Algoritmo per la soluzione di un sistema lineare di n equazioni in n incognite su un sistema parallelo

di Giuseppe Cardinale Ciccotti

Da questo numero la rubrica viene divisa in due sezioni: una dedicata al progetto del simulatore digitale di circuiti elettronici che continua a completarsi e ci serve come pretesto per illustrare i problemi della programmazione parallela che si possono incontrare nei casi pratici e le soluzioni che si possono adottare. La nuova sezione invece non è riferita ad un argomento specifico, quanto piuttosto ad argomenti di carattere vario che si esauriscono nel corso di un solo articolo. Potremo così valutare problemi ed algoritmi più generali della programmazione parallela che non siano necessariamente legati all'Occam e ai Transputer, come pure daremo notizie di nuovi dispositivi hardware, tool, sistemi di sviluppo e commerciali che il mercato inizia a mettere a disposizione.
Questa volta affrontiamo un problema un po' più teorico, la risoluzione di un sistema di n equazioni lineari in n incognite su un sistema parallelo; come vedremo la soluzione è alquanto inusuale ma adatta all'esecuzione su un sistema parallelo

Un sistema di equazioni lineari, nel quale ognuna di esse è appunto rappresentata da una relazione di primo grado che lega le variabili, può essere risolto in maniera determinata soltanto se il numero delle incognite è in numero uguale a quello delle equazioni.

Un sistema del genere come stabiliscono ben noti teoremi di algebra può essere risolto calcolando gli n rapporti tra il determinante della matrice dei coefficienti sostituendo la colonna dei termini noti a quella della variabile in soluzione, e il determinante della matrice dei coefficienti. In definitiva è necessario calcolare n+1 determinanti di matrici nxn più n divisioni.

Esistono numerosi algoritmi anche assai efficienti per il calcolo dei determinanti delle matrici e dei sistemi lineari in genere, come molti lettori sicuramente sapranno. Anche nel campo del parallel processing sono state proposte molteplici soluzioni a questo problema proprio perché è un problema classico, molto frequente nelle applicazioni e che impegna massivamente le risorse di calcolo.

In questo nostro appuntamento vogliamo invece seguire un approccio quantomeno poco ortodosso, tuttavia assai efficiente e soprattutto semplice.

Supponiamo di avere a disposizione un computer parallelo con un numero di processori tutti uguali, pari al numero delle equazioni e quindi delle incognite del nostro sistema, l'obiettivo è naturalmente quello di trovare la soluzione cioè gli n valori che soddisfano il sistema nella maniera più veloce possibile. Una considerazione a grandi linee ci lascia giustamente intuire che la cosa migliore è dividere le operazioni fra gli n processori in modo che ognuno esegua lo stesso numero di operazioni degli altri. Con quest'approccio sarebbe possibile raggiungere un miglioramento di prestazioni pari a T/n, dove T è il tempo che un computer seriale impiegherebbe ad eseguire il medesimo algoritmo implementato sull'architettura parallela. In realtà, come vedremo anche nel nostro caso, non è possibile raggiungere questo speed-up teorico se non in situazioni e su architetture molto particolari, perciò esso rimane un limite superiore al quale tendere ed usare come riferimento.

Tali considerazioni applicate al nostro sistema ci inducono ad una suddivisione naturale del problema: assegnare ad ogni processore la ricerca della soluzione di una sola equazione. Ogni processore esegue così lo stesso algoritmo con lo stesso numero di operazioni e di conseguenza con una presumibile buona efficienza.

Le equazioni sono però interdipendenti e ciò si rifletterà sull'algoritmo parallelo che dovrà prevedere un certo grado di comunicazioni fra i processori. Tuttavia, anche se indispensabili, le comunicazioni «rubano» tempo prezioso, tra l'altro sono in genere pesanti da gestire, e devono essere minimizzate.

L'algoritmo

L'idea alla base dell'algoritmo è quella di cercare la soluzione del sistema iterando le n soluzioni di ogni singola equazione finché tutti i processori non trovano gli stessi valori. In pratica si tratta di convergere ad una soluzione comune; come tutti gli algoritmi di questo tipo è necessario un criterio di convergenza che assicuri se ci stiamo avvicinando alla soluzione. Basterà valutare l'errore che si commette scegliendo la nuova soluzione rispetto a quella della precedente iterazione. Per capire meglio il meccanismo di quest'algoritmo illustriamo con un esempio il caso minimo di un sistema a due equazioni e due incognite:

```
y=a-x
y=x-b
```

Il processore A cercherà la soluzione per la prima equazione e il processore B

per la seconda; siccome stiamo usando un criterio di ricerca per convergenza è necessario stabilire un punto dal quale partire cioè stimare in qualche modo la soluzione. Tralasciando tutti i discorsi teorici dell'analisi numerica, prendiamo una soluzione a caso cioè assegnamo due valori a x e a y come ci pare.

Il primo, la x, verrà passato al processore A, il secondo, la y, al processore B, e per chiarezza li indicheremo con xA e yB; il primo processore risolverà di conseguenza la prima equazione rispetto a x fornendo y ed il secondo viceversa risolverà la seconda rispetto a y fornendo x.

Perciò avremo:

yA=a−xA dal processore A
xB=b+yB dal processore B

Il secondo passo consiste nel valutare una nuova soluzione, ogni processore stimerà allora un nuovo valore della propria variabile «indipendente» (x per A e y per B) e ricalcolerà le soluzioni.

Avremo quindi:

yA'=a−xA' dal processore A
xB'=b+yB' dal processore B

dove xA' è la nuova stima per la x sul processore A e yB' è la nuova stima per la y sul processore B.

Confrontiamo i valori calcolati da un processore con quelli stimati dall'altro, se si ottiene:

yA'=yB' e xA'=xB'

allora significa che è stata raggiunta la soluzione del sistema.

Se invece anche una sola di queste condizioni non è verificata, bisogna applicare il criterio per vedere se la nuova stima ci sta portando verso la soluzione; verranno perciò calcolati i due valori seguenti per ciascun processore:

$E=(xA-xB)^2+(yA-yB)^2$
$E'=(xA'-xB)^2+(yA'-yB)^2$ per il processore A

$E=(xB-xA)^2+(yB-yA)^2$
$E'=(xB'-xA)^2+(yB'-yA)^2$ per il processore B

In questo passo saranno necessarie comunicazioni dal processore A verso B per passare xA e yA e dal processore B verso A per trasmettere xB e yB.

Di seguito ogni processore confronterà E con E' e se E'<E la nuova soluzione verrà accettata, inoltre significa che la stima fatta della variabile indipendente ci avvicina alla soluzione perciò dobbiamo proseguire nell'incremento se la stima era più grande della precedente o diminuire se invece avevamo scelto una stima inferiore; se viceversa E'>E allora stiamo divergendo dalla soluzione, quindi manterremo la soluzione dell'iterazione precedente e cambieremo il verso della nuova stima.

È importante puntualizzare, se non fosse chiaro, che ogni processore esegue questi test in modo del tutto indipendente perciò l'unico punto di contatto fra i processori è il passaggio della soluzione stimata (esclusa la prima iterazione in cui è necessario passare anche la prima soluzione).

L'estensione a n equazioni con n processori è diretta, lo scambio delle soluzioni avverrà non soltanto con un singolo processore ma con gli altri n−1 della rete, la valutazione delle espressioni E ed E' sarà espansa di conseguenza.

```
PROC nodo

  SEQ
    - inizializzazione
    PAR
      WHILE non è trovata soluzione del sistema
        - processo segretario
        PAR
          - ricezione dagli altri n-1 nodi
          - trasmissione al proc. operaio dei dati ricevuti
          - trasmissione della soluzione agli altri n-1 nodi
          - ricezione dal proc. operario della nuova stima
        - processo operaio
          WHILE non è trovata la soluzione del sistema
            SEQ
              - ricevi dal proc. segretario le n-1 soluzioni
              - calcola la stima e la nuova soluzione
              - valuta la convergenza
              - trasm. la nuova soluz. al proc. segretario
```

Figura 1 - Pseudo codice per il processo di nodo per la soluzione del sistema di equazioni. Il programma parallelo è costituito da n processi tutti uguali in parallelo.

L'implementazione

L'implementazione dell'algoritmo è abbastanza semplice: ci sono infatti due sezioni ben distinte, una dedicata alle comunicazioni e una dedicata al calcolo. La realizzazione è chiaramente legata all'architettura che si ha a disposizione per eseguire l'algoritmo, per generalità supporremo di implementare quest'algoritmo su un sistema multiprocessore asincrono di tipo MIMD, come può essere una rete di Transputer. La tipologia non è vincolante se non per le prestazioni perciò non faremo nessuna assunzione su di essa.

Per generalizzare ulteriormente la situazione, si può pensare di non predisporre più processi su più processori ma un solo programma costituito da più processi asincroni paralleli che possono girare anche su un solo processore.

In figura 1 potete trovare un codice pseudo Occam che illustra i passi necessari alla codifica del programma parallelo per un solo nodo (processo o processore). Il principale problema da risolvere rimane quello di riuscire a ricevere da tutti i nodi della rete e contemporaneamente trasmettere la soluzione calcolata mentre gli stessi dati vengono comunicati tra il processo segretario e quello operaio, senza incorrere in errori di condivisioni di variabile. D'altra parte non è possibile sequenzializzare queste attività per non incorrere in problemi di stallo in cui ogni processore aspetta gli altri, o viceversa dove ognuno dei processori tenta di trasmettere a tutti gli altri mentre nessuno è pronto a ricevere!

Naturalmente in figura 1 è mostrato uno schema assai semplificato in quanto bisognerebbe prevedere tutti i meccanismi di scambio fra i processi coinvolti, non possiamo per ovvie ragioni e per la filosofia di MC esporre tutto il listato del programma; crediamo invece che le informazioni qui illustrate siano sufficienti ai lettori che vogliano costruire un programma basato su questo algoritmo.

Questo spazio rimane comunque aperto per tutte le richieste e le curiosità dei lettori che ci seguono con interesse.

MC

Bibliografia

Vladimir Cerny, ***«Statistically coupled transputer»,*** Parallelogram int., apr. 90, pp. 16 -17