# Assignment #2 - Implement a CSE LLVM pass

R04922067 楊翔雲

1. 問題描述
   1. 環境就如同上一次作業格式。
   2. 實作 Common Sub-expression Elimination (CSE) 修改所有 3-address 的指令，也就是在 LLVM 的 Static Single Assignment (SSA) 格式下，移除重覆的 Binary Operator 等相關的重覆計算，用先前處理過 Value Label 直接取代。
   3. 如下述程式碼中，(a + b) 計算會出現兩次，在還沒做任何的最佳化之前，LLVM IR 會將其翻譯成兩段 load/store 片段來完成。事實上 %add2 = %add1，如果能找到這些按例就能大幅度地減少 load 得成本。

int main(void) {

int a = 30;

int b = 40;

int c = a + b + 10;

int d = (a + b) \* c;

printf("%d\n", c + d);

return 0;

}

store i32 0, i32\* %retval

store i32 30, i32\* %a, align 4

store i32 40, i32\* %b, align 4

%0 = load i32, i32\* %a, align 4

%1 = load i32, i32\* %b, align 4

%add = add nsw i32 %0, %1

%add1 = add nsw i32 %add, 10

store i32 %add1, i32\* %c, align 4

%2 = load i32, i32\* %a, align 4

%3 = load i32, i32\* %b, align 4  
%add2 = add nsw i32 %2, %3

* 1. 由於 load/store 造成的不確定性，不能隨意傳遞看似等價的 Value Label，有時候必須 reload 資料進行計算才會正確。因此，在實驗開始前，考慮先經由 -mem2reg 將不必要的 load/store 大幅度地移動到 register 使用，避免重複的 load 操作，導致在撰寫 -MyCSE 時算法功用無法發揮。

1. 算法設計
   1. 對於每個 binary instruction 都必須查找等價關係，類似 union-find 的方法解決，重新為每一個 instruction 的每一個 Value 標記一個獨特的 ID。
   2. 賦予 <x.type, op, y, z> 成為一個計算狀態，如果存在重複狀態 <a.type, op, b, c> = <x.type, op, y, z>，那麼直接將所有 Label a 都抽換成 x。
   3. 在轉換 x = y <op> z 時，必須將 y, z 也找到等價類的標記，這裡採用遞迴解決，如果 y 仍然是個 binary instruction 就一直抽換，直到它是一個 Value，或者並非 binary instruction，就成為新的 ID，這用來避開 load/store instruction。
   4. 簡單的運行流程如下：

Relabel(Value):

If Value exists in ValueTable

return ValueTable[Value]

If Value is not a instruction

return ValueTable[Value] = create\_newlabel

If Value is a binary instruction

e = ExpressionToState(I)

If e not in ExpressionTable

ExpressionTable[e] = create\_newlabel

return ExpressionTable[Value] = create\_newlabel

ExpressionToState(I)

state = <I.op, I.type, Relabel(I.y), Relabel(I.z)>

Return state

Main(Block):

Create ExpressionTable

for each operation I in Block

if I is binary operation

state = ExpressionToState(I)

if state exists in ExpressionTable as J

I replace With J

1. 測試流程
   1. 根據 ~/LLVM/hw2/test 下的腳本進行環境建置後，並執行 LLVM pass。

cd ~/LLVM/hw2/test

sh test.sh

sh clean.sh

* 1. 一律經過 –mem2reg 產生的 bit code，在經由撰寫的 –MyCSE PASS 產生 bit code 相互比較。

1. 效能測試benchmark
   1. 測試範例 (一)  
        
        
      成功將 %add2 移除。

int main(void) {

int a = 30;

int b = 40;

int c = a + b + 10;

int d = (a + b) \* c;

printf("%d\n", c + d);

return 0;

}

; -mem2reg

%add = add nsw i32 30, 40

%add1 = add nsw i32 %add, 10

%add2 = add nsw i32 30, 40

%mul = mul nsw i32 %add2, %add1

%add3 = add nsw i32 %add1, %mul

; -mem2reg -MyCSE

%add = add nsw i32 30, 40

%add1 = add nsw i32 %add, 10

%add2 = add nsw i32 30, 40

%mul = mul nsw i32 %add, %add1

%add3 = add nsw i32 %add1, %mul

* 1. 測試範例 (四)

int main(void){

int i, j, x, y, z, a, b, c, d, e;

unsigned int checksum = 0;

a = 10; b = 10; c = 10; d = 1; e = 1;

for (i = 0; i < 5000; i += 1){

for (j = 0; j < 50000; j += 1){

x = a \* b / c;

y = a \* b / c + d;

z = a \* b / c + d - e;

checksum += x + y + z;

}

a += 1; b += 1; c += 1; d += 1; e += 1;

}

printf("%u\n", checksum);

return 0;

}

* 1. 測試範例 (五)

#define n 1024

double a[n\*n], b[n\*n];

int main() {

int i, j, k;

for (i = 1; i < n-1; i++) {

for (j = 1; j < n-1; j++) {

b[i\*n+j] = (

5 \* a[i\*n+j] + a[(i-1)\*n+j] + a[i\*n+(j-1)]

+ a[(i+1)\*n+j] + a[i\*n+(j+1)]

)/9;

}

}

return 0;

}

* 1. 實驗結果

|  |  |  |
| --- | --- | --- |
| Test Case | MyCSE Replace Instruction | #BasicBlock |
| Case 1 | 1 | 1 |
| Case 2 | 1 | 5 |
| Case 3 | 1 | 5 |
| Case 4 | 5 | 9 |
| Case 5 | 4 | 9 |

* 1. 運行比較 perf stat ./execute

|  |  |  |
| --- | --- | --- |
| Test Case | -mem2reg (insts/sec) | -mem2reg –MyCSE (insts/sec) |
| Case 1 | 461,125 | 449,583 |
| Case 2 | 406,772 | 406,732 |
| Case 3 | 411,836 | 404,163 |
| Case 4 | 10,503,181,458 | 7,252,049,840 |
| Case 5 | 52,044,279 | 45,764,087 |

1. 結語

從實驗結果發現替換的 instruction 數量不多，但隨著 reference 的數量增加，運行效果就相當顯著。

1. 參考
   1. CSE in LLVM, Loop transformations in Pluto and Polly  
      <http://www.cs.colostate.edu/~cs553/Assignments/PA2/PA2-writeup.html>