SWPP2023: optimization ideas

Team 2

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always]. [%val2, %BB_unreachable]
    ret i32 %val
```

Detect Side Effects

- Remove instructions without side effects

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always], [%val2, %BB_unreachable]
    ret i32 %val
```

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always], [%val2, %BB_unreachable]
    ret i32 %val
```

Basic optimizations

- Detect constant expressions

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always]. [%val2, %BB_unreachable]
    ret i32 %val
```

Basic optimizations

- Always-branch optimization

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always]. [%val2, %BB_unreachable]
    ret i32 %val
```

Basic optimizations

- Delete unreachables

```
BB_entry:
    %x1 = add i32 1, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always]. [%val2, %BB_unreachable]
    ret i32 %val
```

Basic optimizations

- Substitute instructions to cheaper equivalents

```
BB_entry:
    %x1 = add i32 1, 1          mul i32 2, 1
    %x2 = mul i32 %x1, 7
    %xcond = icmp sge i32 %x2, 10
    br i1 %xcond, label %BB_always, label %BB_unreachable

BB_always:
    %tmp1 = fadd double 3.14, 2.72
    %val1 = mul i32 1, 1
    br label %BB_end

BB_unreachable:
    %tmp2 = fsub double 1.41, 1.62
    %val2 = mul i32 2, 2
    br label %BB_end

BB_end:
    %tmp = phi double [%tmp1, %BB_always], [%tmp2, %BB_unreachable]
    %val = phi i32 [%val1, %BB_always]. [%val2, %BB_unreachable]
    ret i32 %val
```

Basic optimizations

- Repeat until no more optimization is possible

```
BB_entry:
    br label %BB_always

BB_always:
    %val1 = mul i32 1, 1
    br label %BB_end

BB_end:
    %val = mul i32 %val1, 1
    ret i32 %val
```

How?

```
BB_entry:
    ret i32 1
```

Loop optimizations

- For-loop: set exit condition as true

```
for.init:
  %i.init = mul i32 1, 1
  %sum.init = mul i32 0, 0
  %cond.init = icmp ult i64 %i.init, %n
  br i1 %cond.init, label for.body, label for.end

for.body:
  %i = phi i32 [ %i.next, %for.body ], [ %i.init, %for.init ]
  %sum = phi i32 [ %sum.next, %for.body ], [ %sum.init, %for.init ]
  %sum.next = add i32 %sum, %i
  %i.next = add i32 %i, 1
  %cond = icmp ult i64 %i.next, %n
  br i1 %cond, label for.body, label for.end

for.end:
  %sum.ret = phi i32 [ %sum.next, %for.body ], [ %sum.init, %for.init ]
  ret i32 %sum.ret
```

True for n times, false for only 1 time

```
int sum = 0;
for (int i = 0; i < n; i++) {
    sum += i;
}
return sum;
```

```
for.init:
  %i.init = mul i32 1, 1
  %sum.init = mul i32 0, 0
  %cond.init = icmp uge i64 %i.init, %n
  br i1 %cond.init, label for.end, label for.body

for.body:
  %i = phi i32 [ %i.next, %for.body ], [ %i.init, %for.init ]
  %sum = phi i32 [ %sum.next, %for.body ], [ %sum.init, %for.init ]
  %sum.next = add i32 %sum, %i
  %i.next = add i32 %i, 1
  %cond = icmp uge i64 %i.next, %n
  br i1 %cond, label for.end, label for.body

for.end:
  %sum.ret = phi i32 [ %sum.next, %for.body ], [ %sum.init, %for.init ]
  ret i32 %sum.ret
```

```
int sum = 0;
for (int i = 0; !(i >= n); i++) {
    sum += i;
}
return sum;
```

Loop optimizations

- How to detect loop?

Other Possible Implementations?

- Usage of Oracle
- Put heap into stack