limitare la lunghezza di queste code. Si è pervenuti così al lavoro di Ranade che nel 1987 dimostrò in [17] che su una butterfly era sufficiente una scelta casuale delle destinazioni tra un dato insieme di permutazioni, per ottenere buone prestazioni. In particolare dimostrò come modificare l'algoritmo a due fasi per la butterfly, per ottenere code di lunghezza $O(1)$, usando la disciplina di coda FIFO, senza che questo alterasse il tempo di esecuzione.

Diamo infine due teoremi, tratti dall'articolo di Valiant [24], che generalizzano ed estendono i risulati già citati. In particolare dimostrano che l'N-cubo può realizzare una $O(\log N)$-relation in tempo $O(\log N)$, mentre la butterfly e la rete CCC possono realizzare, nello stesso tempo, una $O(1)$-relation. È facile vedere che i tempi di routing sono ottimi.

TEOREMA 3. *Per ogni $\alpha > 1$ esiste una costante $\eta > 0$ tale che, se T è il tempo che ogni fase del precedente algoritmo richiede per instradare una permutazione su un N-cubo, allora*

$$Pr(T > \log N + \frac{\alpha \log N}{\log \log N}) = O(N^{-\eta})$$

Indichiamo con T ancora il tempo di esecuzione di ogni singola fase, allora sussiste anche il seguente teorema:

TEOREMA 4. *Esistono delle costanti $\alpha_0, \beta, \delta > 0$ tali che per ogni $\gamma > 0$ e per ogni $\alpha > \alpha_0$, si ha che per realizzare una γ-relation su una rete $\log N$-CCC o su una N-butterfly, oppure, per realizzare una $\gamma \log N$-relation su un N-cubo,*

$$Pr(T > \alpha\gamma \log N) \geq N^{-\alpha\beta\gamma+\delta}$$

Una dimostrazione di questi teoremi si può trovare in [24].

Esiste, un'altra classe di reti sulle quali il routing può essere effettuato in modo efficiente: le reti a espansione. Su tali reti è possibile ottenere algoritmi asintoticamente ottimi. Tuttavia dal lato pratico non sono molto utili in quanto la loro costruzione è molto complessa e i valori, asintoticamente ottimi, del tempo di routing e della taglia della rete, nascondono costanti

molto alte. Tra queste reti, ricordiamo la rete AKS [1] che consente di ordinare un insieme di N elementi in tempo $O(\log N)$, e quindi, se collegata opportunamente, ad esempio, con una rete butterfly, può risolvere il routing di N messaggi in tempo $O(\log N)$. Un'altra rete che gode di proprietà di espansione, ma la cui costruzione risulta più semplice, è la multibutterfly [22]. Questa rete, che si ottiene sovrapponendo più reti butterfly, consente di eseguire il routing di N messaggi in tempo $O(\log N)$. Essendo questa l'oggetto della nostra tesi, ne abbiamo dato una descrizione dettagliata nel Capitolo 2. Sempre nel suddetto capitolo si possono trovare anche la descrizione dettagliata dei principali algoritmi di routing, noti in letteratura, per questa rete.

1.2.2 Il modello di circuit-switching

Un problema di circuit-switching può essere risolto principalmente in due modi diversi. Possiamo cioè usare una modalità "nonblocking", oppure una modalità "rearrangeable". Nel caso di modalità *nonblocking* una richiesta di un cammino, tra un nodo sorgente e un nodo destinazione, può arrivare in qualsiasi istante; questa richiesta deve essere soddisfatta senza disturbare eventuali cammini già presenti nella rete. Invece nella modalità *rearrangeable* le richieste arrivano tutte nello stesso istante e saranno soddisfatte tutte contemporaneamente. Osserviamo però che se arriva una nuova richiesta di comunicazione, bisogna sospendere tutte quelle attivate, per permettere alla rete di ricalcolare i cammini tenendo conto della nuova richiesta. La differenza tra i due modelli è abbastanza evidente. Nel modello nonblocking, ogni cammino deve essere costruito tenendo conto dei cammini che sono già presenti sulla rete e la sua realizzazione influenzerà eventuali cammini successivi. Nell'altro modello invece questo problema non sussiste in quanto i cammini sono tutti costruiti contemporaneamente. Il primo caso modella, ad esempio, una rete di comunicazione telefonica. Vedremo ora alcuni risultati riguardanti il secondo modello, in quanto più vicino allo schema di comunicazione esistente all'interno di un computer parallelo.

Per presentare i risultati evidenziamo due parametri che caratterizzano la rete. Definiamo come *taglia* della rete il numero dei suoi archi, e come *profondità* della rete, il numero degli archi che compongono il cammino più lungo che connette un nodo di input con uno di output. Informalmente possiamo dire che la taglia di una rete rappresenta il suo costo, mentre la profondità rappresenta il ritardo che un segnale può subire nell'attraversarla. Osserviamo che la rete di Beneš, di cui abbiamo già parlato, ha taglia $O(N \log N)$ e profondità $O(\log N)$; Shannon in [20] ha dimostrato che la taglia della rete di Beneš, a meno di una costante moltiplicativa, è la minore possibile, per permettere il routing nonblocking in tempo $O(\log N)$. La profondità, invece, può diminuire a spese della taglia, basta pensare al caso limite in cui esiste un arco tra ogni nodo di input e ogni nodo di output; in questo caso la profondità è 1 ma la taglia è N^2. Per completezza ricordiamo che Pippenger e A.C. Yao in [16] hanno determinato l'intervallo in cui può variare la taglia di una rete nonblocking con profondità k. In particolare questa taglia ha come limite inferiore $\Omega(N^{1+1/k})$ e come limite superiore $O((N \log N)^{1+1/k})$.

Per quanto riguarda gli algoritmi, tra quelli visti per il modello store-and-forward, quelli greedy sono in alcuni casi adattabili anche al circuit switching. Si può dimostrare che, se vogliamo realizzare N comunicazioni tra nodi casuali, tra i livelli estremi di una butterfly, allora $\Theta(\frac{N}{\log N})$ di queste potranno essere soddisfatte con alta probabilità. Precisiamo che in questo caso l'algoritmo è tale per cui se due cammini tentano di passare attraverso lo stesso arco, solo uno può proseguire, mentre l'altro ritorna al nodo da cui era partito. Quindi non è detto che tutti i cammini siano realizzabili.

Per una rassegna completa e dettagliata dei risultati noti in letteratura, preferiamo rimandare al lavoro di Pippenger [15]. Qui vogliamo solo accennare a due risultati ottenuti da Arora, Leighton e Maggs in [3]. Questi hanno ideato due algoritmi on-line per la selezione di cammini ottimi. Il primo è per la rete multibeneš, una rete nonblocking di $O(N \log N)$ nodi di grado limitato. Il secondo, invece, è per la multibutterfly, una rete che gli stessi autori dimostrano essere rearrangeable. In particolare questo secondo

algoritmo è quello che ci ha fornito gli spunti neccessari per realizzare il nostro, per questo abbiamo preferito darne un'ampia descrizione nel prossimo capitolo.

1.3 I risultati ottenuti

Vista l'importanza che i problemi di routing hanno nel campo del calcolo parallelo, in questa tesi abbiamo descritto un algoritmo deterministico per instradare una h-relation sulla rete multibutterfly. Questa rete, che si ottiene come generalizzazione della butterfly, si è rivelata essere una rete molto potente (a questo proposito si veda anche [13]), che aggiunge, alle proprietà della butterfly, una nuova proprietà piuttosto forte: la proprietà di espansione. Tale proprietà assicura che ogni dato insieme di processori abbia un numero di vicini almeno proporzionale, secondo una data costante β, alla cardinalità dell'insieme di partenza. Fino a oggi sono stati descritti solo due algoritmi sulla multibutterfly, che consentono di eseguire il routing di una permutazione in tempo ottimo. Il primo è del tipo store-and-forward, ed è stato formulato dallo stesso Upfal in [22]; il secondo invece è un algoritmo di circuit-switching ideato da Arora, Leighton e Maggs in [3]. Quello di Upfal è il primo algoritmo deterministico che non usa stategie di ordinamento e che consente di eseguire il routing di un'arbitraria permutazione in tempo $O(\log N)$ su una rete di grado limitato. Viste le ottime prestazioni che si sono ottenute, è parso naturale cercare di generalizzare i risultati già noti al caso delle h-relation. L'unico risultato che al momento si conosce è quello apparso molto di recente in letteratura: Maggs e Vöcking in [13] hanno descritto un algoritmo deterministico per il routing di una h-relation su una rete ricavata dalla multibutterfly che richiede tempo $O(h + \log N)$. Tale algoritmo lavora in tempo asintoticamente ottimo, ma fa uso di $N(h + \log N)$ nodi con buffer di taglia costante. Infatti alla rete multibutterfly vengono aggiunti h livelli di N nodi ciascuno. A differenza di questo algoritmo, quello che descriveremo nella tesi lavora sulla multibutterfly standard (una rete cioè con $N(\log N + 1)$

nodi), usando buffer di taglia $\Theta(h)$ su ogni nodo, e consente di instradare una qualsiasi h-relation con un tempo globale pari a $O(\log N(\log N + h))$. Contando solo il contributo dovuto alla comunicazione, il tempo dell'algoritmo scende a $O(\log^2 N + h)$. Fissata una costante L, l'algoritmo richiede che ci siano inizialmente al più hN/L messaggi distribuiti tra i nodi di input (ossia i nodi del primo livello). Quindi procederà in fasi, in ciascuna delle quali tutti i messaggi vengono portati, in più iterazioni, al livello successivo. L'idea è quella di distribuire il più possibile questi messaggi tra i nodi del livello in cui giungono, riuscendo così a evitare congestione ai buffer (la cui taglia ricordiamo essere $\Theta(h)$). Per fare questo sfruttiamo una proprietà della rete che abbiamo ricavato da quella di espansione. Questa ci garantisce che se spediamo un prefissato numero di messaggi lungo ogni arco, è possibile limitare il numero di quelli che arrivano nello stesso nodo. Questa proprietà, inoltre, ci fornisce un tempo limite entro cui tutti i messaggi possono essere portati al livello successivo, in quanto assicura che a ogni iterazione una frazione costante di questi prosegue il suo cammino. La costruzione della rete e le proprietà che abbiamo utilizzato coinvolgono molti parametri, da cui dipende l'efficienza dell'algoritmo. Abbiamo quindi studiato in dettaglio questa relazione, fornendo delle possibili scelte per questi parametri, che permettono di ottenere un equilibrio tra l'efficienza dell'algoritmo e la complessità di costruzione della rete.

1.4 La struttura della tesi

Il resto della tesi è organizzato come segue. Nel Capitolo 2 descriviamo in dettaglio la multibutterfly e alcune delle sue più importanti proprietà. Nello stesso capitolo diamo un'ampia rassegna dei principali risultati, inerenti a problemi di routing, noti in letteratura, per questa rete. In particolare descriviamo l'algoritmo di Upfal [22] e quello di Arora, Leighton e Maggs [3], per quel che riguarda l'instradamento di una permutazione, e l'algoritmo di Maggs e Vöcking [13], che costituisce la prima soluzione al problema dell'h-

relation su una rete simile alla multibutterfly. L'algoritmo di Arora, Leighton e Maggs è quello che ci ha fornito gli spunti per realizzare il nostro, che viene descritto in dettaglio nel Capitolo 3. In questo capitolo, definiamo una proprietà della rete utilizzata dal nostro algoritmo; descriviamo quindi in dettaglio l'algoritmo, analizzandone la complessità in tempo. Dato che la complessità dipende da vari parametri coinvolti nella costruzione della rete e nelle sue proprietà, studiamo le scelte migliori per questi, che garantiscano prestazioni efficienti e che limitino la complessità di costruzione della rete. Nell'ultimo capitolo infine, abbiamo riassunto i risultati principali raggiunti in questo lavoro, dando delle indicazioni su come orientare la ricerca futura.

Capitolo 2

Algoritmi noti in letteratura

2.1 Introduzione

In questo capitolo descriveremo alcuni algoritmi che riguardano i problemi di routing sulla multibutterfly. Inizieremo presentando la rete ed alcune delle sue più importanti proprietà.

Passeremo quindi ad analizzare in dettaglio gli algoritmi di routing noti in letteratura. Cominceremo con l'algoritmo deterministico di Upfal [22] che risolve il problema di instradare una 1-relation su una rete di grado limitato con buffer di taglia limitata. Questo problema è noto anche come il routing di una permutazione, in quanto una 1-relation corrisponde a una funzione biiettiva tra i nodi di input e i nodi di output. Prima dell'algoritmo di Upfal, l'unico algoritmo deterministico, che risolveva il problema del routing di una permutazione, in tempo asintoticamente ottimo, su una rete di $N \log N$[1] nodi, di grado limitato, era un algoritmo che sostanzialmente riconduceva il problema di routing a un problema di ordinamento. Quest'ultimo infatti si basava sulla rete AKS [1], una rete cioè di $O(N \log N)$ nodi, di grado limitato, che contiene nella sua struttura la proprietà di espansione. Tramite questa rete è possibile ordinare N elementi in tempo $O(\log N)$. A questa rete

[1]La base del logaritmo è 2, quando non è esplicitamente indicato. Useremo invece la notazione ln per indicare il logaritmo naturale, ossia in base e.

veniva collegata opportunamente una N-butterfly che completava il routing in tempo $O(\log N)$. L'algoritmo di Upfal invece usa uno schema di calcolo completamente nuovo che, evitando l'ordinamento, gli consente di eseguire il routing sempre in tempo $O(\log N)$, ma con una costante, racchiusa nel tempo di esecuzione, molto più bassa. Dall'analisi fatta dallo stesso Upfal emerge che, mentre il tempo d'esecuzione dell'algoritmo di AKS nasconde una costante dell'ordine di qualche migliaio, quello di Upfal ha una costante dell'ordine delle centinaia. Inoltre l'algoritmo di Upfal viene eseguito su una nuova rete, riscoperta dallo stesso autore: la rete multibutterfly. Questa ha una topologia più semplice della rete AKS anche se è ugualmente potente. Recentemente infatti Maggs e Vöcking in [13] hanno dimostrato come realizzare efficientemente un embedding della rete AKS nella multibutterfly.

Successivamente presenteremo l'algoritmo di Arora, Leighton e Maggs [3] che risolve il problema della 1-relation su una multibutterfly usando buffer di $O(1)$ bit. Questo algoritmo sfrutta una particolare proprietà di espansione della rete. Il nostro algoritmo per l'instradamento di h-relation prende le mosse da questo lavoro.

Infine, nell'ultima sezione, accenneremo brevemente al primo algoritmo che risolve il problema dell'h-relation su una multibutterfly in tempo ottimo, recentemente apparso in letteratura [13].

2.2 La rete multibutterfly

La rete multibutterfly è stata introdotta per la prima volta da Upfal in [22][2]. Preferiamo però presentarla tramite la definizione data da Arora-Leighton-Maggs in [3], in quanto questa risulta essere più generale.

Introduciamo innanzitutto la rete butterfly (si veda a esempio la Figura 2.1). Una N-*butterfly* ha $2N \log N$ archi e $N(\log N + 1)$ nodi suddivisi in N righe e $\log N + 1$ livelli. Ogni nodo è identificato da una coppia di indici

[2]In realtà Upfal ha riscoperto la multibutterfly dandole il nome, ma una versione simile della rete era già nota in letteratura [4].

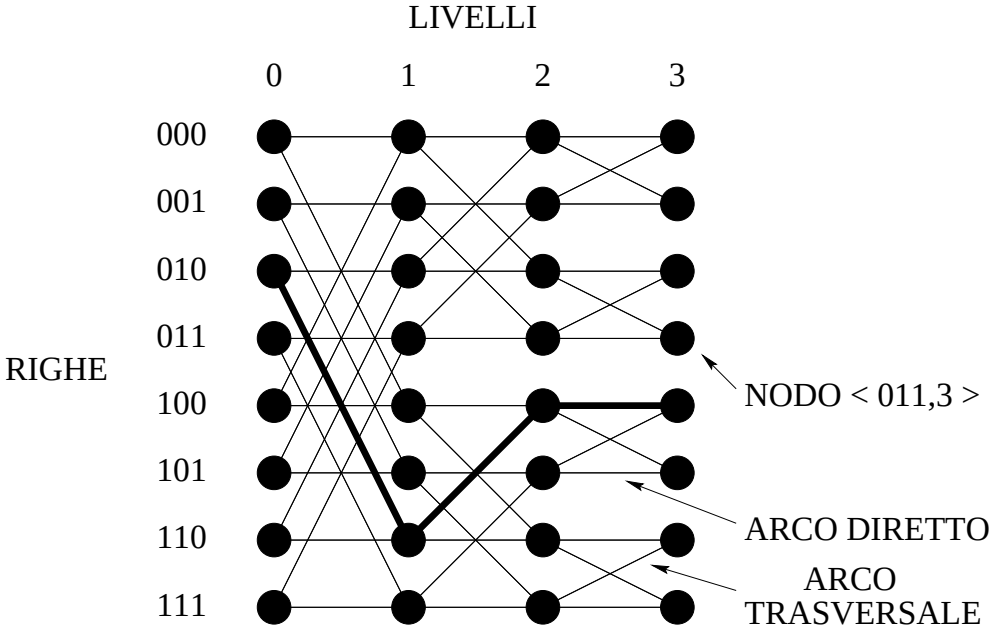

Figura 2.1: Una 8-butterfly, in cui è stato evidenziato il cammino dal nodo $< 011, 0 >$ al nodo $< 100, 3 >$.

$< w, \ell >$ dove w rappresenta la notazione binaria della *riga* a cui il nodo appartiene e ℓ indica il *livello* del nodo ($0 \leq \ell \leq \log N$). Due nodi della rete, $< w, \ell >$ e $< w', \ell' >$, sono collegati da un arco se e solo se $\ell' = \ell + 1$ e w e w' sono identici, oppure w e w' differiscono soltanto nell'ℓ'-esimo bit. Osserviamo che nel primo caso si parlerà di *archi diretti*, nel secondo di *archi trasversali*. Un'importante caratteristica di questa rete è quella che un nodo di una qualsiasi riga w del livello 0 è collegato con un nodo di una qualsiasi riga w' del livello $\log N$ con un unico cammino di lunghezza $\log N$. Questo cammino, detto anche *greedy path*, attraversa ogni livello esattamente una volta, usando l'arco trasversale dal livello ℓ al livello $\ell + 1$ se e solo se w e w' differiscono nell'$(\ell + 1)$-esimo bit.

Una *(d,N)-multibutterfly* (si veda la Figura 2.2) si ottiene sovrapponendo più butterfly in modo opportuno. In particolare, date due N-butterfly B_1 e B_2 e un insieme di permutazioni $\Pi = \langle \pi_0, \pi_1, \ldots, \pi_{\log N} \rangle$ dove $\pi_\ell : [0, \frac{N}{2^\ell} - 1] \longrightarrow [0, \frac{N}{2^\ell} - 1]$, una $(2, N)$-multibutterfly si ottiene fondendo il nodo della riga $\frac{jN}{2^\ell} + i$ del livello ℓ di B_1 con il nodo della riga $\frac{jN}{2^\ell} + \pi_\ell(i)$ del livello ℓ

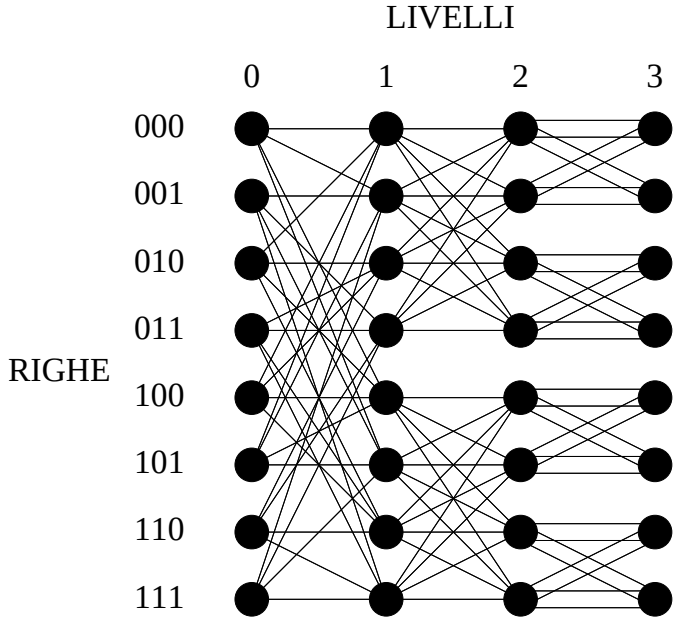

Figura 2.2: Una (2,8)-multibutterfly.

di B_2 dove $0 \leq i \leq \frac{N}{2^\ell} - 1$, $0 \leq j \leq 2^\ell - 1$ e $0 \leq \ell \leq \log N$. Il risultato è una rete con N nodi di input (quelli del primo livello), N di output (quelli dell'ultimo livello) e un totale di $(\log N + 1)$ livelli, in cui ogni nodo ha 4 archi entranti ed altrettanti uscenti. Più in generale una (d, N)-multibutterfly è composta da d butterfly sovrapposte in modo simile a prima, si assume che $d = O(1)$, ossia usando $d - 1$ insiemi di permutazioni, $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$, dove $\Pi^{(i)} = \{\pi_\ell^{(i)} \quad 0 \leq \ell \leq \log N\}$. Il risultato è una rete di $(\log N + 1)$ livelli con nodi di grado $2d$ sia entrante che uscente. Si osservi in particolare che una N-butterfly può essere vista come una $(1, N)$-multibutterfly.

Gli archi uscenti da ogni nodo possono essere divisi in due gruppi, gli archi superiori e quelli inferiori. Questa divisione risulta più chiara se definiamo la rete in termini di splitter. Più precisamente gli archi dal livello ℓ al livello ℓ+1 che partono dalle righe comprese nell'intervallo $[\frac{jN}{s^\ell}, \frac{(j+1)N}{s^\ell} - 1]$ formano un *ℓ-splitter* per ogni $0 \leq \ell \leq \log N$ e $0 \leq j \leq 2^\ell - 1$. Ognuno dei 2^ℓ splitter che partono dal livello ℓ può essere visto come un grafo bipartito con $\frac{N}{2^\ell}$ nodi di input e altrettanti nodi di output. Quest'ultimi si dividono naturalmente in

$\frac{N}{2^{\ell+1}}$ *output superiori* e $\frac{N}{2^{\ell+1}}$ *output inferiori*. Per definizione tutti gli splitter dello stesso livello ℓ sono isomorfi, e ogni input è collegato con d output superiori (tramite gli *archi superiori*) e d output inferiori (tramite gli *archi inferiori*), secondo le permutazioni $\pi_\ell^{(1)}, \ldots, \pi_\ell^{(d-1)}$.

Facciamo notare che per costruire una (d, N)-multibutterfly si richiedono $d-1$ permutazioni. Questo vuol dire che esiste almeno una butterfly a cui non vengono permutate le righe. Questa precisazione ci risulterà molto utile nell'eseguire l'algoritmo che presenteremo nel Capitolo 3; infatti ci consentirà un semplice embedding di un albero binario coi processori della rete.

Come nella butterfly, anche in questo caso è semplice determinare il cammino che un messaggio deve compiere per arrivare a un particolare nodo dell'ultimo livello partendo da un nodo del livello 0. A questo scopo dividiamo i nodi del livello ℓ, per ogni $0 \leq \ell \leq \log N$, in 2^ℓ insiemi: $A_0^\ell, \ldots, A_{2^\ell-1}^\ell$, dove $A_j^\ell = \{\langle \ell, k \rangle : \lfloor \frac{k}{2^{\log N - l}} \rfloor = j\}$. Osserviamo che i nodi dell'insieme A_j^ℓ sono collegati tramite uno splitter con i nodi degli insiemi $A_{2j}^{\ell+1}$ e $A_{2j+1}^{\ell+1}$. Supponiamo che al passo ℓ il messaggio si trovi su un nodo di input di un ℓ-splitter; questo viene spedito verso l'insieme degli output superiori dello splitter se l'ℓ-esimo bit della rappresentazione binaria della riga a cui è destinato vale 0, verso l'insieme degli output inferiori altrimenti (i bit sono numerati a partire da quello più significativo). Ovviamente la scelta dell'arco attraverso cui fare avanzare il messaggio non è univoca; infatti sono possibili molti cammini per portarlo a destinazione. L'unica cosa a cui bisogna prestare attenzione è la direzione che il messaggio assume a ogni passo, ossia se si dirige verso l'insieme degli output superiori o inferiori. Questo vuol dire che quando un messaggio transita per l'insieme A_j^ℓ, i primi ℓ bit della rappresentazione binaria della riga di destinazione devono coincidere con i primi ℓ bit della rappresentazione binaria di j. Quindi, dopo $\log N$ passi, un messaggio arriva nel singolo nodo $A_{\log N}^k$, se k è la riga del nodo di destinazione.

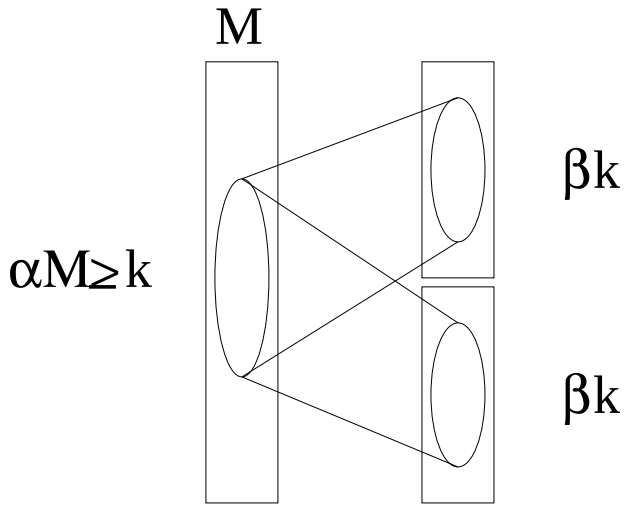

Figura 2.3: Un M-input splitter con proprietà di espansione.

2.2.1 Alcune proprietà della rete

Negli algoritmi che analizzeremo, e in particolare in quello che svilupperemo nel prossimo capitolo, sfrutteremo reti multibutterfly che godono di un'importante proprietà di espansione.

Diremo che un M-input splitter gode della proprietà di $(\alpha, \beta)-$*espansione* (si veda la Figura 2.3) se ogni insieme di $k \leq \alpha M$ input dello splitter è connesso con almeno βk output superiori e altrettanti output inferiori ($\beta > 1$). Analogamente diciamo che una multibutterfly gode della proprietà di (α, β)-espansione se ognuno degli splitter che la compongono gode della suddetta proprietà. [3]

I parametri α e β avranno un ruolo fondamentale nell'analisi del nostro algoritmo. Vediamo quindi sotto quali condizioni possiamo garantire l'esistenza di una rete con la proprietà richiesta.
In primo luogo, poiché un M-input splitter ha al più $\frac{M}{2}$ output superiori, si deve avere che $\beta\alpha M \leq \frac{M}{2}$ ossia

$$2\alpha\beta < 1,$$

Una seconda disuguaglianza, che assieme a quella precedente ci dà una con-

dizione sufficiente per l'esistenza, è

$$d > \beta + 1 + \frac{\beta + 1 + \ln 2\beta}{\ln(\frac{1}{2\alpha\beta})}$$

Tale condizione appare come Corollario 2.1 in [22]. La dimostrazione che la condizione è sufficiente a garantire l'esistenza dello splitter si può trovare in [10].

Un'altra proprietà, che segue da quella precedente (si veda a tal proposito [3]), è la proprietà di (α, δ) $r - neighbors$. Un M-input splitter gode della proprietà di $(\alpha, \delta) r - neighbors$ se in ogni sottoinsieme X di $k \leq \alpha M$ input, ci sono due sottoinsiemi X_U e X_D di X tali che $|X_U| \geq \delta k$ e $|X_D| \geq \delta k$; inoltre ogni nodo in X_U (X_D) è collegato con almeno r output superiori (output inferiori), ciascuno dei quali ha al più r nodi vicini in X.

2.3 L'algoritmo di Upfal

L'algoritmo che Upfal presenta in [22], per risolvere il problema di instradare un'arbitraria permutazione sulla multibutterfly, sfrutta in modo determinante la proprietà di (α, β)-espansione.

Il problema dell'instradamento di permutazioni equivale al problema del routing di 1-relation e possiamo formalizzarlo nel seguente modo: ogni nodo di input contiene al più un messaggio che deve essere spedito a un nodo di output, con il solo vincolo che ogni nodo di output è destinazione per al più un messaggio. Il problema è quello di far pervenire tutti i messaggi nel minor tempo possibile. Upfal presenta un algoritmo deterministico che risolve il problema suddetto in tempo $O(\log N)$.

L'idea principale dell'algoritmo è la stessa che sta alla base del routing su una butterfly. Un messaggio contenuto in un nodo, ad esempio del livello ℓ, tenta di avanzare attraverso un arco diretto verso i nodi di output superiori se l'ℓ-esimo bit della rappresentazione binaria della riga del nodo di destinazione è 0, verso quelli inferiori altrimenti. Per decidere lungo quale arco (dei d possibili) si deve muovere, Upfal divide gli N messaggi in $L = \lceil \frac{1}{\alpha(\beta+1)} \rceil$

gruppi di priorità, in modo tale che non più di $\lceil \alpha m(\beta+1) \rceil$ messaggi possano attraversare ogni m-input splitter. Per garantire questa proprietà, il k-esimo gruppo, $B(k)$, è formato dai messaggi che hanno come destinazione un nodo appartenente alla riga q, dove $q \bmod L = k$. Infatti i messaggi che passano attraverso un m-input splitter possono accedere ad al più m nodi di destinazione, quindi possiamo avere al più $\lceil m/L \rceil = \lceil \alpha m(\beta+1) \rceil$ messaggi di ogni gruppo. Inoltre un messaggio del gruppo $B(k)$ ha una priorità maggiore rispetto a tutti i messaggi contenuti in $\bigcup_{i>k} B(i)$. Gli archi di ogni splitter sono colorati con $2d$ colori, in modo tale che non ci siano due archi dello stesso colore adiacenti a un unico nodo. L'algoritmo viene quindi diviso in fasi. Nelle fasi dispari sono attivati gli archi che collegano i livelli dispari con quelli pari; in quelle pari sono attivati gli archi che collegano i livelli pari con quelli dispari. In ogni fase gli archi sono attivati uno di seguito all'altro secondo l'ordine stabilito sui colori. In questo modo ad ogni passo è attivato solo un arco per ogni nodo.

Lo scopo dell'algoritmo è quello di fare avanzare sempre i messaggi aventi priorità maggiore tra quelli contenuti nel nodo. Infatti quando è attivato un arco che connette due nodi, v e u, se il messaggio con priorità maggiore in v è del gruppo $B(k)$ (per un qualche k) e u non contiene messaggi di questo gruppo, allora il messaggio sarà spedito, da v a u, tramite l'arco. Osserviamo che questo ci garantisce che non più di L messaggi si trovino nello stesso nodo in quanto c'è al più un messaggio da ogni gruppo; questo vuol dire che l'algoritmo richiede buffer di taglia $O(L) = O(1)$ su ogni nodo.

Indichiamo con $X_s^t(k)$ il numero dei messaggi del gruppo k presenti nel livello s dopo l'iterazione t dell'algoritmo. Fissate le costanti α, β, γ e θ, in modo che valga

$$\frac{4}{\beta+1} + \frac{3\gamma^\theta}{1-\gamma^\theta} \leq \gamma^2 < 1,$$

dall'analisi di Upfal si ricava il seguente lemma:

LEMMA 1. *Sia t pari, allora dopo l'iterazione t si ha:*

$$X_s^t(k) \leq \begin{cases} \frac{N}{L}\gamma^{t-\theta k}(1+\beta)^{(s+1)/2} & s \text{ dispari;} \\ \frac{N}{L}\gamma^{t-\theta k}(1+\beta)^{s/2} & s \text{ pari.} \end{cases}$$

Da questo lemma segue il teorema:

TEOREMA 5. *Un'arbitraria permutazione può essere instradata sulla rete multibutterfly in*

$$2d\left(\frac{1+\frac{1}{2}\log(1+\beta)}{\log\frac{1}{\gamma}}\right)\log N + O(1) \in O(\log N)$$

passi, nell'ipotesi che ad ogni passo un nodo possa spedire e ricevere al più un messaggio.

Osserviamo ancora una volta che la soluzione di Upfal permette di usare meno hardware. Infatti la rete multibutterfly ha solo $\log N + 1$ livelli ciascuno con N nodi, mentre il numero dei livelli richiesti nella rete AKS è uguale al tempo di esecuzione, e anche qui ogni livello ha N nodi. Quindi, anche se il grado di ogni nodo e la taglia dei buffer dei nodi è più grande nella multibutterfly, il numero totale degli archi, e il numero totale dei buffer è minore di quello della rete AKS.

Per completezza facciamo notare che nello stesso lavoro [22] Upfal presenta anche un altro risultato molto importante: la possibilità cioè di compiere il routing di N messaggi su una rete di grado limitato, di N processori, in tempo $O(\log N)$. Anche questo risultato va confrontato con l'unico algoritmo deterministico che fino ad allora si conosceva: l'algoritmo di Columnsort di Leighton [11] che si basava sulla rete d'ordinamento AKS. In entrambi i casi si fa uso di una rete di grado limitato di N nodi e i tempi d'esecuzione sono entrambi asintoticamente ottimi. Ma l'algoritmo di Upfal permette di abbassare la costante racchiusa nel tempo di esecuzione e di usare una rete dalla topologia più semplice.

2.4 L'algoritmo di Arora-Leighton-Maggs

Nel loro lavoro [3] Arora, Leighton e Maggs presentano il primo algoritmo di circuit-switching per la multibutterfly. A differenza cioè dell'algoritmo di Upfal, descritto nella sezione 2.3, questo lavora con buffer di $O(1)$ bit,

ossia i messaggi non vengono mai immagazzinati in un nodo[3]. Anche questo sfrutta un'importante proprietà di espansione della rete: la proprietà di (α, δ) unshared neighbor. Diciamo che un M-input splitter gode della proprietà di (α, δ) *unshared neighbor* se in ogni sottoinsieme X di $k \leq \alpha M$ input, ci sono δk nodi che hanno un vicino tra gli output superiori che non è adiacente a nessun altro nodo di X. Inoltre ci sono δk nodi in X che hanno un vicino tra gli output inferiori che non è adiacente a nessun altro nodo di X. Il seguente lemma dimostra come la proprietà di (α, β)-espansione sia sufficiente per ottenere la proprietà di unshared neighbor.

LEMMA 2. *Uno splitter che gode della proprietà di (α, β)-espansione, gode anche della proprietà di (α, δ) unshared neighbor dove $\delta = 2\beta/d - 1$ e $\beta > d/2$.*

Rimarchiamo il fatto che la proprietà di espansione è una condizione sufficiente ma non neccessaria per la proprietà di unshared neighbor.

Diamo ora una descrizione dettagliata dell'algoritmo, in quanto quello che presenteremo nel prossimo capitolo è una generalizzazione di questo al caso delle h-relation. Assumiamo che i cammini siano stabiliti solo tra nodi di input e di output che si trovano, rispettivamente, sulle colonne 0 e $\log N$, e su righe congrue a 0 mod L, dove L è una potenza di 2 e $L \geq 1/\alpha$. In questo modo evitiamo che un numero eccessivo di messaggi transiti per lo stesso splitter. Cominciamo col descrivere un algoritmo on-line che instradi un insieme di messaggi su una rete multibutterfly, che gode della proprietà di espansione, in tempo $O(\log^2 N)$. Vedremo successivamente come migliorarlo per ottenere il tempo, asintoticamente ottimo, $O(\log N)$.

Inizialmente abbiamo al più N/L messaggi distribuiti sui nodi di input che appartengono alle righe congrue a 0 mod L. Ogni nodo ha al più un messaggio. L'algoritmo è diviso in fasi, in ognuna delle quali il fronte dei messaggi avanza di un livello. Soffermiamoci quindi ad analizzare la fase ℓ, nella quale i messaggi, giunti al livello ℓ, saranno portati al livello $\ell + 1$. L'idea è quella di fare avanzare solo i messaggi diretti verso nodi che hanno

[3]Si veda nella Sezione 1.1 la differenza tra i due modelli di routing: ossia il circuit-switching e il store-and-forward.

un solo vicino. Infatti in questo caso possiamo estendere i cammini senza preoccuparci di bloccare altri messaggi che stanno tentando di avanzare al livello successivo. La fase consiste in un ciclo in cui ogni singola iterazione è divisa in tre aparti.

Parte 1. Ogni cammino che deve essere esteso invia una "proposta" ad ogni nodo di output (cioè del livello $\ell + 1$) a cui è collegato nella direzione desiderata.

Parte 2. Ogni nodo di output che riceve esattamente una proposta invia di ritorno la sua "accettazione".

Parte 3. Ogni cammino che riceve una "accettazione" avanza verso uno dei nodi del livello $\ell + 1$ che gli hanno inviato l'"accettazione".

Si osservi che se ci poniamo nel *bit model*, cioè nel modello in cui ogni nodo può ricevere e spedire lungo ogni arco un solo bit in tempo $O(1)$, ogni singola parte può essere implementata in un numero costante di passi. In particolare nella Parte 3 verranno inviati i restanti bit della riga di destinazione a partire dal bit $(\ell+2)$-esimo, in quanto questo è sufficiente per consentire alla fase successiva di iniziare. Poiché gli splitter che collegano il livello ℓ con il livello $\ell + 1$ hanno $M = N/2^\ell$ input, per la scelta degli output della rete si ha che al più M/L cammini possono transitare attraverso questi input. Essendo inoltre $L \geq 1/\alpha$, possiamo dire che ci sono al più αM cammini che attendono di essere estesi in ogni splitter. In base alla proprietà di unshared neighbor, una frazione costante di questi viene estesa a ogni iterazione, e quindi, in un numero finito di iterazioni, tutti i messaggi saranno spediti al livello successivo. In particolare vale il seguente lemma:

LEMMA 3. *Durante la fase ℓ tutti i cammini verranno estesi dal livello ℓ al livello $\ell + 1$ in $\log(N/L2^\ell)/\log(1/(1-\delta))$ iterazioni.*

Da questo lemma segue immediatamente il seguente teorema:

TEOREMA 6. *L'algoritmo descritto consente di estendere tutti i cammini dal livello 0 al livello* $\log N$ *in*

$$\sum_{\ell=0}^{\log n-1} \frac{\log \frac{N}{L2^\ell}}{\log \frac{1}{1-\delta}} \in O(\log^2 N)$$

passi.

È possibile costruire i cammini richiesti anche in tempo $O(\log N)$, modificando il precedente algoritmo. L'idea è quella di non attendere $\Theta(\log N/2^\ell)$ passi affinché tutti i cammini del livello ℓ siano estesi, ma di procedere con la fase successiva non appena qualche messaggio giunga al livello $\ell + 1$. Ad esempio dopo la prima iterazione i cammini che devono ancora essere estesi si riducono a una frazione costante pari a $1-\delta$ del numero originale. Il pericolo è che questi si trovino bloccati nel momento in cui avanzano di un livello. Onde evitare questo, i cammini che non vengono estesi inviano un segnale di "prenotazione" a tutti i nodi vicini che appartengono al livello successivo. Quest'ultimi parteciperanno, da questo punto in poi, alla costruzione dei cammini come se possedessero realmente dei messaggi. Ovviamente questi nodi dovranno estendere il cammino in entrambe le direzioni, in quanto non sanno qual è il messaggio che alla fine ne usufruirà. In altri termini i messaggi che non vengono estesi subito si prenotano un cammino attraverso i livelli successivi, che percorreranno non appena è possibile.

Ogni cammino che non viene esteso invia $2d$ prenotazioni al livello successivo. È quindi facile vedere come in poche iterazioni i nodi prenotati possano aumentare al punto da intasare il sistema. Per evitare questo, ogni volta che un cammino viene esteso invia un segnale di cancellazione a tutti i nodi che aveva precedentemente prenotato, in quanto non sono più necessari per quel particolare cammino. Quindi se un nodo prenotato riceve un segnale di cancellazione da tutti i nodi che in precedenza l'avevano prenotato, smette di cercare un cammino inviando a sua volta un segnale di cancellazione a tutti i nodi che aveva eventualmente prenotato. L'analisi degli autori dimostra che questo accorgimento è sufficiente a prevenire crescite eccessive del numero di prenotazioni.

Anche questo algoritmo viene suddiviso in fasi, durante le quali ogni messaggio può avanzare di al più un livello. Più precisamente definiamo come *fronte dei cammini* i primi cammini, reali o prenotati, che arrivano ad un dato livello. Durante ogni fase il fronte dei cammini avanza di un livello. Ogni fase consiste in tre parti:

Parte 1. C passi durante i quali vengono inviati i segnali di cancellazione. Questi segnali ad ogni passo attraversano un livello della rete.

Parte 2. T passi durante i quali vengono estesi i cammini di un livello. Durante questa parte il numero dei cammini, in ogni splitter, ancora da estendere diminuisce di almeno un fattore $(1 - \delta)^T$.

Parte 3. Un singolo passo riservato alle prenotazioni; ossia tutti i cammini appartenenti al fronte, che non sono stati estesi nei T precedenti passi, inviano una prenotazione a tutti i vicini del livello successivo.

Anche in questo caso possiamo osservare che nel bit model, supponendo C e T costanti, ogni singola parte può essere eseguita in tempo $O(1)$. L'analisi degli autori dimostra che i cammini bloccati diminuiscono di una frazione costante ad ogni iterazione; questo consente di dimostrare il seguente teorema:

TEOREMA 7. *Con un'opportuna scelta dei parametri e delle costanti C e T, l'algoritmo descritto consente di estendere tutti i cammini sulla multibutterfly in $O(\log N)$ passi.*

2.5 L'algoritmo di Maggs-Vöcking

In [13] viene presentato, il primo algoritmo deterministico per risolvere il routing di una h-relation su una multibutterfly che gode della proprietà di (α, β)-espansione. Un'assunzione fondamentale su cui poggia l'algoritmo, è quella che alla rete vengono aggiunti $h - 1$ livelli numerati con $-(h - 1), \ldots, -1$, che permettono di distribuire meglio i messaggi. Questi livelli sono connessi tra loro tramite degli (α, β)-*expander*, ossia dei grafi bipartiti

$G = (X, Y)$, tali che $|X| = |Y| = n$ e per ogni $A \subseteq X$ tale che $|A| \leq \alpha n$ si ha $|\Gamma(A)| > \beta|A|$ (Dove con $\Gamma(A)$ indichiamo i nodi di Y che sono connessi con i nodi di A). L'idea è quella di decomporre l'h-relation in h permutazioni disgiunte, che possono essere risolte in pipeline tramite l'algoritmo di Upfal che abbiamo visto nella Sezione 2.3. Il problema principale rimane quindi quello di dividere i messaggi di ogni nodo in gruppi, in modo tale che al più αm messaggi di ogni gruppo transitino attraverso un m-input splitter. Ricordiamo che questa era un'ipotesi fondamentale dell'algoritmo di Upfal. Per fare questo gli autori suddividono sostanzialmente l'algoritmo in due fasi. Individuano come prima cosa delle sottomultibutterfly che costituiscono gli ultimi livelli della rete. Quindi nella prima fase i messaggi sono suddivisi in gruppi, a seconda della riga a cui appartiene il nodo di destinazione, e tramite l'algoritmo di Upfal vengono spediti alle sottomultibutterfly opportune. I messaggi saranno immagazzinati lungo tutta la riga a cui appartiene il nodo a cui erano destinati. Ad esempio tutti i messaggi destinati al nodo (v, ρ), dove ρ indica il livello in cui si trovano i nodi di input delle sottomultibutterfly individuate in precedenza, vengono spediti in un qualsiasi nodo della riga v in modo tale che ogni nodo contenga al massimo un numero costante di messaggi. Terminata questa fase, i messaggi con la stessa destinazione vengono assegnati a indici distinti tramite i quali vengono ripartiti nuovamente in gruppi. Quindi nella seconda fase viene rieseguito l'algoritmo di Upfal, tenendo presente la nuova suddivisione in gruppi, che consente di portare i messaggi alle loro destinazioni finali.

TEOREMA 8. *L'algoritmo descritto consente il routing di una h-relation su una multibutterfly, con buffer di taglia costante, in tempo $O(\log N + h)$.*

Rimarchiamo il fatto che l'algoritmo usa una rete di $N(h + \log N)$ nodi con buffer di taglia costante. Nel prossimo capitolo descriveremo un algoritmo alternativo, per l'instradamento di un'h-relation, su una rete con soli $N(\log N + 1)$ nodi ma con buffer di taglia $O(h)$.

Capitolo 3

Routing di una h-relation su una multibutterfly

3.1 Introduzione

In questo capitolo presentiamo un algoritmo deterministico per risolvere il routing di una h-relation su una multibutterfly che gode della proprietà di (α, β)-espansione. Detto V_i l'insieme dei nodi di input e V_o l'insieme dei nodi di output, una h-relation consiste in un insieme di coppie formate da nodi di input e nodi di output: $R \subseteq V_i \times V_o$, in modo tale che ogni nodo in V_i e ogni nodo in V_o appare in al più h coppie di R. Ogni coppia (v_i, v_o) rappresenta un messaggio che deve essere spedito dal nodo di input v_i al nodo di output v_o.

Fino ad ora si conoscono per la multibutterfly solo algoritmi per risolvere il routing di una permutazione (o 1-relation) in tempo $O(\log N)$. (Si vedano ad esempio i lavori di Upfal [22] o di Arora-Leighton-Maggs [3]). Recentemente è stato pubblicato un algoritmo efficiente per il routing di una h-relation su una multibutterfly. Maggs e Vöcking [13], sfruttando l'algoritmo di Upfal, hanno trovato un metodo per risolvere il problema in tempo $O(\log N + h)$. Come osservato nel capitolo 2, il loro risultato richiede di aggiungere alla rete $h-1$ colonne di N processori che gli permettono di distribuire più uniformemente

i vari messaggi[1].

Vedremo ora come risolvere il problema senza l'utilizzo di colonne aggiuntive, ottenendo un tempo di esecuzione $O(\log N(\log N + h))$ se prendiamo in considerazione sia il tempo di computazione che quello di comunicazione. Solitamente però si considera solo il tempo di comunicazione, in quanto questo è il più "costoso" tra i due. In questo caso il nostro algoritmo richiede tempo $O(\log^2 N + h)$, che è ottimo per valori di $h = \Omega(\log^2 N)$.

3.2 Una nuova proprietà per la rete

Definiamo una nuova proprietà simile alla $(\alpha, \delta)\, r - neighbors$, che abbiamo visto nel capitolo precedente, ma più restrittiva.

DEFINIZIONE 1. *Un M-input splitter gode della proprietà di $(\alpha, \delta, r, s) - neighbor$ se in ogni sottoinsieme X di $k \leq \alpha M$ input, ci sono due sottoinsiemi X_U e X_D di X tali che $|X_U| \geq \delta k$ e $|X_D| \geq \delta k$; inoltre ogni nodo in X_U (X_D) è collegato con almeno r output superiori (output inferiori) ciascuno dei quali ha al più s vicini in X.*

Il seguente lemma dimostra che un M-input splitter con sufficiente espansione gode anche della proprietà di $(\alpha, \delta, r, s) - neighbor$:

LEMMA 4. *Un M-input splitter con $(\alpha, \beta) - espansione$, gode anche della proprietà di $(\alpha, \delta, r, s) - neighbor$, con*

$$\delta \geq \frac{\frac{s+1}{s}\beta - \frac{d}{s} - r + 1}{d - r + 1}$$

Dimostrazione. Sia X un sottoinsieme di $k \leq \alpha M$ input in un M-input splitter. Sia n_s il numero di output superiori che hanno almeno uno ma al più s vicini in X e n_+ il numero degli output superiori che hanno più di s vicini in X. Ora $n_s + n_+ \geq \beta k$ (per la proprietà di $(\alpha, \beta) - espansione$) e $n_s + (s+1)n_+ \leq dk$. Dalle due disuguaglianze si ottiene che $n_s \geq (\frac{s+1}{s}\beta - \frac{d}{s})k$.

[1]Una descrizione dettagliata degli algoritmi citati si può trovare nel Capitolo 2.

Sia ora δk il numero dei nodi di X che hanno almeno r output superiori vicini, ciascuno dei quali ha al più s vicini in X; allora $\delta kd + (1-\delta)k(r-1) \geq n_s$ ossia

$$\delta \geq \frac{\frac{s+1}{s}\beta - \frac{d}{s} - r + 1}{d - r + 1}$$

□

3.3 Costruzione di un r-bundle

In questa sezione faremo uso della seguente notazione.
Sia G un M-input splitter e indichiamo con u un nodo di input, con v un nodo di output, con S un sottoinsieme dei nodi di input di G e con Q un sottoinsieme dei nodi di output di G.

- $\Gamma(u)$ indica l'insieme dei nodi di output collegati ad u, $\Gamma(S)$ l'insieme $\cup_{u \in S}\Gamma(u)$.
- $E(u)$ (risp. $E(v)$) indica l'insieme degli archi adiacenti ad u (risp. v), mentre $E(S) = \cup_{u \in S}E(u)$ (risp. $E(Q) = \cup_{v \in Q}E(v)$).
- Sia $E \subseteq E(S)$. E è detto un *r-bundle* per S se $|E \cap E(u)| = r$ per ogni $u \in S$. La quantità $\max_{v \in V} |E \cap E(v)|$ è detta *congestione* di E.

Dato quindi un M-input splitter G che gode della proprietà di $(\alpha, \delta, r, s) -$ *neighbor* e detto S un sottoinsieme dei suoi nodi di input tale che $|S| \leq \alpha M$, un r-bundle T per S di congestione al più s può essere costruito con la seguente procedura, ottenuta modificando quella data in [6]:

PROC1
T:=$\emptyset$; $R := S$;
while $R \neq \emptyset$ **do**
$Q := \{v \in \Gamma(R) \ : \ |E(R) \cap E(v)| > s\}$;
si marchino tutti gli archi in $E(R) \cap E(\Gamma(R) \setminus Q)$;
per ogni $u \in R$ si aggiungano r archi marcati a T (se esistono)
$R = R \setminus \{u \in R \ : \ |E(u) \cap T| = r\}$
end.

LEMMA 5. *La procedura PROC1 costruisce un r-bundle T per S di congestione al più s in $\frac{\log|S|}{\log\frac{1}{1-\delta}}$ iterazioni.*

Dimostrazione. In base alla proprietà di $(\alpha, \delta, r, s) - neighbor$, nella prima iterazione esistono almeno $\delta|S|$ nodi che hanno almeno r vicini che non stanno in Q; questi nodi ovviamente saranno selezionati e tolti da R. A ogni iterazione quindi la cardinalità di R diminuisce di un fattore δ; ossia in $\log_{\frac{1}{1-\delta}}|S|$ iterazioni $|R| < 1$, ossia $R = \emptyset$. La procedura terminerà sicuramente in al più $\frac{\log|S|}{\log\frac{1}{1-\delta}}$ iterazioni.
Analizziamo ora la congestione. Definiamo R_i il valore di R dopo l'i-esima iterazione e analogamente Q_i il valore di Q dopo l'i-esima iterazione. Poiché $R_j \subset R_i$ per ogni $j > i$ allora possiamo dire che il numero dei vicini considerati di un nodo di output non può aumentare nel corso dell'algoritmo. Per $j > i$ si ha che $Q_j \subset Q_i$ e quindi $(\Gamma(R_i) \setminus Q_i) \cap Q_j = \emptyset$. Questo vuol dire che dall'istante in cui vengono marcati gli archi di un nodo di output v, fino a che questo non esce da $\Gamma(R_j)$ per un qualche j, il nodo v avrà sempre degli archi marcati. Inoltre, per quanto detto prima, il numero di questi non può aumentare nel corso dell'algoritmo (in quanto l'insieme R diminuisce). Non solo, possiamo anche dire che a ogni iterazione a ogni nodo vengono marcati soltanto una parte (o al limite tutti) degli archi che gli erano stati marcati nell'iterazione precedente. Ossia non vengono marcati archi diversi da quelli della prima marcatura. Quindi il massimo numero di archi entranti in un nodo di output che vengono marcati lo si ha la prima volta. Ma questo è, ovviamente, al più s.

□

Consideriamo ancora una volta un sottoinsieme S dei nodi di input di G. Assegniamo a ogni nodo $u \in S$ un peso intero $w(u)$. Distribuiamo questo peso equamente sugli archi, assegnando a ogni arco uscente da u un peso $w(u)/r$. Sia T un r-bundle per S. Posto W_v^T uguale alla somma dei pesi degli archi contenuti in $T \cap E(v)$, definiamo il *peso* di T come il $\max_{v \in V} W_v^T$.

Sia $h = \max_{u \in S} w(u)$; dividiamo i nodi di S in $\log h + 1$ famiglie così definite:

$$F_i^S = \left\{ u \in S \quad t.c. \quad \frac{h}{2^{i+1}} < w(u) \leq \frac{h}{2^i} \right\} \quad \text{per} \quad 0 \leq i \leq \log h$$

Posto $M = \mid S \mid$, $Z = \sum_{u \in S} w(u)$ e $x_i = \lceil \mid F_i^S \mid / \alpha M \rceil$ il seguente algoritmo calcolerà un r-bundle T per S di peso minimo.

Algoritmo r-BUNDLE

Per ogni i partiziona F_i^S in gruppi $F_i^S(1), \ldots, F_i^S(x_i)$, dove $|F_i^S(j)| \leq \alpha M$;
Esegui PROC1, in parallelo, su ogni $F_i^S(j)$ per produrre un r-bundle $T_i(j)$;
$T := \bigcup_{i=0}^{\log h} \bigcup_{j=1}^{x_i} T_i(j)$;
end

LEMMA 6. *L'algoritmo r-BUNDLE calcola un r-bundle T per S di peso strettamente minore di*

$$\frac{2sZ}{r\alpha M} + 2h\frac{s}{r} + s(\frac{1}{\alpha} + 1 + \log h)$$

Dimostrazione. Poniamo per semplicità $\epsilon = s/r$.
Sappiamo che la famiglia F_i^S è composta di x_i gruppi; $x_i - 1$ di questi saranno completi, ossia in totale ci sono almeno $\alpha M(x_i - 1)$ nodi per la suddetta famiglia. In tutto i nodi devono essere al più M, quindi:

$$\sum_{i=0}^{\log h} \alpha M(x_i - 1) \leq M,$$

ossia

$$\sum_{i=0}^{\log h} x_i \leq \frac{1}{\alpha} + 1 + \log h$$

I nodi della famiglia F_i^S hanno peso almeno pari a $\frac{h}{2^{i+1}}$; quindi, tenendo presente che il peso totale è al più Z, si ha:

$$\sum_{i=0}^{\log h} \alpha M(x_i - 1)\frac{h}{2^{i+1}} \leq Z$$

ossia

$$\sum_{i=0}^{\log h} \frac{x_i - 1}{2^i} \leq \frac{2Z}{Mh\alpha}$$

Ogni gruppo di F_i^S contribuirà per un singolo nodo con un peso pari ad al più $\lceil h/(2^i r) \rceil s$. In totale quindi ogni nodo può ricevere al più un peso uguale a:

$$\begin{aligned}\sum_{i=0}^{\log h} \left\lceil \frac{h}{2^i r} \right\rceil s x_i &\leq \sum_{i=0}^{\log h} \epsilon x_i \frac{h}{2^i} + \sum_{i=0}^{\log h} s x_i = h\epsilon \sum_{i=0}^{\log h} \frac{x_i - 1}{2^i} + h\epsilon \sum_{i=0}^{\log h} \frac{1}{2^i} + \epsilon r \sum_{i=0}^{\log h} x_i \\ &< h\epsilon \left(\frac{2Z}{Mh\alpha} + \sum_{i=0}^{\infty} \frac{1}{2^i} \right) + \epsilon r \left(\frac{1}{\alpha} + 1 + \log h \right) \\ &\leq h\epsilon \left(\frac{2Z}{Mh\alpha} + 2 \right) + \epsilon r \left(\frac{1}{\alpha} + 1 + \log h \right)\end{aligned}$$

□

Osserviamo infine che con una opportuna scelta dei parametri e del valore di h si può fare in modo che $T \in \Theta(h)$.

3.4 L'algoritmo

Diamo ora la descrizione di un algoritmo on-line per instradare una h-relation su una (d, N)-multibutterfly. Si assuma che le destinazioni dei messaggi siano su nodi di output che appartengono a righe congrue a 0 mod L, ove L è una potenza del 2 e $L \geq 1/\alpha$. Questo ci assicurerà una maggiore distribuzione dei cammini. Cercheremo di sfruttare gli strumenti sviluppati nella prima parte del capitolo.

L'algoritmo sarà suddiviso in *fasi*, dove in ciascuna fase il fronte dei messaggi avanza di un livello. In particolare nella fase ℓ i messaggi sono portati dal livello ℓ al livello $\ell + 1$. Per l'assunzione fatta sugli output risulta che all'inizio si hanno al più hN/L messaggi distribuiti tra i nodi di input del livello 0. Infine si richiede che ogni nodo di input contenga al più h messaggi. Soffermiamoci quindi sulla generica fase ℓ.

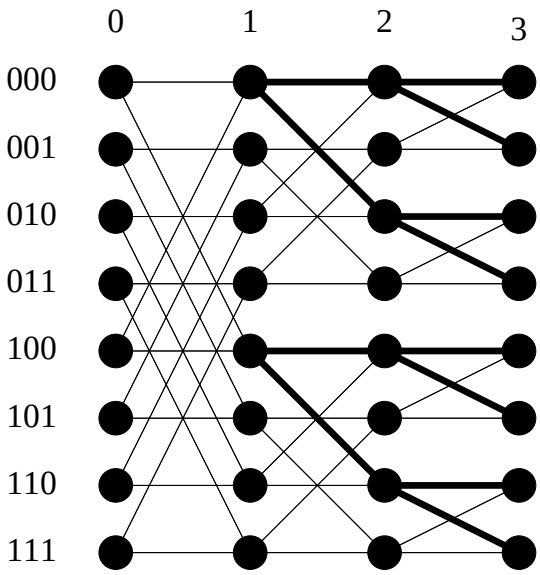

Figura 3.1: Una 8-butterfly in cui sono stati evidenziati degli alberi binari aventi la radice nel livello 1.

Il livello ℓ è suddiviso in 2^ℓ ℓ-splitter aventi $M = N/2^\ell$ nodi di input. Da ogni splitter si possono raggiungere al più M/L nodi finali di output (nodi cioè destinatari di messaggi). Essendo $L \geq 1/\alpha$ ci sono al più $hM/L \leq \alpha hM$ messaggi distribuiti tra i nodi di input all'inizio della fase ℓ. Assumiamo che ogni nodo della rete abbia dei buffer di taglia $2^k h$, per un qualche $k \in O(1)$: ossia che vi siano al più $2^k h$ messaggi per nodo. Verificheremo la validità di tale assunzione nell'analisi. Dei nodi finali di output metà sono raggiungibili tramite i nodi di output superiori dello splitter, e metà tramite i nodi di output inferiori; quindi vi sono al più $(\alpha hM)/2$ messaggi destinati ai nodi di output superiori e $(\alpha hM)/2$ a quelli inferiori. Durante la fase ℓ ogni ℓ-splitter lavora in parallelo, compiendo le stesse azioni sia per quel che riguarda i messaggi destinati agli output superiori dello splitter, sia per quelli destinati agli output inferiori. Prenderemo quindi in considerazione un generico splitter e, per tale splitter, solo gli archi diretti verso nodi di output superiori, con la convenzione che i vari passi dovranno essere ripetuti per gli archi diretti verso i nodi di output inferiori.
Dividiamo la fase ℓ in passi di cui forniamo ora una descrizione dettagliata.

Passo 1. Il primo passo consiste nel dividere i nodi del livello ℓ in famiglie e nel dividere ogni famiglia in gruppi di al più αM nodi. Sia $w(u)$ il numero dei messaggi contenuti nel nodo u; le famiglie sono così definite:

$$F_i = \left\{ u \ : \ \frac{h}{2^{i+1}} \leq w(u) \leq \frac{h}{2^i} \right\}, \quad \text{per} \quad -k \leq i \leq \log h$$

La suddivisione in famiglie risulta molto semplice; infatti ogni nodo conoscendo il numero dei messaggi che contiene può determinare facilmente l'indice della famiglia a cui appartiene. A questo punto ogni famiglia F_i deve essere suddivisa in gruppi di al più αM nodi. Questo può essere fatto sfruttando una caratteristica della multibutterfly evidenziata durante la descrizione della rete nella Sezione 2.2. Sappiamo che a una delle d butterfly usate per la costruzione della multibutterfly non sono state permutate le righe. Risulta quindi semplice, per ogni splitter, individuare un albero binario avente le foglie nell'ultimo livello della multibutterfly e la radice tra i nodi di input dello splitter (si veda a tal proposito la Figura 3.1). Possiamo quindi eseguire in pipeline il calcolo di $\log h + k + 1$ somme prefisse, una per ogni famiglia F_i di nodi. In particolare i nodi di ogni famiglia, a cominciare da quelli della famiglia F_{-k}, invieranno la loro informazione (ossia 0 se non contengono messaggi relativi alla famiglia che si sta considerando, 1 altrimenti) lungo la loro riga di appartenenza, ai nodi dell'ultimo livello. Tramite gli alberi binari, a cui si accennava prima, possiamo compiere una somma prefissa per ogni famiglia in tempo $O(\log M)$ (vedi ad esempio [9]). I valori così ottenuti verranno rispediti, sempre lungo le righe, ai nodi del livello ℓ.

Passo 2. Il secondo passo è un ciclo in cui a ogni iterazione vengono eseguiti i seguenti sottopassi:

a) Ogni processore che contiene dei messaggi invia una "proposta" a tutti i nodi di output a lui collegati (nodi cioè appartenenti al livello $\ell + 1$). La "proposta" consiste in una coppia di indici: il

primo indica la famiglia, il secondo il gruppo a cui il processore appartiene.

b) Ogni processore del livello $\ell+1$ risponde con una "accettazione" lungo tutti quegli archi che provengono da gruppi che hanno inviato un numero di proposte inferiori o uguali ad s.

Ogni processore riceve al più $2d$ proposte (tante quante gli archi entranti). Per poter gestire questo sottopasso il processore può ordinare alla prima iterazione le varie proposte in ordine lessicografico (ricordiamo che una proposta è una coppia di indici) e rispondere a quelle che sono in numero minori o uguali ad s. Per le iterazioni successive basta tener traccia di questo ordine, in quanto non possono giungere proposte diverse da quelle ricevute alla prima iterazione, e controllare solo quali sono quelle in numero minori o uguali ad s.

c) Ogni processore del livello ℓ che riceve esattamente r accettazioni risponde a tutte inviando $\lceil m/r \rceil$ bit (se m è il numero totale di messaggi contenuti nel processore) lungo ogni arco da cui proviene un' accettazione. Ovviamente lungo l'ultimo arco verranno spediti $m - \sum_{i=1}^{r-1} \lceil \frac{m}{r} \rceil$ bit. In particolare per ogni messaggio il processore invia l'$(\ell+2)$-esimo bit della rappresentazione binaria della riga a cui è destinato (sempre con la convenzione che il primo bit è il più significativo), in modo da permettere all'algoritmo di continuare senza aspettare l'esecuzione del terzo passo.

Passo 3. Quest'ultimo passo viene eseguito mentre il livello successivo ha cominciato la nuova fase. In questo passo i nodi inviano effettivamente i messaggi, ovviamente lungo gli stessi archi attraverso i quali al passo 2(c) avevano inviato i bit. Onde consentire all'algoritmo di proseguire, a ogni iterazione viene inviato un bit di ogni messaggio a cominciare dal bit in cui siamo arrivati, ossia il bit $(\ell+3)$-esimo. Terminati i bit che riguardano le righe di destinazione (che a questo punto sono $\log N - \ell$)

vengono inviati i bit dei messaggi veri e propri. Nel calcolo del tempo d'esecuzione questo passo contribuirà solo nell'ultima fase per il tempo neccessario per spedire i messaggi veri e propri. Infatti nell'ultima fase non dobbiamo spedire alcun bit riguardante la riga di destinazione, ma ci rimangono i bit dei messaggi.

3.4.1 Analisi dell'algoritmo

Per quanto riguarda l'analisi dell'algoritmo, vediamo come prima cosa un lemma che ci permetterà di dimostrarne la correttezza.

LEMMA 7. *Sia k un intero arbitrario. Se $\epsilon < 2^k/(2^{k+1}+1)$ e $h > c$ dove*

$$c \in \Theta\left(a\left(\frac{1}{\alpha} + k + \log\left(2\left[a\left(\frac{1}{\alpha}+k\right)\right]^2\right)\right)\right),$$

in cui si è posto per comodità $a = s/(2^k(1-2\epsilon)-\epsilon)$, allora al più $2^k h$ messaggi sono presenti in un nodo nel corso dell'algoritmo.

Dimostrazione. Osserviamo che il secondo passo dell'algoritmo usa la stessa tecnica che avevamo usato nell'Algoritmo r-BUNDLE visto nella Sezione 3.3 per la costruzione di un r-bundle. L'unica differenza riguarda il numero delle famiglie che vengono prese in considerazione; ossia quello visto nella sezione 3.3 è un caso particolare in cui si richiede che i nodi abbiano buffer di taglia h (in altri termini si è posto $k = 0$). Possiamo quindi ripercorrere la dimostrazione del Lemma 6, ponendo $Z = \alpha M h/2$ e tenendo conto del maggiore numero di famiglie. Si ottiene che il peso dell'r-bundle (e quindi la congestione dei messagi in un nodo) sarà strettamente minore di $h\epsilon(2^{k+1}+1) + \epsilon r(\log h + 1/\alpha + k)$. Questo valore sarà al più $2^k h$ se

$$h \geq \frac{\epsilon r \log h + \frac{\epsilon r}{\alpha} + \epsilon r k}{2^k(1-2\epsilon)-\epsilon} = \frac{s\left(\log h + \frac{1}{\alpha} + k\right)}{2^k(1-2\epsilon)-\epsilon}$$

Si vede facilmente che esiste una costante $c \in \Theta(a(1/\alpha + k + \log(2[a(1/\alpha + k)]^2)$ (con $a = s/(2^k(1-2\epsilon)-\epsilon)$) tale che se $h \geq c$ la relazione precedente è verificata, non appena $2^k(1-2\epsilon)-\epsilon > 0$ ossia $\epsilon < 2^k/(2^{k+1}+1)$.

□

Se ora applichiamo, come esempio, il Lemma 7 con $k = 0$, si ottiene che, se $\epsilon < 1/3$ e $h > c$ ove $c = \Theta(s(1/\alpha + \log(2(s/(\alpha(1-3\epsilon)))^2))/(1-3\epsilon))$, la taglia dei buffer sarà al più h. Il Lemma 7 ci permette di applicare iterativamente l'algoritmo, in quanto ci assicura che all'inizio di ogni fase i nodi di input abbiamo un numero di messaggi pari ad al più $2^k h$. Più in generale il Lemma 7 ci dice che con un'opportuna scelta dei parametri il nostro algoritmo è corretto, nel senso che i nodi non ricevono mai un numero di messaggi superiore alla taglia dei buffer che contengono.

Analizziamo ora la complessità in termini di tempo. Per questa analisi useremo il seguente modello: supponiamo che in tempo unitario possano essere eseguite $O(1)$ operazioni elementari per nodo e possano essere trasferite su ogni link, parole di $O(\log N)$ bit (a tal proposito si veda il *word-model* in [9]). Calcoliamo quindi il tempo necessario per la generica fase ℓ.

LEMMA 8. *Supponiamo che x sia il numero massimo di bit in ogni messaggio. Se i buffer della rete hanno taglia h, la generica fase ℓ richiede un tempo di comunicazione pari a:*

$$O\left(\log M + \log h + \frac{1}{\log \frac{1}{1-\delta}} \log \alpha M + \frac{h}{\log N} + \frac{(x + \log N - \ell)h}{\log N}\right)$$

Dimostrazione. Per dimostrare il lemma analizzeremo i passi uno per volta. Il primo passo consiste di $\log h + 1$ somme prefisse compiute su alberi binari di altezza $\log M$; quindi il suo costo è pari a $(4 \log M + \log h)$. Si veda ad esempio [9]. Il secondo passo consiste di $1/(\log(1/(1-\delta))) \log \alpha M$ iterazioni in cui i sottopassi (a) e (b) richiedono tempo $O(2d \log 2d) = O(1)$. Il sottopasso (c), invece, richiede un tempo complessivo pari a $O(h/\log N)$ in quanto, durante questo passo, vengono inviati $m \in \Theta(h)$ bit, ossia uno per ogni messaggio contenuto nel nodo. Nell'ultimo passo infine, verranno spediti i messaggi veri e propri, oltre alla restante parte dell'indirizzo (ossia $\log N - \ell$ bit). Poiché sono neccessarie $h/\log N$ iterazioni per recapitare un bit di ogni messaggio, e per ipotesi la massima lunghezza di un messaggio è x, questo passo richiederà un tempo pari a $O((x + \log N - \ell)h/\log N)$.

□

A questo punto possiamo ricavare facilmente il seguente teorema:

TEOREMA 9. *La complessità dell'algoritmo, eseguito su una multibutterfly con buffer di taglia h, dovuta alla sola comunicazione, è pari a:*

$$O(\log^2 N + h + \frac{xh}{\log N})$$

Dimostrazione. Per spedire i messaggi a destinazione sono neccessarie $\log N$ fasi. Osserviamo però che il passo 3 di ogni fase viene eseguito contemporaneamente alla fase successiva; questo vuol dire che il suo costo contribuirà solo nell'ultima fase e solo per i bit che riguardano i messaggi (e non le destinazioni). Dal Lemma 8 possiamo dire che la complessità totale, dovuta alla sola comunicazione, per spedire tutti i messaggi a destinazione, è pari a:

$$\sum_{\ell=0}^{\log N} O\left(\log \frac{N}{2^\ell} + \log h + \frac{1}{\log \frac{1}{1-\delta}} \log \alpha \frac{N}{2^\ell} + \frac{h}{\log N}\right) + \frac{xh}{\log N},$$

ossia

$$O\left(\log^2 N + h + \frac{xh}{\log N}\right)$$

□

Si noti che questo tempo è ottimo non appena $h \in \Omega(\log^2 N)$.

Osservazione. In realtà possiamo sfruttare l'algoritmo descritto per portare i messaggi sino al livello $\log N/L$, ossia compiere soltanto $\log N/L$ fasi, e poi instradare i messaggi senza preoccuparci di problemi di congestione. Infatti dal livello $\log N/L$ in poi gli splitter che compongono la rete contengono nei nodi di input un numero totale di messaggi minore o uguale ad h. Quindi basterà instradare i messaggi lungo un qualsiasi arco diretto verso i nodi di output superiori se il bit a cui siamo arrivati è 0, verso quelli inferiori altrimenti.

Vediamo ora di calcolare il tempo di routing globale, incorporando cioè anche il contributo dovuto alla computazione interna ai nodi.

TEOREMA 10. *L'algoritmo descritto nella Sezione 3.4, può instradare una h-relation su una multibutterfly con buffer di taglia h in*

$$O(\log^2 N + h \log N + \frac{xh}{\log N})$$

passi, tenendo conto sia del tempo di comunicazione, sia del tempo di computazione locale.

Dimostrazione. L'algoritmo prevede che in un nodo possano esserci $O(h)$ messaggi. Questo vuol dire che una qualsiasi operazione, sui singoli messaggi, da parte di un nodo richiede tempo $O(h)$ di computazione locale. Ad esempio nel passo 1 ogni nodo deve dividere gli $m \in \Theta(h)$ messaggi in due gruppi, quelli diretti verso la parte superiore della rete e quelli diretti verso quella inferiore. Questa operazione è semplice in quanto basta controllare il valore di un bit, ma, essendoci m messaggi, richiede $m \in O(h)$ passi. Quindi, tenendo presente il Teorema 9, il tempo totale per eseguire il nostro algoritmo è pari a:

$$O(\log^2 N + h \log N + \frac{xh}{\log N})$$

□

Abbiamo preferito tenere distinte la complessità dovuta alla sola comunicazione da quella totale, perché generalmente la prima è considerata, in algoritmi di questo tipo, più "costosa".

3.4.2 Analisi e scelta dei parametri

Nell'algoritmo si è sfruttato in modo fondamentale la proprietà di (α, δ, r, s)-neighbor, che sappiamo essere legata alla proprietà di (α, β)-espansione.
Quindi i vari parametri coinvolti nella nostra analisi devono soddisfare alcune disuguaglianze che qui riassumeremo:

$$2\alpha\beta < 1$$

$$d > \beta + 1 + \frac{\beta + 1 + \ln 2\beta}{\ln(\frac{1}{2\alpha\beta})}$$

$$\delta \geq \frac{\frac{s+1}{s}\beta - \frac{d}{s} - r + 1}{d - r + 1}.$$

Inoltre devono ovviamente valere le seguenti restrizioni:

- $0 < \alpha < 1$
- $1 < \beta < d$
- $s \leq r \leq d$
- $0 < \delta < 1$

Per comodità esprimiamo alcuni parametri in funzione di d (che ricordiamo essere il numero degli archi uscenti dagli input di uno splitter verso gli output superiori/inferiori):

$$r = \theta d \quad \beta = \gamma d \quad s = \epsilon r = \epsilon\theta d \qquad \text{dove } \theta, \epsilon \text{ sono } \leq 1 \quad \text{e} \quad \gamma < 1$$

Analizziamo per prima cosa la proprietà di espansione;

$$d > \gamma d + 1 + \frac{\gamma d + 1 + \ln 2\gamma d}{\ln \frac{1}{2\alpha\gamma d}}$$

Ora poiché $2\alpha\beta < 1$ allora $\ln \frac{1}{2\alpha\gamma d} > 0$ e quindi si ricava che

$$\alpha < \frac{1}{2\gamma d} e^{\frac{\gamma d + 1 + \ln 2\gamma d}{\gamma d - d + 1}}$$

Per quanto riguarda i parametri coinvolti nella proprietà di (α, δ, r, s)-neighbor vogliamo che ci garantiscano un $\delta > 0$, ossia

$$\frac{\frac{s+1}{s}\beta - \frac{d}{s} - r + 1}{d - r + 1} > 0.$$

Poiché $d > r - 1$ possiamo scrivere

$$\frac{s+1}{s}\beta - \frac{d}{s} - r + 1 > 0$$

risolvendo rispetto ad r troviamo

$$r^2 - (\beta + 1)r + \frac{d - \beta}{\epsilon} < 0$$

Essendo $(d - \beta)/\epsilon > 0$ e $-(\beta + 1) < 0$, l'equazione associata alla precedente disuguaglianza ammette due soluzioni reali e positive non appena

$$(\beta + 1)^2 - 4(\frac{d - \beta}{\epsilon}) > 0,$$

ossia

$$\gamma^2 d^2 + \left(-\frac{4}{\epsilon} + 2\gamma + \frac{4\gamma}{\epsilon}\right) d + 1 > 0$$

e quindi

$$d > \frac{\frac{2}{\epsilon} - \gamma - \frac{2\gamma}{\epsilon} + \sqrt{(\frac{2}{\epsilon} - \gamma - \frac{2\gamma}{\epsilon})^2 - \gamma^2}}{\gamma^2}$$

Osserviamo che questa disuguaglianza è ben definita solo se il radicando è non negativo, ossia solo se

$$0 < \gamma \leq \frac{1}{1+\epsilon}$$

In questo caso il valore di r è compreso nell'intervallo

$$\frac{\beta + 1 - \sqrt{(\beta+1)^2 - 4(\frac{d-\beta}{\epsilon})}}{2} < r < \frac{\beta + 1 + \sqrt{(\beta+1)^2 - 4(\frac{d-\beta}{\epsilon})}}{2}$$

Per finire riassumiamo in una tabella dei possibili valori per i parametri usati nell'analisi dell'algoritmo. Nelle ultime tre colonne abbiamo riportato i limiti inferiori per la taglia dei buffer nel caso in cui questi possano contenere rispettivamente h, $2h$ o $4h$ messaggi.

									taglia dei buffer		
ϵ	γ	d	β	α	r	s	δ	$\frac{1}{\log \frac{1}{1-\delta}}$	$k=0$	$k=2$	$k=4$
2/5	1/2	18	9	1/91	5	2	1/28	19	/	/	2235
2/5	4/9	27	12	1/77	5	2	1/46	32	/	/	1788
1/3	1/2	22	11	1/100	6	2	1/34	23	/	1445	947
1/3	4/9	36	16	1/94	6	2	1/31	21	/	1371	897
1/3	4/9	36	16	1/94	9	3	1/21	14	/	2078	1360
3/10	1/2	28	14	1/115	10	3	1/57	39	4185	1630	1240
3/10	4/9	36	16	1/94	10	3	1/81	56	3537	1371	1039
1/4	1/2	30	15	1/121	8	2	1/46	32	1134	745	630
1/4	4/9	45	20	1/112	8	2	1/76	52	1065	699	590
1/5	1/2	38	19	1/142	10	2	1/38	26	809	641	583

Abbiamo fornito molte possibile scelte dei parametri, in quanto questa scelta dipende da ciò che vogliamo ottenere e dai dati a nostra disposizione. Ad

esempio per avere d basso e α alto deve essere alta la taglia dei buffer. Al contrario con una rete un po' più complicata, ossia con d alto e α basso, possiamo avere dei buffer più ridotti, cioè dell'ordine di h (si veda a questo proposito l'ultima riga della tabella). Se invece siamo più interessati alla costante racchiusa nel tempo di esecuzione, dobbiamo considerare la colonna che riporta il valore $1/(\log(1/(1-\delta)))$, in quanto questo influisce sul numero di iterazioni neccessarie per costruire l'r-bundle al passo 2. Risulta conveniente usare valori alti per ϵ, in quanto questo ci consente di avere una rete più semplice e un tempo di esecuzione abbastanza buono. Ma questo è possibile solo se ammettiamo dei buffer di taglia più grande di h. Dai valori delle ultime colonne si deduce, che nel caso in cui h sia basso e si abbiano a disposizione buffer sufficientemente grandi, conviene far girare l'algoritmo con un valore di k abbastanza alto, cioè aumentare il numero delle famiglie.

Capitolo 4

Conclusioni

Tra le architetture parallele conosciute, una delle più potenti è apparsa la rete multibutterfly. È già stato studiato per questa rete il problema del routing di una permutazione, ossia il problema di scambiare messaggi e dati tra i processori nel minor tempo possibile. In letteratura infatti possiamo trovare due algoritmi che risolvono questo problema in tempo asintoticamente ottimo. Ben poco invece si sa riguardo al problema delle h-relation, un problema che ha acquistato molta importanza soprattutto negli ultimi anni con la nascita di nuovi modelli di computazione. Tra questi modelli ricordiamo il BSP, in quanto è uno dei migliori candidati ad assumere il ruolo di modello universale per il calcolo parallelo. Tale modello utilizza come primitiva di comunicazione l'h-relation. Questo ci ha portato a studiare anche per la multibutterfly possibili strategie che consentono di risolvere efficientemente questo problema. In particolare in questa tesi abbiamo formulato un algoritmo deterministico che permette di instradare una h-relation su una multibutterfly in tempo $O(\log N(\log N+h))$, usando dei buffer di taglia $\Theta(h)$. L'unica altra soluzione che si conosce per questo problema, è opera di Maggs e Vöcking, ed è apparsa di recente in letteratura [13]. Questa seconda soluzione ottiene un tempo di esecuzione asintoticamente ottimo, pari cioè a $O(\log N + h)$. Per riuscire a raggiungere queste prestazioni, gli autori aggiungono alla rete multibutterfly h livelli di N nodi; ottenendo così una rete di $N(\log N + h)$ nodi anche se

buffer costanti. Visto che il limite inferiore per questo tipo di problema, sulla rete multibutterfly, è pari a $O(\log N + h)$, un problema ancora aperto è quello di riuscire a ottenere le stesse prestazioni (ottime) dell'algoritmo di Maggs e Vöcking con la nostra rete. Una ottimizzazione del nostro algoritmo potrebbe procedere in due tempi. In primo luogo, bisognerebbe limitare il numero di computazioni interne fatte da un singolo processore in ogni passo, portandole a un numero costante. Ad esempio si potrebbero combinare i messaggi aventi la stessa destinazione e permettere a ogni nodo di contenere quelli destinati a un numero limitato di destinazioni. Questo potrebbe permettere di ottenere una complessità pari a $O(\log^2 N + h)$. Ulteriori miglioramenti si possono ottenere, in un secondo tempo, usando una tecnica molto simile a quella che Arora, Leighton e Maggs impiegano per ottimizzare il loro algoritmo. Questa tecnica, come abbiamo visto nella Sezione 2.4, usa "segnali di prenotazione" che consentono ai messaggi, che in quel momento non hanno la possibilità di proseguire, di prenotarsi un cammino che seguiranno non appena è possibile. Per quanto riguarda i buffer, un modo per limitarli è quello di usare la strategia di Maggs e Vöcking, ossia di scomporre l'h-relation in h permutazioni disgiunte e di instradarle in pipeline una di seguito all'altra. Si potrebbe provare anche a instradare una alla volta i messaggi contenuti nei nodi, cioè risolvere il routing di N messaggi senza pretendere che questi abbiano destinazioni distinte, richiedendo solo che ci siano al massimo h messaggi diretti verso lo stesso nodo.

A nostro avviso comunque, a livello pratico il nostro algoritmo dovrebbe risultare più efficiente di quello che l'analisi lascia supporre. Ossia i valori che abbiamo richiesto, ad esempio, per la taglia dei buffer, sono difficilmente raggiungibili, se non con particolari problemi studiati appositamente.

4.1 Ringraziamenti

Vorrei ringraziare tutti coloro che, in qualche modo, hanno contribuito per la buona riuscita di questo lavoro. Tra tutti gli amici e i compagni di corso, un ringraziamento particolare va a

Paciacio, per i suoi consigli tecnici in fase di scrittura,

e a Lucia, per la sua infinita pazienza nel correggere e ricorreggere le varie stesure che si sono susseguite.

Bibliografia

[1] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. *15th Symp. on Theory of Computing*, 1983.

[2] R. Aleliunas. Randomized parallel communication. In *Proc. ACM Symp on Priciples of Distributed Computing*, pages 60–72, 1982.

[3] S. Arora, F. T. Leighton, and B. M. Maggs. On-line algorithms for path selection in a non-blocking network. *SIAM Journal on Computing*, 25(3):600–625, June 1996.

[4] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Trasmission*, 9:64–66, 1974.

[5] K. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Comp. Conf*, pages 307–314, 1968.

[6] A. Bäumker, W. Dittrich, and A. Pietracaprina. The deterministic complexity of parallel multisearch. In *Proc. ot the 5th Scandinavian Workshop on Algorithm Theory*, pages 404–415, 1996.

[7] V. E. Beneš. Mathematical theory of connecting networks and telephone traffic. Academic Press, New York, 1965.

[8] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. Bsp vs logp. In *Proc. of the 8th ACM Symp. on Parallel Algorithms and Architectures*, pages 25–32, 1996.

[9] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[10] F. T. Leighton, C. E. Leiserson, and D. Kravets. *Theory of parallel and VLSI computation.* Research Seminar Series Report MIT/LCS/RSS 8, Laboratory for Computer Science, Massachusette Institute of Technology, Cambridge, MA, 1990.

[11] T. Leighton. Tight bounds on the complexity of parallel sorting. *16th Symp. on Theory of Computing*, pages 71–80, 1984.

[12] G. Lev, N.J. Pippenger, and L.G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. Comput.*, 30(2):93–100, 1981.

[13] B.M. Maggs and B. Vöcking. Improved routing and sorting on multibutterflies. Technical report, CMU-CS-96-192, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, November 1996.

[14] D. Nassimi and S. Sahni. Parallel algorithms to set-up the beneš permutation network. *IEEE Trans. Comput.*, 31(2):148–154, 1982.

[15] N. Pippenger. Comunication networks. In *Handbook of Theoretical Computer Science*, volume A, pages 805–833. J. van Leeuwen, North-Holland, Amsterdam, 1990.

[16] N. Pippenger and A.C.-C. Yao. Rearrangeable networks with limited depth. *SIAM J. Algebraic Discrete Methods*, 3:411–417, 1982.

[17] A.G. Ranade. Constrained randomization for parallel communication. Technical report, YALEU/DCS/TR-511 Dept. Comput. Sci. Yale Univerity, New Haven, CT, 1987.

[18] R.Cypher and G.Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *22nd ACM Symposium on Theory of Computing*, pages 193–203, 1990.

[19] J. T. Schwartz. Ultracomputers. In *ACM TOPLAS 2*, pages 484–521, 1980.

[20] C.E. Shannon. Memory requirements in a telephone exchange. *Bell Systems Tech. J.*, 29:343–349, 1950.

[21] E. Upfal. Efficient schemes for parallel communication. *J. ACM*, 31(3):507–517, 1984.

[22] E. Upfal. An $O(\log n)$ deterministic packet routing scheme. *Proceedings of the 21st Annual ACM Symphosium on Theory of Computing*, pages 241–250, May 1989.

[23] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[24] L.G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science*, volume A, pages 943–971. J. van Leeuwen, North-Holland, Amsterdam, 1990.

[25] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *Proc. 13th Ann. ACM Symp. on Theory of Computing*, pages 263–277, 1981.

Indice