Forme di parallelismo

Note in aggiunta alla Sez. 3.6 del libro di testo

M. Danelutto

AEa 2020—21

Misure (primitive)

- **Latenza**
  tempo fra l'inizio di un calcolo e la produzione del relativo risultato
- **Tempo di completamento**
  *(di **m** calcoli diversi)*
  tempo fra l'inizio del primo calcolo e la terminazione dell'ultimo
- **Tempo di servizio**
  (*nel calcolo di **m** task diversi)*
  tempo che intercorre fra la produzione di due risultati consecutivi o fra l'inizio di due calcoli consecutivi

- **Throughput**
  *(nel calcolo di **m** task diversi)*
  numero di calcoli completati per unità di tempo (inverso del tempo di servizio)

- *Diminuire la latenza significa rendere disponibile prima il risultato del singolo calcolo*
- *Diminuire il tempo si servizio significa aumentare il throughput*

Misure (derivate)

- **Speedup(n)**
  rapporto fra il miglior tempo sequenziale e il tempo utilizzato in parallelo con grado di parallelismo pari a n

  speedup(n) = Tseq / T(n)

- **Scalabilità(n)**
  rapporto fra il tempo impiegato a calcolare con grado di parallelismo 1 e quello impiegato con grado di parallelismo n

  scalab(n) = T(1) / T(n)

- **Tempo ideale(n)**
  rapporto fra il tempo sequenziale e il grado di parallelismo n

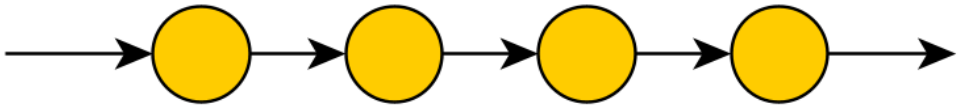
  Tid(n) = Tseq / n

- **Efficienza(n)**
  rapporto fra il tempo ideale e quello ottenuto con grado di parallelismo n

  eff(n) = Tid(n) / T(n)

  eff(n) = (Tseq / n) / T(n) = Tseq / (n*T(n)
  = speedup(n) / n

Forme di parallelismo: pipeline



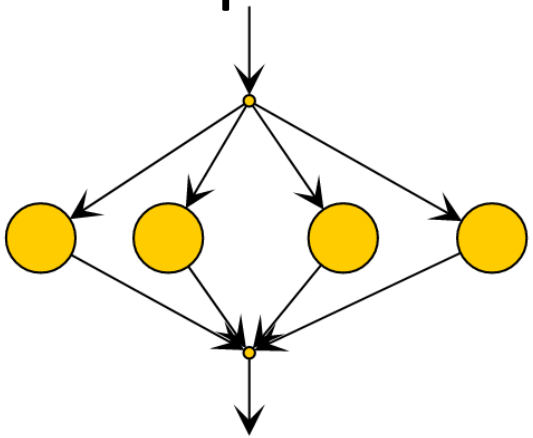
- *Riceve uno stream in dati in input*
- *Calcola dei risultati e li manda sullo stream di output*
- *Ogni stadio calcola un risultato parziale che diventa input per lo stadio successivo*
- *Stadio j calcola f_j*
- *Pipeline calcola $f_m(f_{m-1}(\ldots(f_2(f_1(x))..))$*

- **Tempo di servizio**
  (grado di parallelismo n)
  $Ts = max\{T_i\}$
  T_i= tempo di servizio stadio I

- **Tempo di completamento**
  (grado di par n e m tasks)
  $Tc(n,m) = \Sigma\ Ti + (m-1)\ max\{Ti\}$

- **Massimo speedup**
  numero degli stadi

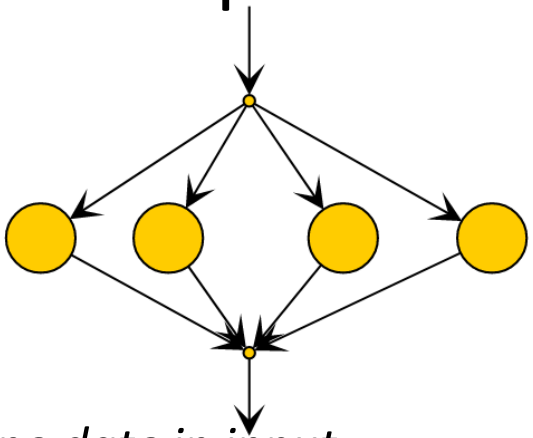
Forme di parallelismo: task farm



- *Riceve uno stream in dati in input*
- *Calcola dei risultati e li manda sullo stream di output*
- *Ognuno dei worker calcola la stessa funzione f sui dati che riceve in input*
- *La politica di schedulazione dei task in input può essere round robin, a richiesta o altro*
- *La politica di raccolta dati può mantenere o meno l'ordinamento di ingresso*

- **Tempo di servizio**
  (grado di parallelismo n)
  $Ts = max\{T_{sched}, T_{coll}, T_w/n\}$
  T_w= *tempo di servizio del worker*
  n = numero di worker
- **Tempo di completamento**
  (grado di par n e m tasks)
  $Tc(n,m) = m * T_s = m * T_w / n$
- **Massimo speedup**
  n

Forme di parallelismo: map



- *Riceve uno dato in input*
- *Divide il dato in sottotask*
- *Ognuno dei worker calcola la stessa funzione f sul sottotask che riceve in input*
- *La politica di schedulazione dei task in input può essere round robin o altro*
- *Quando tutti I worker hanno finito, I risultati dei sottotask vengono utilizzati per ricostruire il risultato finale (normalmente avente la stessa struttura del dato in input)*

- **Latenza**
  (grado di parallelismo n)
  $Tc = T_{split} + T_{seq}/n + T_{merge}$
  T_{seq}= *tempo di calcolo sequenziale*
  n = numero di worker
- **Massimo speedup**
  n
- **Tempo di servizio**
  (grado di par n e m tasks)
  $Ts = max\{T_{split}, T_{seq}/n, T_{merge}\}$

Tipi di parallelismo

- Temporale / spaziale (libro di testo)
  pipeline / map

- Data / Stream parallel
  {map} / {pipeline, farm}
  - Data parallelism: riduce latenza
    (se stadio di computazione stream parallel => di conseguenza riduce tempo di servizio)
  - Stream parallelism: aumenta throughput

Composizione

- Normalmente:
  - Componenti di una computazione stream parallel possono essere
    - Data parallel
    - Stream parallel
    - Sequenziali
  - Componenti di una computazione data parallel possono essere
    - Data parallel
    - Sequenziali
- Misure primitive si compongono di conseguenza:
  - Pipe(Seq(Latenza=L1), farm(Seq(Latenza=Lw),n worker), Seq(Latenza = L3))
  - Ts = max{T1, T2, T3} = max{L1, max{Tsched,Lw/n,Tcoll},L3}

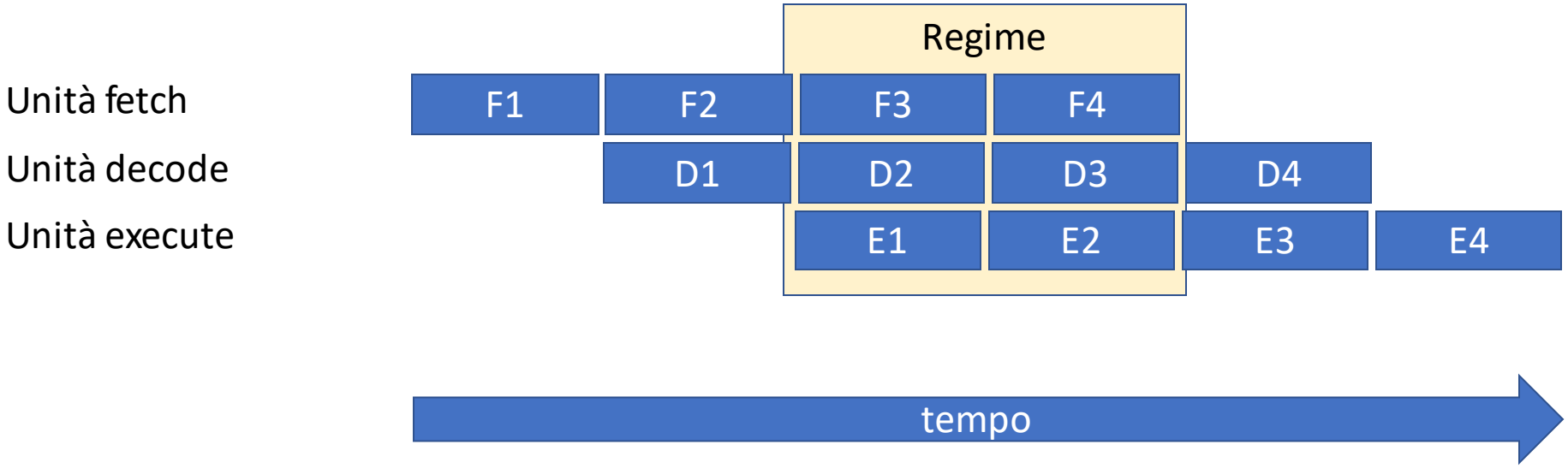
Rimozione colli di bottiglia

- Collo di bottiglia
  - Parte della computazione che rallenta tutta la computazione

- Esempio
  - Pipeline con 3 stadi
  - Stadio lento => collo di bottiglia
    - Renderlo un farm
    - Utilizzare data parallelism (se possibile)

Esempio (pipeline: parallelismo temporale)

- Fetch, decode, execute



Esempio map (parallelismo spaziale)

- Esecuzione di istruzioni vettoriali (SIMD, SSE, AVX, …)
  forall k : x[k] = y[k]+z[k]