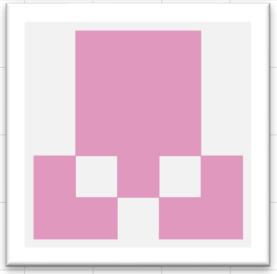
LLVM COMPILER OPTIMIZE PLAN

SWPP TEAM 5

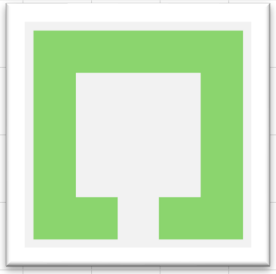
TEAM 5 MEMBERS

김승효

이현우



장호림



정다운

LEARNING GOALS OF THIS PROJECT

1. *BASICS OF LLVM*

2. *IR OPT PASS LOGICS*

3. *HOW TO COLLABORATE*

PASS IMPLEMENTATION PLANS

Loop False Condition	Loop on false conditions instead of true conditions
Simple Arithmetic	Reduce costs in simple arithmetic (**add**, **sub**, **shl**, ...) where possible
Recursive Call	Change **call** to **rcall**
IF/ELSE to Switch	Exploit **switch** instead of if/else
Move Free	Use **free** at an earlier point, reduce peak memory usage
Vector Load/Store	Use vector load/store/(manipulation) to reduce cost in fetching data
Vector Arithmetic	Use vector arithmetic to reduce cost
Vector Parallel Arithmetic	Use vector parallel arithmetic to reduce cost

LOOP ON FALSE CONDITION

```
entry:
  br label %loop.cond

loop.cond:
  %i = phi [ 0, %entry ], [ %next, %loop ]
  %next = incr %i
  %cmp = icmp slt %next, 10
  br %cmp, label %loop.body, label %exit

loop.body:
  ...
  br label %loop.cond

exit:
  ...
```

```
entry:
  br label %loop.cond

loop.cond:
  %i = phi [ 0, %entry ], [ %next, %loop ]
  %next = incr %i
  %cmp = icmp sge %next, 10
  br %cmp, label %exit, label %loop.body

loop.body:
  ...
  br label %loop.cond

exit:
  ...
```

SIMPLE ARITHMETIC

1. add %x %x
2. sub 0 %x
3. shl %x n
4. ashr %x n
5. lshr %x n
6. add %x, 2
7. sub %x, 4

1. mul %x 2
2. mul %x -1
3. mul %x (2^n)
4. sdiv %x (2^n)
5. udiv %x (2^n)
6. incr %x, incr %x
7. decr %x, decr %x, decr %x, decr %x

RECURSIVE CALL

```
define i32 @foo(i32 %n) {
entry:
  %cmp = icmp eq i32 %n, 0
  br i1 %cmp, label %exit, label %recurse

recurse:
  %m = sub i32 %n, 1
  %call = call i32 @foo(i32 %m)
  %result = mul i32 %n, %call
  ret i32 %result

exit:
  ret i32 1
}
```

```
define i32 @foo(i32 %n) {
entry:
  %cmp = icmp eq i32 %n, 0
  br i1 %cmp, label %exit, label %recurse

recurse:
  %m = sub i32 %n, 1
  %call = rcall i32 @foo(i32 %m)
  %result = mul i32 %n, %call
  ret i32 %result

exit:
  ret i32 1
}
```

IF/ELSE TO SWITCH

reg_cond1 = icmp eq <reg> <cst1> <bw1>
reg_cond2 = icmp eq <reg> <cst2> <bw2>
...
reg_condn = icmp eq <reg> <cstn> <bwn>
br <reg_cond1> <true_bb1> <false_bb1>
br <reg_cond2> <true_bb2> <false_bb2>
...
br <reg_condn> <true_bbn> <false_bbn>

switch <reg> <cst1> <true_bb1> <cst2> <true_bb2>
... <cstn> <true_bbn> <false_bbn>

MOVE FREE

```
%a = malloc 64
; some code not using p
free %p
```

```
free %p
; some code not using p
%a = malloc 64
```

VECTOR LOAD/STORE

```
define void @foo() {
  %value1 = load i32, i32* %ptr
  %value2 = load i32, i32* %ptr2
  %value3 = load i32, i32* %ptr3
  %value4 = load i32, i32* %ptr4
  %value5 = load i32, i32* %ptr5
  %value6 = load i32, i32* %ptr6
  %value7 = load i32, i32* %ptr7
  %value8 = load i32, i32* %ptr8
  ... other logic
}
```

```
define void @foo() {
  %vec_value = vload <8 x i32>, <8 x i32>* %ptr
..other logic
}
```

VECTOR ARITHMETIC

```
define void @foo() {
  %sum = add i32 %sum, %value1
  %sum = add i32 %sum, %value2
  %sum = add i32 %sum, %value3
  %sum = add i32 %sum, %value4
  %sum = add i32 %sum, %value5
  %sum = add i32 %sum, %value6
  %sum = add i32 %sum, %value7
  %sum = add i32 %sum, %value8
  ... other logic
}
```

```
define void @foo() {
  %vec_values = vload <8 x i32>, <8 x i32>* %ptr
  %vec_sum = vadd <8 x i32> %vec_sum, %vec_values
  ... other logic
}
```

VECTOR PARALLEL ARITHMETIC (PSEUDO CODE)

```
for (i = 0, i < 1024, i++) {
   for (j = 0, j < 1024, j++) {
      sum = 0.0
      for (k = 0, k < 1024, k++) {
         sum += A[i][k] * B[k][j];
      }
      res[i][j] = sum;
   }
}
```

```
%0 = 0
for (i = 0, i < 1024, i++) {
   for (j = 0, j < 1024, j++) {
      vec1 = vbcast %0 32
      vec2 = vbcst %0 32
      for (k = 0, k < 1024, k+=8) {
         x = vload A[i][k]
         y = vload B[k][j]
         z = vmul x, y
         if (k % 2 == 0) vec1 = vpadd z, vec2
         else vec2 = vpadd z, vec1
      }
      res[i][k] = {sum of elements in vec2}
   }
}
```

THANKS!

SWPP TEAM 5