ARCHITETTURE PARALLELE

Corso di "Sistemi per Elaborazione dell'Informazione"
Prof. Carpentieri Bruno
A.A. 2004/2005

Celentano Carla
Iannaccone Lucia Carmela

Introduzione

- L'idea alla base di una architettura parallela e' collegare piu' processori per ottenere elevate prestazioni.
- I computer paralleli si possono classificare considerando:
  1) Organizzazione della memoria
  2) Organizzazione del processore
  3) Numero di flussi istruzione supportate

Introduzione

- Parlando di parallelismo, dobbiamo considerare due grandi famiglie:
  - famiglia di sistemi multiprocessore
    - Un multiprocessore è un computer parallelo che ha due o più processori.
  - famiglia di sistemi multicomputer
    - Tali sistemi sono collegati tramite una infrastruttura di rete.

(1) Organizzazione della memoria

- Nel caso di sistemi multiprocessori:
  - la gestione del sistema viene affidata alla memoria
  - Il meccanismo di gestione è di carattere fortemente hardware
- Un multiprocessore può essere:
  - Fortemente accoppiato
  - Debolmente accoppiato

Multiprocessore fortemente accoppiato

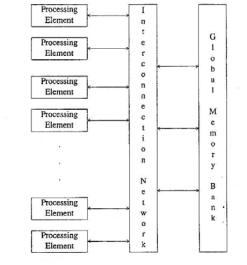


Figure 1.1 Tightly coupled multiprocessor

- I processori condividono:
  - Una memoria fisica
  - Uno spazio di indirizzamento comune

Multiprocessore debolmente accoppiato e multicomputer

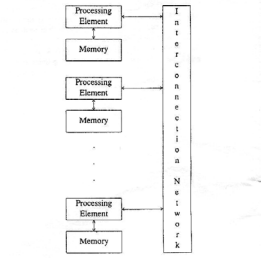


Figure 1.2 Block diagram for a loosely coupled multiprocessor and a multicomputer

- Ogni processore ha una memoria locale.
- Queste memorie locali, insieme, formano lo spazio d'indirizzamento condiviso del computer.

Differenza tra multicomputer e multiprocessore

- La differenza significativa tra un multicomputer e un multiprocessore è che un multicomputer non ha alcuna memoria condivisa né memoria condivisa per lo spazio d'indirizzamento.
- Di conseguenza per usare dati in una memoria remota, è necessario trasferire esplicitamente i dati in memoria locale.
- Queste e altre comunicazioni tra processori sono fatte attraverso scambi di messaggi (attraverso la rete) tra di essi.

7

(2) Organizzazione del processore

- L'organizzazione del processore è definita dall'interconnessione di rete utilizzata per connettere i processori al multicomputer. Tra le più comuni ci sono:
  - Mesh a due dimensioni
  - Anello
  - Albero
  - Ipercubo

8

Tipi di interconnessione

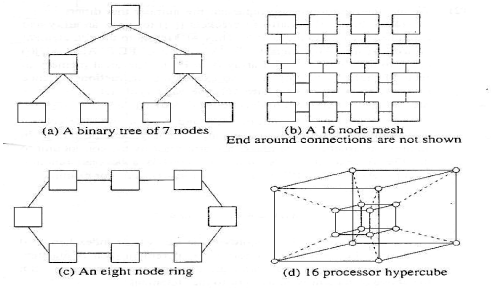


Figure 1.3 Different types of interconnection networks

9

Organizzazione del processore
Ipercubo

- In un ipercubo di dimensione d, ci sono 2^d processori.
- Assumiamo che siano etichettati con 0,1,….. 2^d-1. Due processori i e j sono direttamente connessi sse la rappresentazione binaria di i e j differisce esattamente in un bit.
- Ogni arco rappresenta una connessione diretta.
- In un ipercubo di dimensione d, ogni processore è connesso ad altri d processori.

10

Organizzazione del processore
Ipercubo

- Se la connessione diretta tra due processori i e j è unidirezionale, in un determinato istante un messaggio può fluire da i a j, o da j a i.
- Se la connessione è bidirezionale , è possibile che i manda un messaggio a j, e contemporaneamente j ne manda uno a i.

11

Organizzazione del processore
Ipercubo

- La popolarità di una rete a ipercubo può essere attribuita ai seguenti fattori:
  - Usando d connessioni per processore, possono essere connessi 2^d processori di modo che la distanza massima tra ogni coppia di processori sia d
  - Molte altre reti sono facilmente mappate in un ipercubo
  - Un ipercubo è completamente simmetrico. Inoltre un ipercubo può essere decomposto in sottoipercubi (ipercubi di dimensione più piccola)
    - Questa proprietà permette l'implementazione di algoritmi ricorsivi divide-and-conquer su un ipercubo.

12

Reti mappate in un ipercubo

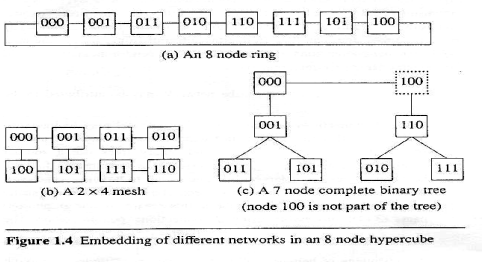


Figure 1.4 Embedding of different networks in an 8 node hypercube

(3) Flussi di istruzioni

- Flynn ha classificato le architetture dei calcolatori basandosi sui flussi di istruzioni e dati. Le due rilevanti sono:
  - ***SIMD*** *(single instruction multiple* data): tutti i processori eseguono la stessa istruzione, ma su dati diversi (fortemente sincrona)
  - ***MIMD*** *(multiple instruction multiple data):* processori separati eseguono differenti istruzioni allo stesso tempo, quindi ciascuno ha il proprio clock e lavora in modo indipendente (generalmente asincrona)

Diagramma a blocchi di un multicomputer SIMD e MIMD

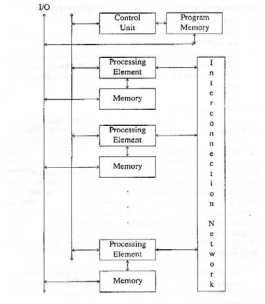


Figure 1.5 Block diagram of an SIMD multicomputer

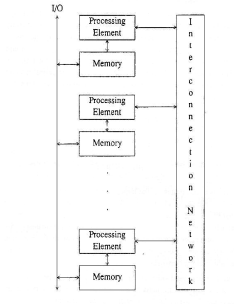


Figure 1.6 Block diagram of an MIMD multicomputer

Flussi di istruzioni

- Le caratteristiche importanti di un ipercubo con architettura SIMD e le notazioni che useremo sono:
  - Ci sono $P=2^p$ elementi di elaborazione (PE). Ogni PE ha un indice nel range [0, p-1]. Una rete a ipercubo p-dimensionale connette $P=2^p$ PE
  - Sia $i_{p-1}i_{p-2}\ldots i_0$ la rappresentazione binaria del PE con indice i. Una rete a ipercubo connette direttamente due processori i cui indici differiscono esattamente in un bit. (Il processore $i_{p-1}i_{p-2}\ldots i_0$ è connesso ai processori $i_{p-1}i_{p-2}\ldots \overline{i_k}\ldots i_0$, $0\le k\le p-1$, dove $\overline{i_K}$ è il complemento del bit i_k
  - Usiamo la notazione $i^{(b)}$ per rappresentare il numero che differisce esattamente di *b* bit da i.
  - Useremo le parentesi quadre per indicizzare un array ([])
    - A[i] si riferisce all'i-esimo elemento dell'array A
  - e le parentesi tonde per indicizzare i PE (( ))
    - A(i) si riferisce al registro A del PE i.

Flussi di istruzioni

- La memoria locale di ogni elemento di elaborazione contiene solo dati, quindi gli elementi di elaborazione devono compiere solo le operazioni aritmetiche di base.
- La struttura classica (Fig. 1.5) è composta da una unità di controllo (UC), una memoria programma, e da un array di elementi di elaborazione (PE).
  - La memoria contiene il programma che deve essere eseguito. L'unità di controllo ha il compito di prelevare le istruzioni dalla memoria e di separare le istruzioni scalari da quelle vettoriali.
    - Quelle scalari sono eseguite direttamente dalla UC mentre quelle vettoriali sono inviate a tutti i PE dell'array in parallelo.

Flussi di istruzioni

- L'assegnamento tra processori è denotato con il simbolo "←", mentre gli assegnamenti all'interno dei processori sono denotati con il simbolo ":="
- In un singolo instradamento, un' unità di dati può essere trasmessa tra una coppia di processori che sono direttamente connessi.
- Ogni processore può inviare e/o ricevere solo un' unità di dati in un singolo instradamento.
  - Se i link sono unidirezionali i dati possono essere trasmessi solo in un'unica direzione
  - Se i link sono bidirezionali i dati possono essere trasmessi in entrambe le direzioni in un singolo instradamento

Flussi di istruzioni

- Le assunzioni per i multicomputer MIMD (Fig. 1.6) differiscono dai SIMD per i seguenti aspetti:
  - Non c'è un'unità di controllo e una memoria programma separata.
  - La memoria locale di ogni PE contiene sia dati che il programma che il PE deve eseguire
  - In certo istante, diversi PE possono eseguire differenti istruzioni. In particolare in un ipercubo MIMD, il PE i può trasferire dati al PE $i^{(b)}$, mentre il PE j trasmette dati al PE $j^{(a)}$, i≠j e a≠b.

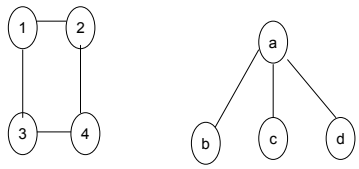
19

Embedding in un Ipercubo (1)

Una serie di processori interconnessi può essere modellata come un grafo indiretto, nel quale ciascun vertice denota un unico processore e fra due vertici c'è un arco se i due processori sono direttamente connessi.

20

Embedding in un Ipercupo (2)



G H

21

Embedding in un Ipercupo (3)

G ed H sono due grafi indiretti che modellano due serie di processori interconnessi.

- Denotiamo: V(G) e V(H) vertici in G e H.

Un ***embedding*** (Incastramento) di G in H è un mapping dei vertici di G nei vertici di H e degli archi di G nel cammino di H.

Ciascun vertice di G è mappato in un vertice distinto di H, ed inoltre deve esistere |V(H)|>= |V(G)|.

22

Embedding in un Ipercupo (4)

Nell'esempio precedente:

(1 → a,2→ b, 3→ c, 4→d) è un mapping di vertici

((1,2)→ab, (2,4) →bad, (3,4)→cad, (1,3) →ac) mapping degli archi nel path.

Entrambe formano un embedding di G in H.

23

Embedding in un Ipercupo (5)

Alcune definizioni:

- Espansione: Rapporto fra |V(H)|/ |V(G)|.
- Dilazione: Lunghezza del più lungo cammino in cui ogni arco di G è mappato.

Nell'esempio:

Espansione: 1
Dilazione: 2

24

Chains and Rings (1)

G è un anello con 2^d vertici.
Assumiamo tali vertici siano numerati da 0 a 2^d -1 .
G può essere "embedded" in H_d usando uno schema Gray.
Tale codice è definito nella maniera seguente:

$S_1 = 0,1$, $S_{k+1} = 0[S_k]^r, 1[S_k]^r$, $k>1$

Dove:

- $0[S_k]$ è ottenuto prefissando ciascun ingresso di S_k con 0
- $1[S_k]^r$ è ottenuto prima di tutto invertendo l'ordine delle entry di S_k e poi prefissando ciascuna entry con 1.

Chains and Rings (2)

Due entry adiacenti in S_k differiscono in esattamente 1 bit.

Ad esempio tramite gray troviamo:
S_3 = 000, 001, 011, 010, 110, 111, 101,100

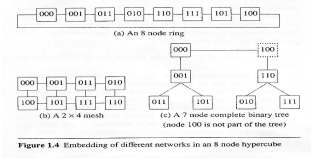


Figure 1.4 Embedding of different networks in an 8 node hypercube

Chains and Rings (3)

Gray(i,k) è l'i-esima entry in S_k, dove $0<i< 2^k$
Possiamo usare la seguente embedding di G in H_d:

- Il vertice i di G è mappato nel vertice gray(i,d) di H $0<i< 2^d$
- Ciascun arco (i,j) di G è mappato in un unico arco in H_d che connette i vertici gray (i,d) e gay (j,d)

Meshes (1)

La embedding ad anello può essere generalizzata per ottenere un PXQ mesh, quando P e Q sono entrambi potenze di 2.

Dato :

- $p = 2^p$
- $q = 2^q$

Possiamo incastrare il mesh in H_d dove d=p+q.
Il vertice in posizione (i,j) del mesh è mappato al vertice di un ipercubo con la rappresentazione binaria gray(i,p)gray(j,q)

Meshes (2)

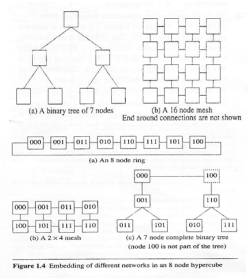
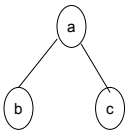


Figure 1.4 Embedding of different networks in an 8 node hypercube

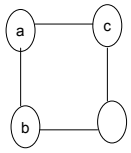
La figura 1.4 (b) mostra il mapping risultante per un mesh 2x4.
In questo caso p=1 e q= 2.
Per ciascun arco nel mesh ((i,j), (k,l)) c'è un unico arco in H_d che connette i vertici gray(i,p)gray(j,d) e gray(k,p)gray(l,q) in H_d.

Alberi binari pieni (1)

T_i albero binario di altezza i.
Poiché T_i ha 2^i -1 vertici, il meglio che potremmo fare è incastrare T_i in H_i, i >1.



T_2



T_2 embedded in H_2

Dilazione 1, embedding T_2 in H_2

Alberi binari pieni (2)

Teorema

Non c'è Dilazione 1 embedding T_i in H_i per i>2

Dim.

Supponiamo che valga l'inverso. Poichè H_i è simmetrico, possiamo assumere che la radice di T_i sia mappata al vertice 0 di H_i. I suoi figli devono essere mappati ai vertici dell'ipercubo che hanno un 1 nella loro rappresentazione binaria. Il livello tre sarà mappato ai vertici dell'ipercubo con entrambi zero o due 1. Il livello quattro deve essere mappato con vertici di uno o tre 1; e cosi via.

Alberi binari pieni (3)

Dunque, la dilazione deve soddisfare :

- Vertici di Ti che sono ad un livello dispari sono mappati in vertici dell'ipercubo che hanno un numero pari di1
- Vertici di Ti che sono ad un livello pari sono mappati in vertici dell'ipercubo che hanno un numero dispari di1.

Alberi binari pieni (4)

- Il numero dei vertici dell'ipercubo con un numero pari di 1 è uguale al numero di vertici dell'ipercubo con numero dispari di 1.
- H_i ha 2^{i-1} vertici in ciascuna categoria
- Se i è pari : H_i non può avere abbastanza vertici con un numero dispari di 1 per accomodare i vertici di T_i che sono su un livello pari.
- Se i è dispari: H_i non può avere abbastanza vertici con un numero pari di 1 per accomodare i vertici di T_i che sono su un livello dispari.

Dunque…
Non ci può essere dilazione 1 embedding T_i in H_i per i>2,al contrario ci può essere per H_{i+1} i>2 con dilazione 1 e in H_i, i>2 con dilazione 2.

Misure di performance (1)

Le performance di algoritmi uniprocessori è tipicamente misurata dal loro spazio e tempo richiesto.

Queste misurazioni sono anche usate per algortmi multicomputer.
Andremo ora a definire altre misurazioni altrettanto usate.

- Denotiamo con **Tp** e **Sp** rispettivamente il tempo e lo spazio richiesto su un nodo p multicomputer.

Misure di performance (2)

- S_p sarà l'ammontare totale di memoria richiesta da un nodo p multicomputer.

Altre misure potrebbero anche essere le quantità di speedup(accellerazione) e l' efficienza.

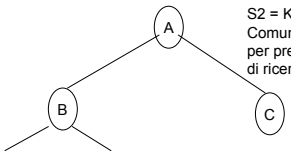
- Sia T_0 il tempo richiesto per risolvere un dato problema su un singolo nodo usando un algoritmo uniprocessore, dunque, l' accellerazione **Sp**, usando un processore p sarà: **$Sp= T0/Tp$.**

Misure di performance (3)

L'efficienza E_p sarà: **$E_p = S_p/p$.**

Problema: Cercare un nodo con le caratteristiche di C.

L'algoritmo uniprocessore esplorerebbe prima B e poi C, mentre un algoritmo parallelo esplorerebbe i sottoalberi di B e quelli di C in parallelo. In questo caso, $T_2 = 2$ (esamina A e C), mentre $T_0 = k$, dove k-1 è il numero di nodi nel sottoalbero B.



S2 = K/2 e E2 = K/4
Comunque non c'e' un metodo per predire una buona strategia di ricerca.

Misure di performance (4)

Una interessante proprietà sugli algoritmi paralleli è l'ammontare con il quale il problema della dimensione deve incrementare quando il numero di processori aumenta in modo da mantenere una certa efficienza.

Nasce il concetto di **isoefficienza** (ie(p)) di un algoritmo parallelo → di quanto il lavoro deve aumentare per mantenere una certa efficienza.

37

Misure di performance (5)

- L'isoefficienza ci consente di testare programmi paralleli usando un piccolo numero di processori e di predire le performance per un gran numero di essi.
- E'possibile sviluppare programmi paralleli su un piccolo ipercubo .
- Attraverso analisi di performance ed isoefficienza si ottiene una buona stima degli assolvimenti del programma in un target commerciale dove i multicomputer hanno molti processori e le istanze del problema sono grandi.

38