3조 `고계원실` 최적화 계획

고계훈, 고재준, 류지민, 안중원

목차

1. switch문의 활용
2. register 수 부족시 call 활용
3. oracle 사용
4. Bulk load/store
5. async load의 활용
6. Arithmetic cost optimization

1. switch문의 활용

- 선택지의 수에 제한이 없음, variable이 가질 수 있는 값은 이산적(정수 한정)
- 따라서, 비교(ule, ugt 등) 등을 통한 조건문 + 분기문을 switch문에서 범위 내 정수를 선택지에 모두 적으면서 cost 절약

1. switch문의 활용: 예시

before:

```
cmp1:
  %cond1 = icmp ule i32 %r1 10
  br %cond1 %cmp2 %b1
cmp2:
  %cond2 = icmp ule i32 %r1 20
  br %cond2 %b2, %b3
b1:
  ; %r1 <= 10
b2:
  ; 11 <= %r1 <= 20
b3:
  ; 21 <= %r1
```

1. switch문의 활용: 예시

after:

```
cmp1:
  switch %r1, 0 %b1, ..., 10 %b1, 11 %b2, ... 20 %b2, %b3
```

- cost: 조건 n개를 사용할 때, 4.5 × n → 4

2. register 수 부족시 call 활용

- function call시 기존의 register 값들은 hidden space에 저장되고 return시 다시 복구됨
- register 개수가 부족한 경우 stack을 사용하는 대신 비용이 훨씬 덜 드는 call을 활용해 기존의 레지스터 값들을 hidden space로 넘기기
- 이 경우 굳이 scalar 변수에 관해서는 stack을 쓰지 않아도 됨. hidden space가 stack의 기능을 한다.

2. register 수 부족시 call 활용: 예시

before:

```
function main
  ; ...
  %r31 = add %sp, 4
  store %r1, %sp
  %r31 = add %sp, 8
  store %r2, %r31
  ; ...
  ; %r1, %r2를 다른 변수로 사용하고, 결과값이 %r30에 저장됨
  ; ...
  %r31 = add %sp, 4
  %r1 = load %r31
  %r31 = add %sp, 8
  %r2 = load %r31
  ; ...
end
```

2. register 수 부족시 call 활용: 예시

after:

```
function @main
  ; ...
  %r30 = call @__42
  ; ...
end

function @__42
  ; %r1, %r2를 맘대로 사용하고, 결과값을 리턴
  ret %r1
end
```

- cost: 레지스터 n개에 대해 50 × n → 3
- 추가 이점: stack 사용량 감소, 스택 포인터 계산을 위해 써야 했던 레지스터(%r31)를 다른 곳에 사용 가능

3. oracle 사용

- 배열 접근(load, store)을 반복문 내에서 수행할 경우, oracle function 사용
- load, store의 비용이 oracle 내에서 압도적으로 낮음(90% 절감)

3. oracle 사용: 예시

before:

```
define i32 @main() {
  ; ...
for.body:
  %i = phi i32 [ %inc, %for.body ], [ 1, %entry ]
  ; ...
  %num = load i32, i32* %ptr
  %sum = phi i32 [ %newsum, %for.body ], [ 0, %entry ]
  %newsum = add i32 %sum, %num
  ; ...
  %inc = add i32 %i, 1
  %cmp = icmp i32 ult %inc, 10
  br %cmp, %for.body, %for.end
  ; ...
}
```

3. oracle 사용: 예시

after:

```
define i32 @main() {
  %sum = call @oracle(%ptr)
}

define i32 @oracle(%ptr) {
for.body:
  ; load 사용
}
```

- stack n번 사용, heap m번 사용할 때, 20n + 30m → 2n + 3m + 40
- 내부에서 load/store를 2~3번만 하더라도 oracle cost를 넘는 이득

4. Bulk load/store

- 메모리에 여러 바이트를 한 번에 저장하여 cost를 아낌
- mul이 저렴해서 더욱 큰 효과
- 한 번에 같이 사용할 변수를 관리하는 데에도 편함

4. Bulk load/store: 예시

before:

```
store 1 %r1 %sp
%sp = decr %sp
store 1 %r2, %sp
%sp = decr %sp
store 1 %r3 %sp
%sp = decr %sp
store 1 %r4, %sp
%sp = decr %sp
; ...
%sp = incr %sp
%r1 = load 1 %sp
%sp = incr %sp
%r2 = load 1 %sp
%sp = incr %sp
%r3 = load 1 %sp
%sp = incr %sp
%r4 = load 1 %sp
```

4. Bulk load/store: 예시

after:

```
%r4 = mul %r4 2^24
%r3 = mul %r3 2^16
%r2 = mul %r2 2^8
%r1 = sum %r1 %r2 %r3 %r4
store 4 %r1 %sp
%sp = sub %sp 4
; ...
%sp = add %sp 4
load 4 %r1 %sp
%r4 = udiv %r1 2^24
%r3 = udiv %r1 2^16
%r3 = urem %r3 2^8
%r2 = udiv %r1 2^8
%r2 = urem %r2 2^8
%r1 = urem %r1 2^8
```

- cost: 168 → 69

5. async load의 활용

- 실행 자체는 매우 적은 비용이 들며, load하는 동안 다른 instruction 실행 가능
- load을 기다리는 동안 실행하는 instruction의 비용 총합이 5를 넘어가면 비용 측에서 이득을 봄
- 여러 개의 load를 연속으로 사용할 때 용이

5. async load의 활용: 예시

before:

```
%num1 = load i8, i8* %ptr1
%num2 = load i8, i8* %ptr2
%num3 = add i8 %num1, %num2
%num4 = load i8, i8* %ptr4
%num5 = load i8, i8* %ptr5
```

5. async load의 활용: 예시

after:

```
%num1 = aload 1 %ptr1
%num2 = aload 1 %ptr2
%num4 = aload 1 %ptr4
%num5 = aload 1 %ptr5
%num3 = add %num1 %num2 8
```

- cost: heap 사용을 가정할 때, 95 → 41

6. Arithmetic cost optimization

- 특수하게 cost가 낮은 instruction이 있다.
- add가 여러 번이라면 sum을 대체
- shift는 mul 또는 div로 대체

예: add를 7번 하면 cost = 35, sum으로 한 번에 처리하면 cost = 10

```
%reg1 = add %val1 %val2
%reg2 = add %reg1 %val3
...
%reg7 = add %reg6 %val8
```

↓

```
%reg = sum %val1 %val2 ... %val8
```