Modelli dataflow, flusso del controllo

Lezione 03 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2018-19

Indice

argomenti della lezione

di che si tratta:

- grafi dataflow
- grafi dataflow sincroni (SDF)
- algoritmo PASS per l'analisi di SDF
- limiti dei grafi SDF per il flusso del controllo
- estensioni dei grafi SDF per l'analisi di prestazioni
- trasformazioni di grafi SDF

grafi dataflow

modelli di alto livello per il progetto di sistemi: diagrammi a blocchi

- noti anche come architetture di filtri e canali
- ogni blocco (filtro) elabora un flusso di dati proveniente dai canali d'ingresso e produce un flusso di dati sui canali di uscita
- partizionamento funzionale
- non forza realizzazioni in hardware o in software

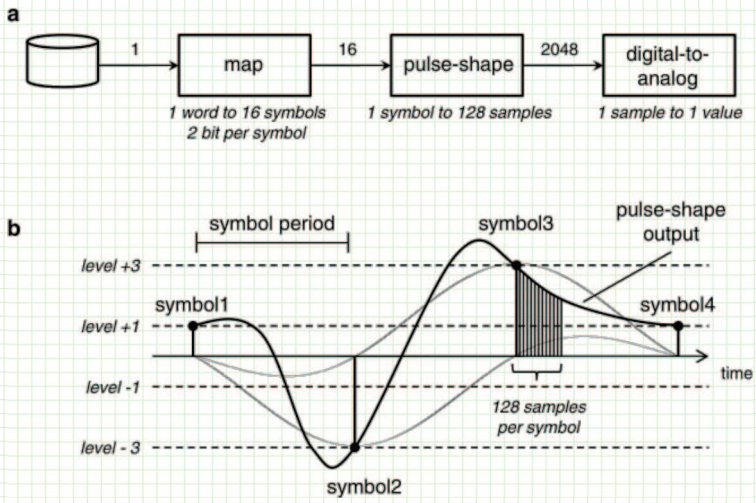
grafi dataflow: modello matematico dei diagrammi a blocchi

- sono un modello di computazione (MoC)
- di largo impiego nell'elaborazione di segnali
- altri modelli correlati: sistemi di attori (Hewitt), reti di Kahn, reti di Petri, ...

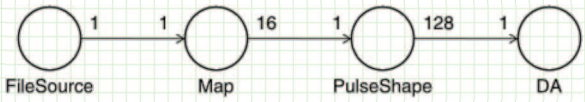
esempio: diagramma a blocchi di un sistema di modulazione di ampiezza a 4 livelli (PAM-4)

v. appresso

esempio: modulazione di ampiezza di segnali a 4 livelli (PAM-4)



Schaumont, Figure 2.1 - (a) sistema di modulazione di ampiezza (b) formazione del segnale



Schaumont, Figure 2.2 - modello dataflow del sistema di modulazione di ampiezza

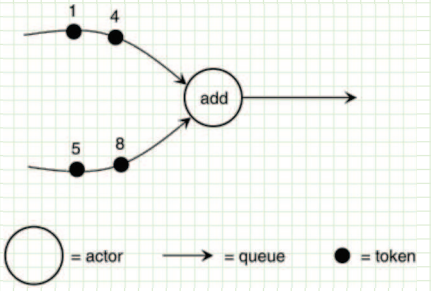
un modello software del sistema PAM-4 è fattibile, v. per esempio il programma C in Schaumont, Listing 2.1, dove ciascun blocco è rappresentato da una funzione, tuttavia:

- il modello software impone un'ordine sequenziale di esecuzione delle funzioni
- sono fattibili realizzazioni hardware in cui la loro esecuzione è parallela, con i rispettivi componenti operanti in pipeline
- un modello dataflow permette entrambe le soluzioni

nel modello in figura le funzioni sono rappresentate da *attori* collegati da canali di comunicazione modellati da *code FIFO*

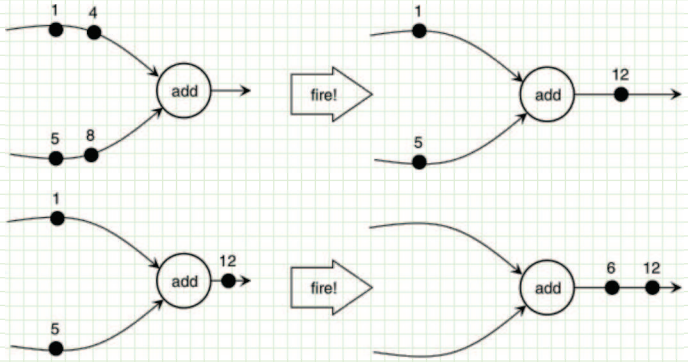
- ogni attore agisce come unità di elaborazione indipendente, con ingressi e uscite etichettati dai rispettivi *tassi di consumo e produzione di dati* in ciascuna attivazione dell'attore
- vantaggi: i grafi dataflow sono un MoC concorrente, distribuito e modulare
- per il loro carattere matematico, i grafi dataflow si prestano ad analisi con cui il progettista può determinare utili proprietà di un'architettura astratta, e.g. assenza di deadlock, ben prima della sua realizzazione

costituenti dei modelli dataflow



Schaumont, Figure 2.3 - modello dataflow di un'addizione

- attori: esecuzione iterativa, ogni iterazione (*firing*) è soggetta alla disponibilità di dati in tutti i canali di ingresso
- canali: code FIFO illimitate
- dati: rappresentati da valori di *token* presenti nei canali



Schaumont, Figure 2.4 - risultato di due attivazioni dell'attore add, ciascuna produce un marking diverso

marking: stato di un grafo dataflow, costituito da un assegnamento di token ai canali

Schaumont, Figure 2.7 - esempio di modello dataflow multi-rate

grafi dataflow sincroni (SDF)

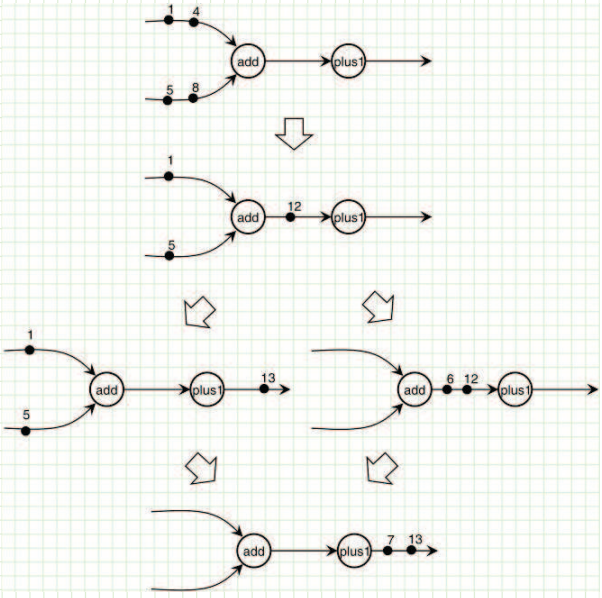
gli attori sono unità funzionali di elaborazione privi di stato interno

> la sola informazione di stato osservabile di un grafo dataflow è il marking

nei grafi dataflow *multirate* ogni attore può consumare più di un token su ciascun canale d'ingresso e produrre più di un token su ciascun canale di uscita, ad ogni attivazione

quando i tassi di produzione e consumo sono fissi, il grafo è detto *grafo dataflow sincrono* (SDF)

- i grafi SDF non possono rappresentare attivazioni dipendenti dai dati
- tuttavia sono meglio suscettibili di analisi matematica delle loro proprietà
- inoltre, sono *deterministici* (v. figura)

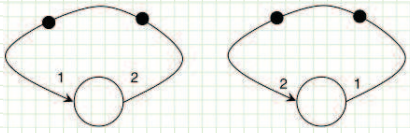


Schaumont, Figure 2.8 - i grafi SDF sono deterministici

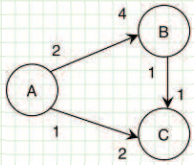
analisi di grafi SDF

proprietà desiderabili di un grafo SDF:

- *esecuzione illimitata*
- *buffer limitati*



Schaumont, Figure 2.9 - quale grafo SDF va in stallo e quale è instabile?



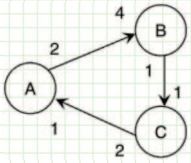
Schaumont, Figure 2.10 - grafo SDF di esempio per la costruzione di una PASS

PASS: schedule sequenziale ammissibile periodica

- sequenziale: attivazione di un attore per volta
- ammissibile: assenza di stalli + buffer limitati
- periodicità della sequenza di stati → esecuzione illimitata

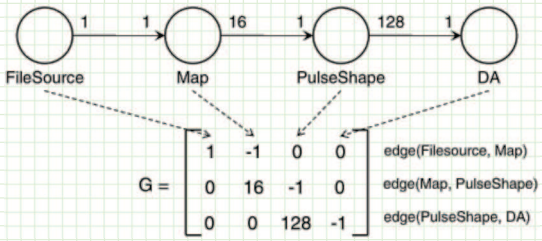
algoritmo di costruzione di PASS, se ne esiste una (Lee):

1. crea la matrice topologica G del grafo SDF
2. verifica che rango della matrice = n. di attori meno uno
3. determina un vettore di attivazione q tale che Gq = 0
4. prova a turno l'attivazione di ogni attore fino al conteggio nel vettore



Schaumont, Figure 2.11 - un grafo in stallo

esempio di PASS per il modello di elaborazione di segnali



Schaumont, Figure 2.12 - matrice topologica del sistema PAM-4

applicazione del metodo di Lee:

1. v. figura
2. OK, le tre righe di G sono linearmente indipendenti
3. $q^T = [1 \quad 1 \quad 16 \quad 2048]$
4. la sequenza di attivazioni descritta da q^T è una PASS, che però richiede una capacità di 2048 posizioni nella coda più a destra
   una soluzione più economica è la sequenza (1, 1, 16*(1, 128)), che la riduce a 128

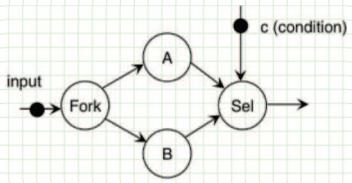
modellare il flusso del controllo con i grafi SDF non è facile...

alcuni aspetti problematici:

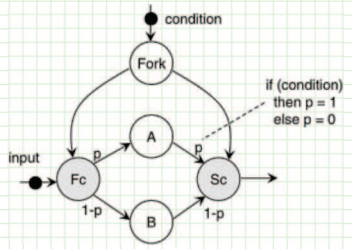
- arresto e reinizio dell'esecuzione
- reti a topologia dinamica
- gestione di eccezioni
- condizioni a run-time, e.g. esecuzione condizionale

  una semplice condizione if-then-else è problematica con i grafi SDF

  soluzioni:
  - emulazione in SDF con attori Fork-Sel
  - estensione della semantica SDF a Boolean Data Flow (BDF)



Schaumont, Figure 2.13 - Emulazione di condizioni if-then-else in SDF

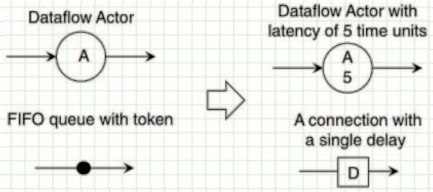


Schaumont, Figure 2.14 - condizioni if-then-else in Boolean Data Flow

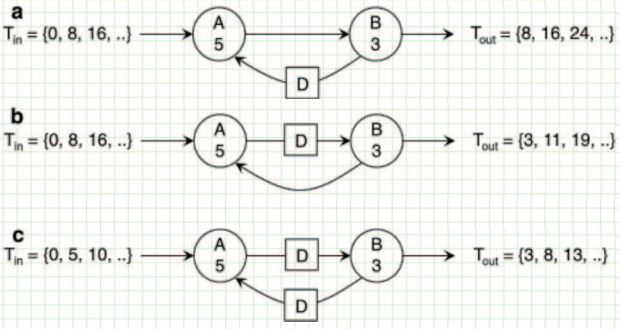
estensioni minime, preservano la semantica SDF:

- tempo: latenza di attori
- spazio: code di capacità limitata

utili per l'analisi di prestazioni e per la valutazione di trasformazioni volte a migliorarle



Schaumont, Figure 2.15 - Estensione del modelloSDF con risorse: latenza di attori, buffer per code FIFO



Schaumont, Figure 2.16 - tre grafi dataflow

nello stato iniziale i buffer contengono sempre un token

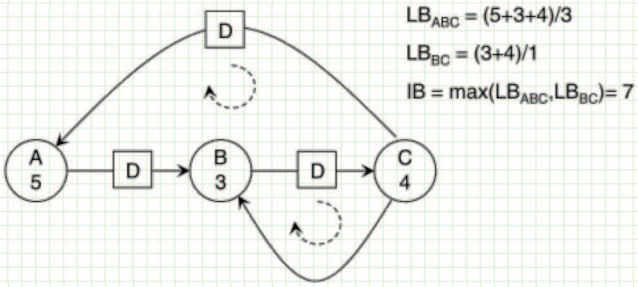
un buffer può detenere un token per la durata di una attivazione dell'attore a valle

con l'aggiunta di buffer, il pipelining può migliorare il *throughput* di un grafo SDF

numero e distribuzione dei buffer in un grafo SDF ne influenzano il throughput

anche le maglie, o cicli, nel grafo influiscono su latenza e throughput; due concetti per rendersene conto:

- limite di maglia: latenza lungo una data maglia divisa per il numero di buffer sulla maglia
- limite d'iterazione: massimo limite di maglia nel grafo



Schaumont, Figure 2.17 - Calcolo di limiti di maglia e d'iterazione

il throughput di un grafo SDF non può superare il (reciproco del) suo limite d'iterazione

anche i grafi lineari hanno un limite d'iterazione (feedback implicito da uscita a ingresso)

Schaumont, Figure 2.18 - limite d'iterazione per un grafo lineare

trasformazioni che non alterano la funzionalità ma possono migliorare le prestazioni di grafi SDF

aumentare il throughput e/o ridurre la latenza

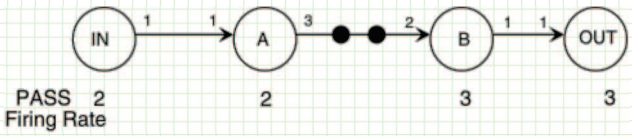
quattro le più frequenti:

- *espansione multirate*: trasforma un grafo SDF multirate in uno single-rate
  utile perché le altre tre sono definite per grafi SDF single-rate
- *retiming*: redistribuzione di buffer sugli archi del grafo
  per migliorare il throughput, senza effetto sulla latenza né sul transitorio
- *pipelining*: aggiunta di buffer per migliorare il limite d'iterazione
  modifica il throughput e il transitorio
- *unfolding*: replica di attori per aumentare il parallelismo computazionale del grafo
  può modificare il throughput, non il transitorio

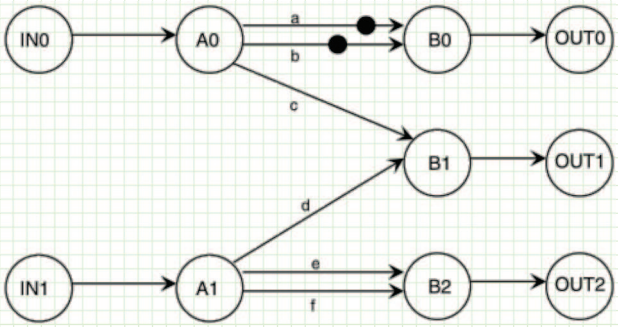
espansione di grafi SDF multirate

trasformazione di un grafo SDF multirate in single-rate, in 5 passi:

1. calcolare il vettore di attivazione degli attori
2. replicare ogni attore tanto quanto dettato dal tasso di attivazione
3. replicare ogni I/O di ogni replica di attore in tanti I/O single-rate quanti indicati dal suo tasso di consumo/produzione
4. reinserire le code che connettano tutti gli attori nel grafo SDF
5. reinserire i token iniziali distribuendoli in sequenza sulle code single-rate



Schaumont, Figure 2.19 - grafo dataflow multi-rate

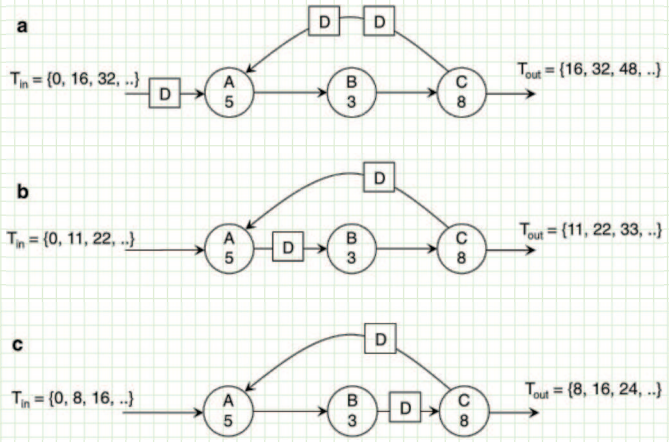


Schaumont, Figure 2.20 - grafo SDF multi-rate espanso a single-rate

retiming

questa trasformazione redistribuisce i buffer senza alterarne il numero totale

si trattano i buffer come token, spostandoli fra le code come determinato dalle attivazioni degli attori, e si valutano le prestazioni del grafo nei diversi stati della successione per scegliere quello che dà le migliori

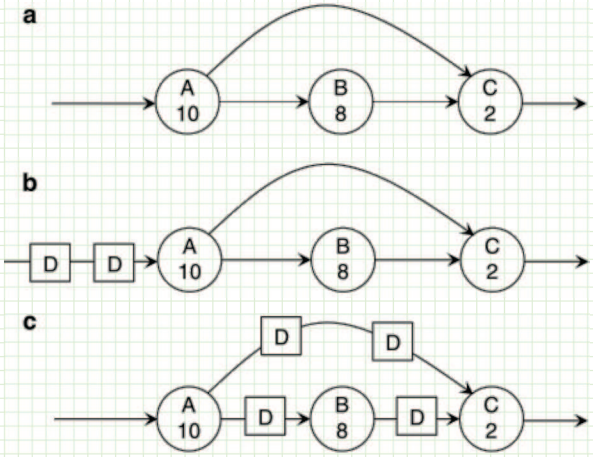


Schaumont, Figure 2.21 - Retiming: (a) grafo originale. (b) grafo dopo la prima trasformazione di re-timing. (c) grafo dopo la seconda trasformazione di re-timing

pipelining

pipelining = aggiunta di buffer + retiming

questa trasformazione migliora il throughput ma peggiora la latenza



Schaumont, Figure 2.22 - pipelining: (a) grafo originale (b) grafo dopo l'aggiunta di due stadi di pipeline (c) grafo dopo il retiming degli stadi di pipeline

unfolding

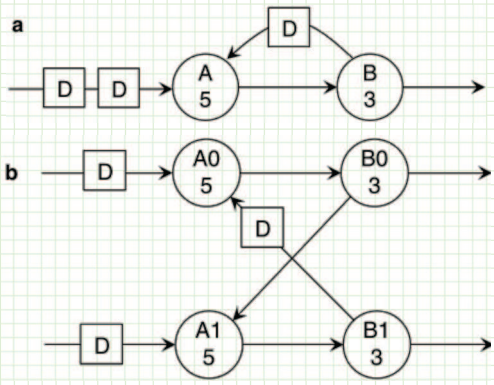
replica del grafo SDF in più copie per aumentare il parallelismo dell'elaborazione del flusso di dati in ingresso

regole in parte simili a quelle dell'espansione multirate

il V-unfolding di un grafo SDF genera un grafo con V copie di ogni attore e di ogni arco

se l'arco AB connette gli attori A e B nel grafo originale, dove ha n buffer, allora nel grafo trasformato, per $i = 0..V-1$:

- l'arco $(AB)_i$ connette l'attore A_i con l'attore B_k dove $k = (i + n) \bmod V$
- l'arco $(AB)_i$ ha $(i + n) / V$ buffer (con $V-n$ archi senza buffer se $n<V$)



Schaumont, Figure 2.23 - unfolding: (a) grafo originale (b) grafo dopo 2-unfolding

letture raccomandate:

Schaumont (2012) Cap. 2

per ulteriore consultazione:

E.A. Lee & S.A. Seshia (2015) Cap. 6, Sez. 6.3

P. Marwedel (2011) Cap. 2, Sez. 2.5

A.H Ghamarian *et al.* (2007)

S. Stuijk *et al.* (2006)