Architetture e progettazione di sistemi dedicati

Lezione 02 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2019-20

Indice

argomenti della lezione

sommario:

- dualismo dei paradigmi di progettazione hardware e software
- modelli di codesign
- esempio: un componente Collatz delay per il codesign
- concorrenza e parallelismo
- problemi proposti (nell'area riservata)

paradigmi di progettazione hardware e software

sfida professionale decisiva nel codesign hardware-software:

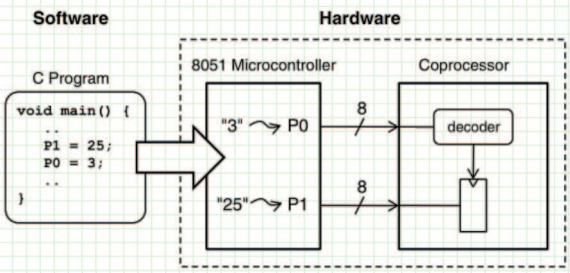
capacità di combinare due paradigmi di progettazione radicalmente diversi

hardware e software sono reciprocamente duali in molti aspetti

ecco una sintesi comparativa delle loro differenze fondamentali (cf. Schaumont, Table 1.1)

	Hardware	Software
paradigma di progettazione	decomposizione nello spazio	decomposizione nel tempo
misura di costo di risorse	area (# di porte logiche)	tempo (# di istruzioni)
flessibilità	va progettata	implicita
parallelismo	implicito	va progettato
modellazione	modello ≠ implementazione	modello ~ implementazione
riuso	non comune	comune

un semplice esempio mette in luce la varietà di modelli che entrano in gioco nel codesign hardware-software:



Schaumont, Fig. 1.3 - A codesign model

- modelli software: il programma C, il suo eseguibile in linguaggio macchina del microprocessore
- modelli hardware: del microprocessore, del coprocessore, dell'interfaccia hardware fra essi
- un modello dell'interfaccia hardware-software: quali istruzioni determinano quali interazioni fra microprocessore e coprocessore

i dettagli della formalizzazione in Gezel di questo esempio sono omessi

il circuito hardware presentato nella prima lezione difficilmente potrebbe essere usato come coprocessore per accelerare la visualizzazione di una traiettoria di Collatz

perché?

può però essere incorporato in un coprocessore progettato per accelerare il calcolo di funzioni su una traiettoria di Collatz

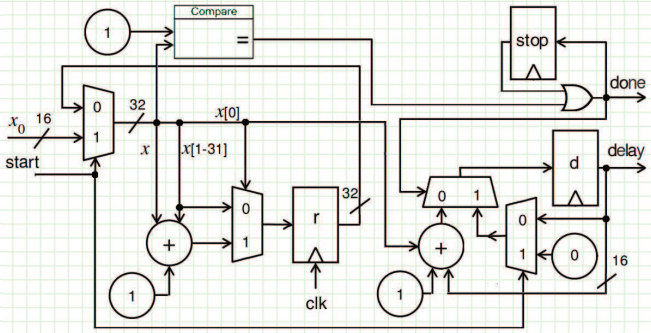
per esempio: il *delay* della traiettoria, il valore di picco (massimo) raggiunto ecc.

a tal fine occorre ridefinire l'interfaccia del circuito ed estenderlo con logica di controllo, e.g. per fermare la computazione e produrre il risultato in uscita alla prima occorrenza di '1' nella traiettoria

N.B. con riguardo al Collatz delay, tenere in conto che:

- il delay cresce di 2 a ogni iterazione da un valore dispari
- '1' è un valore iniziale lecito, nel qual caso il delay è 0

un'estensione del circuito visto nella prima lezione che non dà in output la traiettoria bensì il suo *delay*:



Datapath hardware per il *delay* di una traiettoria di Collatz

rappresentazione in Gezel:

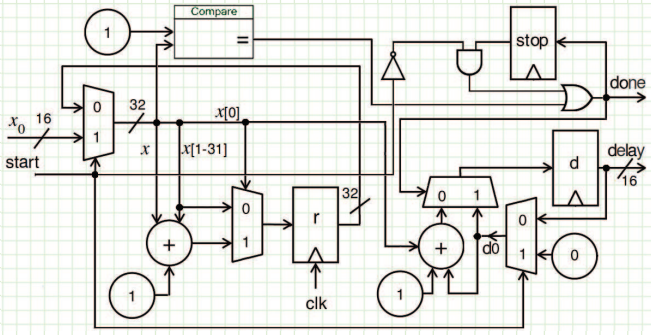
```
dp delay_collatz (
   in start : ns(1) ; in x0 : ns(16) ;
   out done : ns(1) ; out delay : ns(16))
{  reg r : ns(32) ;
   reg d : ns(16) ;
   reg stop : ns(1) ;
   sig x : ns(32) ;
   always { x = start ? x0 : r ;
        r = x[0] ? x + (x >> 1) + 1 : x >> 1 ;
        done = ( x == 1 ) | stop ;
        stop = done ;
        d = done ? ( start ? 0 : d ) : d + 1 + x[0] ;
        delay = d ;
} }
```


l'interfaccia del circuito appena visto suggerisce una facile implementazione del coprocessore come dispositivo di I/O mappato in memoria, per esempio dotato di:

- registro di controllo, con bit *start*
- registro di stato, con bit *done*
- registri dati per l'input del valore iniziale e l'output del risultato

ma... è adeguato il suddetto circuito a effettuare il calcolo richiesto per successive interazioni con il software?

revisione del circuito per il *delay* di traiettorie di Collatz:



Datapath hardware per il *delay* di traiettorie di Collatz

rappresentazione in Gezel:

```
dp delay_collatz_rev (
   in start : ns(1) ; in x0 : ns(16) ;
   out done : ns(1) ; out delay : ns(16))
{  reg r : ns(32) ;
   reg d : ns(16) ;
   reg stop : ns(1) ;
   sig x : ns(32) ;
   sig d0, dd : ns(16) ;
   always { x = start ? x0 : r ;
        r = x[0] ? x + (x >> 1) + 1 : x >> 1 ;
        done = ( x == 1 ) | ( stop & ~start ) ;
        stop = done ;
        dd = 1 + x[0] ;
        d0 = start ? 0 : d ;
        d = done ? d0 : d0 + dd ;
        delay = d ;
} }
```


concorrenza e parallelismo non sono sinonimi:

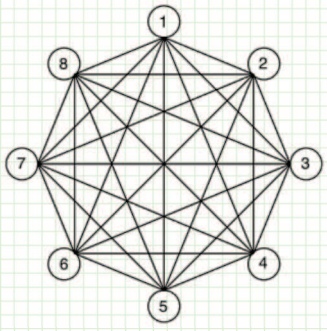
- processi *concorrenti*: indipendenza reciproca delle loro computazioni
- processi *paralleli*: simultaneità delle loro esecuzioni su processori o circuiti distinti

la concorrenza è una proprietà dell'applicazione, il parallelismo è una proprietà della sua implementazione, che presuppone:

- concorrenza nell'applicazione, e
- un'architettura hardware parallela

e.g. la Connection Machine (CM), v. figura

la legge di Amdahl limita a $1/s$ il massimo guadagno di prestazione, o *speed-up*, conseguibile con il parallelismo per un'applicazione che contenga una frazione s di esecuzione sequenziale



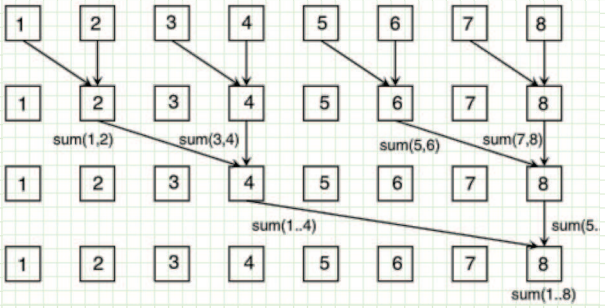
Schaumont, Fig. 1.9 - Eight node connection machine

è difficile escogitare algoritmi concorrenti per architetture parallele?

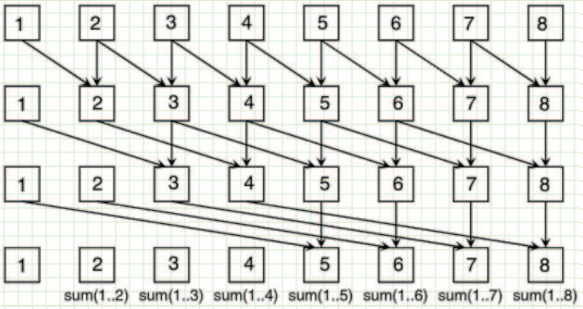
non necessariamente, dipende da formazione e abitudini di programmazione

si consideri per esempio la somma di n numeri sulla CM, diciamo con $n = 8$, assegnando inizialmente un numero a ciascun processore

gli algoritmi illustrati appresso effettuano la somma in $\lceil \log_2(n) \rceil$ passi



Schaumont, Fig. 1.10 - Parallel sum



Schaumont, Fig. 1.11 - Parallel partial sum

letture raccomandate:

Schaumont Ch. 1, Sect. 1.5, 1.7

Zwoliński Ch. 1, Sect. 1.1

per ulteriore consultazione:

F. Vahid & T. Givargis Ch. 1, Sect. 1.5-1.6