SWPP Idea Outline

Team 5

컴퓨터공학부 김윤식, 김재윤, 장준서, 정용환

Outline

- Arithmetic optimization
  - Integer `add` to `sum`
  - Other arithmetic operations optimizations
- Branch optimization
- Asynchronous load
- Oracle

Utilize `LLVM` default optimization

- Total cost include `# instructions`
- Eliminate unnecessary code
    - Utilize LLVM default optimization `-adce`
- Constant propagation
    - Utilize LLVM default flag `-sccp`

Integer `add` to `sum`

- Reduce up to 30 costs in integer `add`
    - Integer `add` cost 5
    - Integer `sum` cost 10
- Identify `add` that can be grouped
- Reorder instruction if necessary

```
r5 = add r1, r2

r6 = add r3, r4

; Other operations…

r8 = add r5, r6
```

Integer `add` to `sum`

- Reduce up to 30 costs in integer `add`
  - Integer `add` cost 5
  - Integer `sum` cost 10
- Identify `add` that can be grouped
- Reorder instruction if necessary

```
r5 = add r1, r2

r6 = add r3, r4

; Other operations…

r8 = sum r1, r2, r3, r4
```

Integer `add` to `sum`

- Reduce up to 30 costs in integer `add`
  - Integer `add` cost 5
  - Integer `sum` cost 10
- Identify `add` that can be grouped
- Reorder instruction if necessary
- Not always cheaper cost
  - First modify it and check the cost

```
r5 = add r1, r2

r6 = add r3, r4 // need it

; Other operations…

r7 = sub r6, r1

r8 = sum r1, r2, r3, r4 // same cost
```

Other arithmetic optimization

- Replace `add` to `increment` or `mul`
  - `add r1, r1` → `mul r1, 2`
  - `add r1, 2` → `r2 = incr r1; incr r2;`
- Replace `sub` to `decrement`
  - `sub r1, 1` → `decr r1`
- Replace `shift` to `mul`
  - `shl r1, 2` → `mul r1, 4`

Branch optimization : if else

- find if-else with integer equality
- average case cost : 6

```
if(a==1) else if(a==2) else if(a==3) else

 cond1:
      cmp = icmp seq a 1 32
      br cmp bb1 cond2
 cond2:
      cmp = icmp seq a 2 32
      br cmp bb2 cond3
 cond3:
      cmp = icmp seq a 3 32
      br cmp bb3 bb4
```

Branch optimization : if else

```
if(a==1) else if(a==2) else if(a==3) else
```

- find if-else with integer equality
- change it to switch (cost : 4)

```
cond:

    switch a 1 2 3 bb1 bb2 bb3 bb4
```

Branch optimization : loop

```
while(1 < n) {...}
```

- Branch cost is cheaper in false case
- In loop (while, for) the number of expected true br is more than that of false br.

```
cond:

    cmp = icmp slt i n 32 // true

    br cmp body end       // 6
```

Branch optimization : loop

```
while(1 < n) {...}
```

- Branch cost is cheaper in false case
- In loop (while, for) the number of expected true br is more than that of false br.
- Change that repeat when false
- How to find loop?
  - using control flow graph
  - LoopInfo : https://llvm.org/docs/LoopTerminology.html#loopinfo

```
cond:

    cmp = icmp sge i n 32 // false

    br cmp end body       // 1
```

Asynchronous Load

`load → aload`

- Cost of instructions between loading and using is more than 5
- Overwriting on register without using

```
store 8 3 r3

...

r2 = load 8 r1

call write r2
```

Asynchronous Load

`load → aload`

- Cost of instructions between loading and using is more than 5
- Overwriting on register without using
- find all `load` in basic block and move it to front as possible.
- using dependency between instructions.

```
r2 = aload 8 r1

store 8 3 r3

...

call write r2
```

Oracle function

`load/store → oracle`

- If there are many load and store operations in a row, calling an oracle function can be cheaper

```
store 8 1 r1
store 8 2 r2
store 8 3 r3
call oracle 7
```

```
start oracle 1
existing_block:
    (existing code)
end oracle
```

Oracle function

`load/store → oracle`

- If there are many load and store operations in a row, calling an oracle function can be cheaper
- Replace with oracle function call with unique label assigned to each replacement
- Add one more argument to oracle function if it already exists, set the extra argument to 0 in all already existing oracle function calls

```
call oracle 0 1
call oracle 7 0


start oracle 2
new_block0:
     (code that uses arg2 to
     branch to appropriate block)
existing_block:
     (existing code)
new_block1:
     (code that performs the
     replaced operations)
     ret
end oracle
```

Oracle function

```
load/store → oracle
```

- If there are many load and store operations in a row, calling an oracle function can be cheaper
- Replace with oracle function call with unique label assigned to each replacement
- Add one more argument to oracle function if it already exists, set the extra argument to 0 in all already existing oracle function calls
- Do this with the blocks with more load/store instructions at higher priority until before oracle function exceeds 50 instructions

```
call oracle 0 1
call oracle 7 0


start oracle 2
new_block0:
     (code that uses arg2 to
     branch to appropriate block)
existing_block:
     (existing code)
new_block1:
     (code that performs the
     replaced operations)
     ret
end oracle
```