Ottimizzazioni e trasformazioni eseguite sul codice

Linguaggi di descrizione dell'hardware

Introduzione

- La descrizione di un algoritmo (C,C++,JAVA, ma anche VHDL nei processi) può nascondere il parallelismo intrinseco all'algoritmo stesso
- In sede di compilazione si possono adottare trasformazioni del codice che evidenziano questo parallelismo perché sia sfruttato dagli strumenti di sintesi ad alto livello
- Il problema non riguarda solo la sintesi dell'hardware, ma anche la compilazione per architetture superscalari e sistemi multiprocessore

Sommario

- Tecniche:
  - CSE: common subexpression elimination
  - trasformazioni su cicli
- Obbiettivi:
  - rimozione di operazioni ridondanti
  - riduzione del numero di istruzioni
  - aiuto agli strumenti di HLS
- Propagazione di costante ed eliminazione di codice non raggiungibile
- Trasformazioni che durante lo scheduling aumentano il grado di parall elismo intrinseco al codice

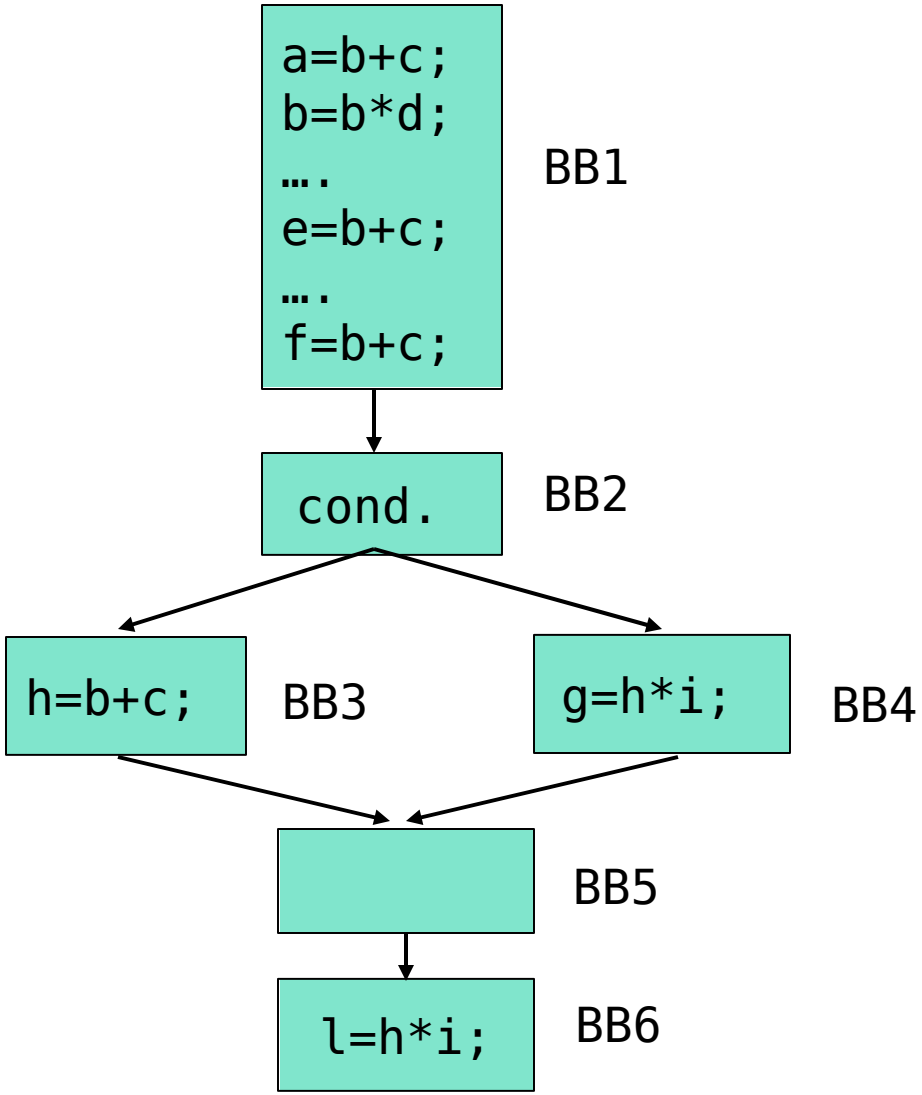
Common subexpression elimination

- Rivela sottoespressioni comuni in un frammento di codice
- Le memorizza in una variabile che viene poi riutilizzata sostituendola nelle occorrenze successive del codice
- Problema: come determinare il range di codice in cui una sottoespressione può essere eliminata?

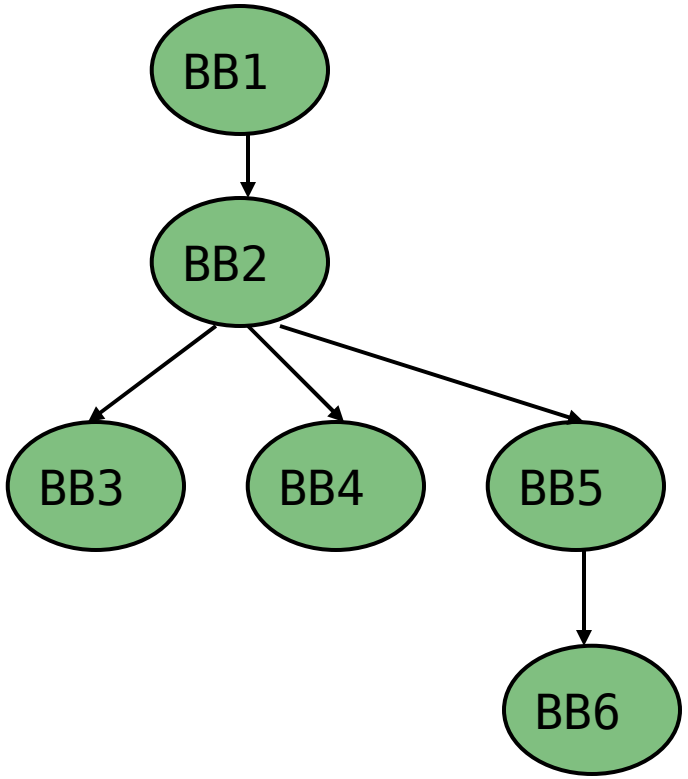
Dominator trees

- Un nodo *n* in un CFG ne domina un altro *m* se ogni cammino che dalla radice del grafo a *m* passa attraverso *n*
- Per un certo nodo *m* si può definire il dominator set *dom(n)* come l'insieme dei nodi che lo dominano
- Per mantenere la semantica di un CFG, una sottoepressione comune in un operazione *op2* può essere sostituita con il risultato di un operazione *op1*, se *op1* risiede in un blocco bb1 che domina quello (bb2) in cui *op2* risiede
- Le variabili che compaiono nella sottoespressione comune non devono venire assegnate dopo che tale sottoespressione viene valutata

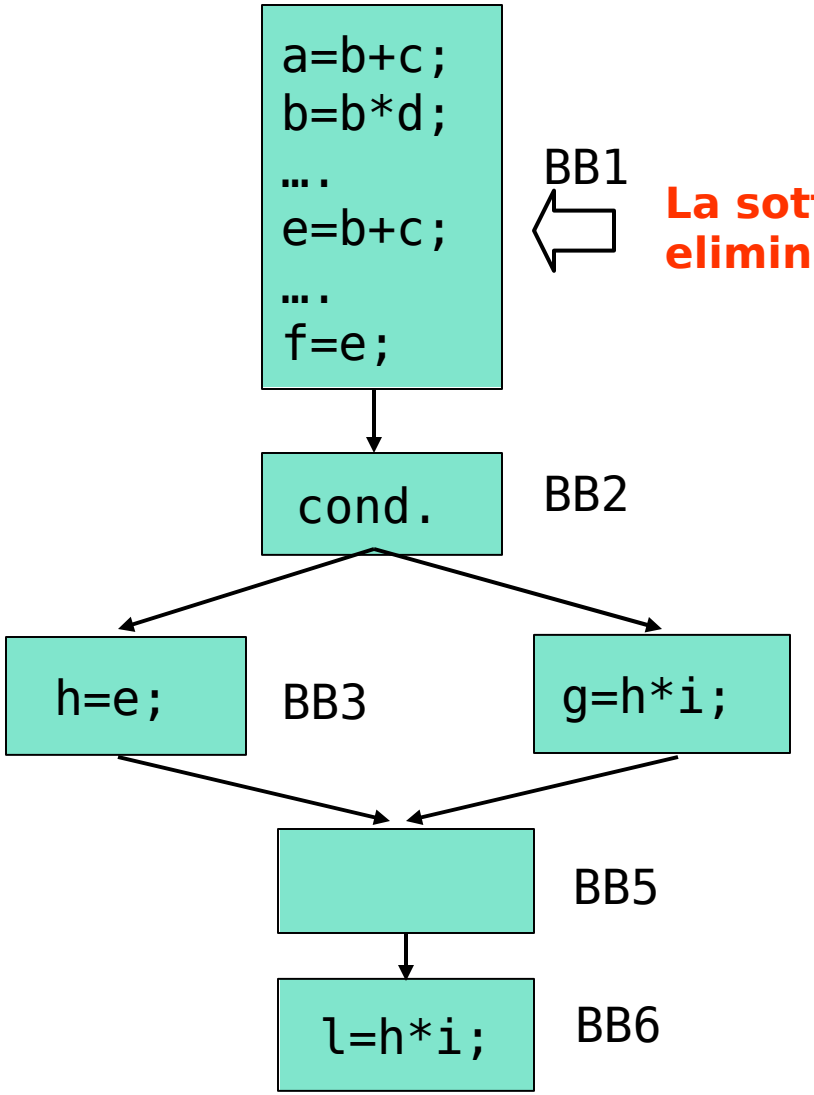
Esempio - I



Dominator tree



Esempio - II



La sottoespressione comune (b+c) può essere eliminata solo qui perché b viene scritta

La sottoespressione comune (h*i) può essere eliminata solo qui perché b viene scritta

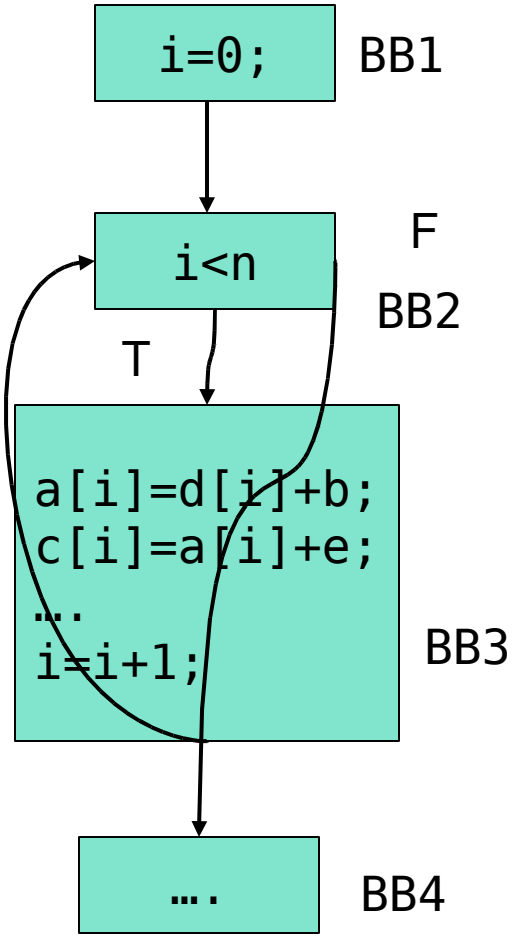
Loop-invariant code motion

- All'interno di un ciclo possono esistere operazioni che producono lo stesso risultato ogni volta che il ciclo viene eseguito
- Spostando fuori dal ciclo tali operazioni si riduce il numero di operazioni svolte nel caso in cui il ciclo venga eseguito più di una volta
- Se il loop viene eseguito molto raramente si potrebbe avere uno svantaggio

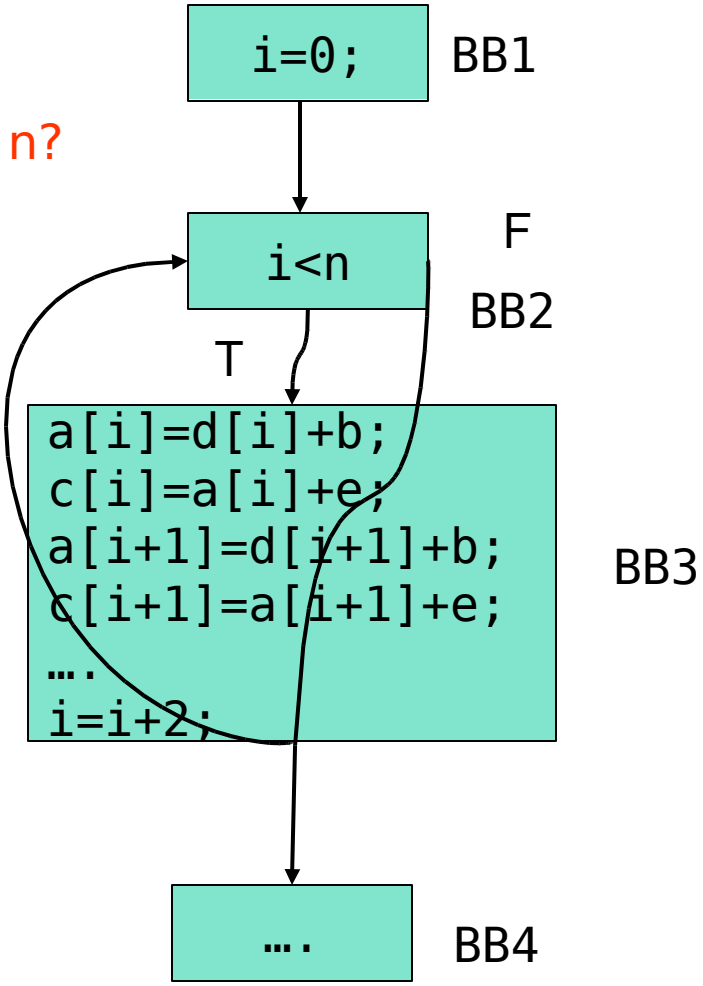
Loop unrolling

- E’ il processo con cui il corpo di un ciclo viene replicato per una (duplicazione) o più volte
- Questo consente di evidenziare il parallelismo all’interno del codice
- E’ una tecnica usata sia nel software che nell’hardware
- In entrambi i casi bisogna evitare di aggiungere troppe istruzioni

Esempio



valore di n?

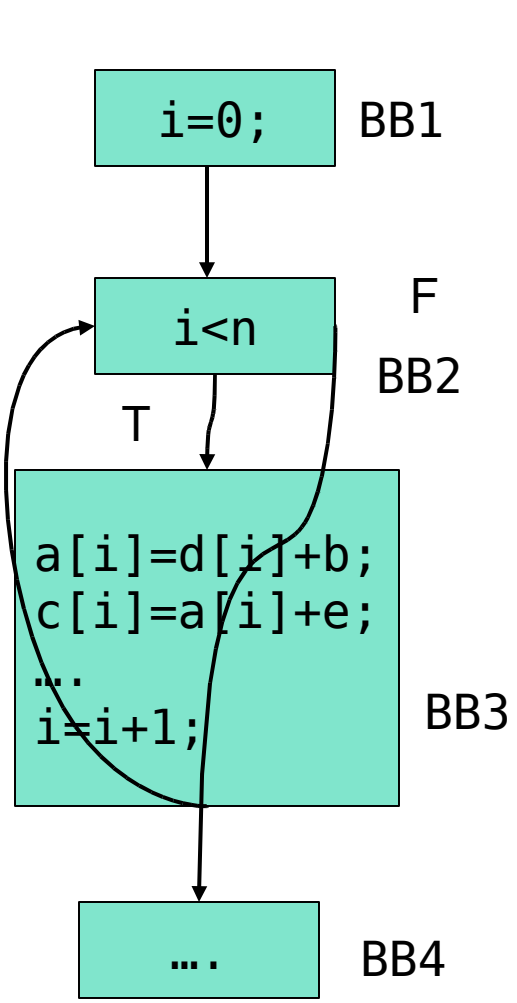


Si tracci il DFG di BB3

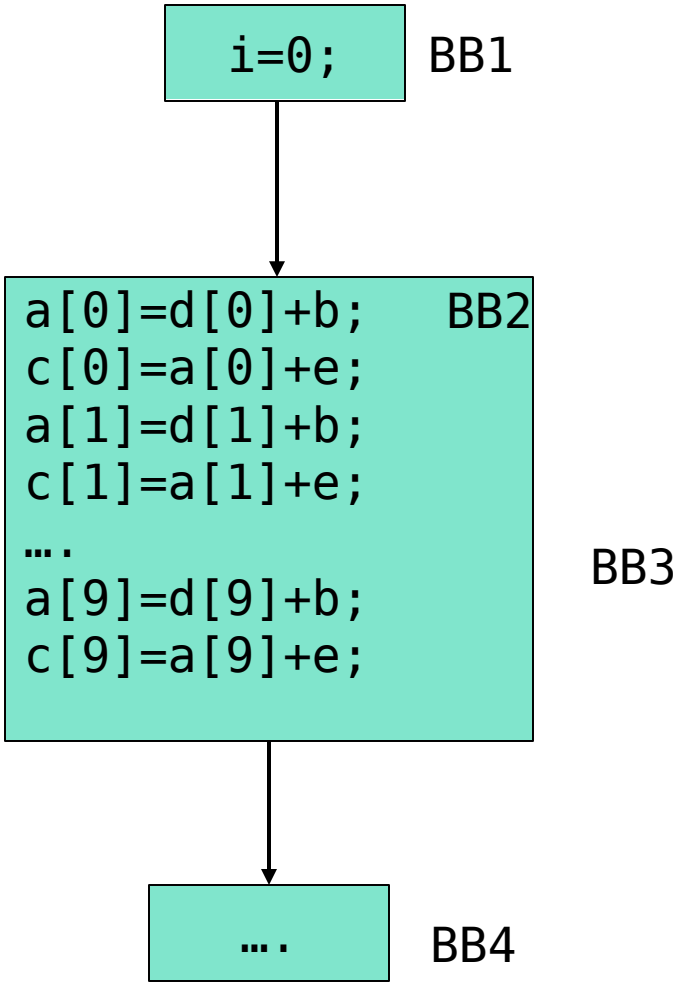
Loop index variable elimination

- Se il loop viene srotolato completamente, si può eliminare la variabile indice
- Condizioni:
  - numero di iterazioni deve essere fissato staticamente
  - numero di iterazioni limitato

Esempio


i=0;
BB1
i<n
F
BB2
T
a[i]=d[i]+b;
c[i]=a[i]+e;
….
i=i+1;
BB3
….
BB4

n=9


i=0;
BB1
a[0]=d[0]+b;
c[0]=a[0]+e;
a[1]=d[1]+b;
c[1]=a[1]+e;
….
a[9]=d[9]+b;
c[9]=a[9]+e;
BB2
BB3
….
BB4

Trasformazioni utilizzate durante lo scheduling

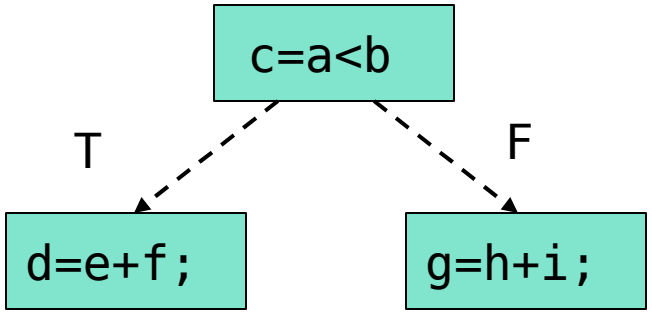
- Si tratta di un ulteriore classe di trasformazioni del codice (che quindi riguardano sia la sintesi che la compilazione) che aumentano il parallelismo del codice
- Il codice dominato dal controllo non rende spesso visibile il parallelismo intrinseco a una data applicazione
- Muovendo alcune istruzioni attraverso diramazioni e cicli si può evidenziare tale parallelismo
- Tali trasformazioni sono particolarmente importanti per la sintesi di sistemi pipelined

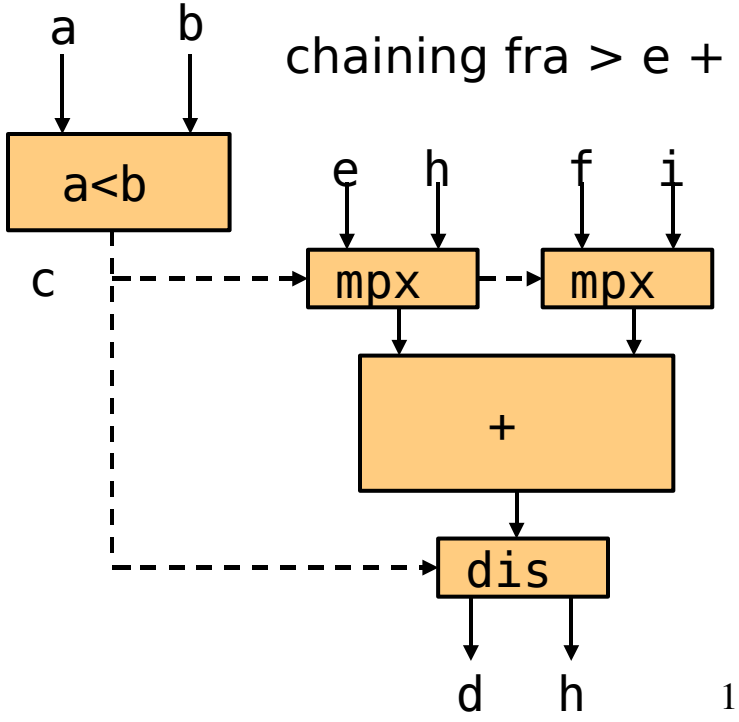
Esecuzione speculativa - I

- E' una tecnica comunemente utilizzata nella compilazione per CPU superscalari e pipelined per evidenziare il parallelismo al livello di istruzione
- Esecuzione di un istruzione prima che sia valutata l'istruzione condizionale che ne controlla l'esecuzione
- Può essere realizzata al livello di compilazione o dinamicamente (branch predictor nelle CPU)
- Analizzeremo il problema dal punto di vista della sintesi ad alto livello dell'hardware

Esecuzione speculativa - II

- Due operazioni si dicono mutuamente esclusive se vengono eseguite in condizioni complementari
- Due operazioni mutuamente esclsuive possono essere programmate nello stesso ciclo di clock sulla stessa risorsa
- Esempio





Esecuzione speculativa - III

- Supponiamo che sia disponibile un adder in più
- Le due operazioni sono eseguite in maniera speculativa e concorrente con il confronto