assignment 4 : loop fusion

- obiettivo: sfruttare meglio il data reuse del programma, avvicinando accessi alle stesse locazioni che si trovano in punti diversi del programma
- ovvero: c'è data reuse nel programma, ma al momento non riesco ad usarlo

struttura del passo

- serve un `FunctionPass`
- vedremo poi come organizzare l'iterazione sui loop della funzione: in particolare la questione si complica in caso di loop innestati, dunque useremo un vettore worklist?

algoritmo

condizioni per la loop fusion:

1. i due loop devono essere adiacenti: non ci devono essere statement tra la fine del primo e l'inizio del secondo
2. devono iterare lo stesso numero di volte
3. devono essere **control flow equivalent**: devo garantire che entrambi eseguano (se eseguono)
4. non devono esistere dipendenze a distanza negativa tra il primo loop e il secondo:
   - una dip a distanza negativa avviene quando nel secondo loop a iterazione m viene usato un valore calcolato nel primo loop a iterazione m+n con n > 0

adiacenza dei loop

devo evidentemente controllare l'uguaglianza tra exit bb e header ? recupera da dani

unica accortezza: se il loop è guarded (rivedi terminologia loop llvm da pagina docu) $\rightarrow$ controllo molto semplice da svolgere

3 control flow equivalenza

affinche si possa fare la funzione devo garantire che l'esecuzione di un loop implichi l'esecuzione anche dell'altro, **in entrambe le direzioni!!**

per fare il controllo verifico nuovamente condizioni di dominanza:

- evidente per la prima condizione (primo loop domina il secondo significa che il secondo deve eseguire per forza se esegue il primo)
- per la seconda: devo controllare le condizioni di **postdominanza** (recupero tramite `AM.getResult<PostDominatorTreeAnalysis>(F)`)

  ovviamente devo includere header file necessari

loop trip count

tipo di analisi che risponde a domande tipo quante volte itera il mio loop, ma anche cose piu sofisticate tipo l'evoluzione di scalari durante il loop

se riesco a farne unanalisi statica riesco a tirarne fuori ottimizzazioni particolarmente importanti e utili?

esemipio slide scalar evolution

scalar evolution

da esempio: ci sono casi in cui i valori intermedi non sono mai usati, e dunque possiamo determinare il valore finale dello scalare senza nemmeno eseguire il loop

vedi slide successive con esempi di llvm ir ssa

la scalar evolution analysis ragiona in termini di **ricorrenze**: esempio tipico numero di fibonacci

dunque cerca di identificare le ricorrenze, definite come una tripla : valore di base, operatore applicato, valore da ?

le catene possono anche essere concatenate

studiando quetso genere di espressioni siamo ingrado di capire come evolve la variabile ed evenutlamente modificare il loop (semplificare)

dall'esempio ci si rende conto che il valore di tk è esattamente tk al di fuori del loop, e lo posso descrivere come k * n (?), allora posso direttamente sostituire con una mul l'operazione

possiamo recuperare i dati che ci servono da questa analisi tramite `AM.getResult<ScalarEvolutionAnalysis>(F)`

in particolare cosa dice la docu per il tripcount e numero di volte che incontriamo preheader

passo 4: dipendenze a distanza negativa

suggerimento di 2 strumenti:

- scalar evolution: analizzando le ricorrenze permette di verificare se ci sono distanze negative nell'accesso al target
- dependence analysis

dependence analysis

`AM.getResult<DependenceAnalysis>(F)`

abbiamo a disposizione primitive di tipo `depends` per controllare possibili dipendenze tra istruzioni

attenzione all'algoritmo usato per analizzare le dipendenze tra loop! approccio bruteforce diventa molto pesante, dunque valuta bene **quali sono le coppie da analizzare**

generazione del codice

operazioni di base da fare

1. modifico usi di induction variable tra i 2 loop (sono diverse tra i 2 loop)
2. modifico il cfg tra i blocchi dei 2 loop

ovviamente in questo modo avremo dei blocchi morti, scollegati dal cfg principale - **non serve che facciamo noi nulla per eliminarli, anche perche basterebbe usare passi appositi di sanificazione del cfg**