SWPP 2023
Compiler Optimization Ideas

Team 1 - 박종한 배준익 이경훈 현재익

Introducing our Team

```
struct Team1 {
    member 박종한;  // foxisobese
    member 배준익;  // heatz123
    member 이경훈;  // pzmaus
    member 현재익;  // SyphonArch
}
```

SNUCSE 19 // 3학년

아이디어 1

Lazy Evaluation With SIMD

/* Cluster integer additions into SIMD style operations */

덧셈 연산은 곱셈 연산에 비해 cost가 매우 크다.

이 cost는 int의 bit 너비와 무관하다.

Condition:

- ➢ 반복문 등에서 순서가 중요치 않은, 상대적으로 작은 int형의 덧셈이 여러 번 일어나는 경우

Optimization:

- ➢ 연산을 뒤로 미룸과 동시에 하나의 register로 모아서, 한번의 덧셈으로 처리한다.

Integer Sum	`<reg> = sum <val1> ... <val8> <bw>` `<bw> := 1|8|16|32|64`	10

example

```
%c1 = add i4 %a1, %b1
%c2 = add i4 %a2, %b2
…
%c16 = add i4 %a16, %b16
//store c1 ~ c16


%ptr1 = bitcast [16 * i4]* %a to i64*
%ptr2 = bitcast [16 * i4]* %b to i64*
%val1 = load i64, i64& %ptr1
%val2 = load i64, i64& %ptr2
%c = add i64 %val1, %val2
%ans = bitcast i64 %c to [16 * i4]*
//store
```

아이디어 2

Loop if false

/* Iterate with false brach */

Condition

➢ 루프의 body로 진입하는 branch가 true인 경우

Optimization

➢ icmp 조건을 invert하여 false branch로 루프를 돌도록 한다

Kind	Syntax	Cost
Return Value - `ret` is equivalent to `ret 0`.	`ret` `ret <val>`	1
Unconditional Branch	`br <bbname>`	1
Conditional Branch	`br <condition> <true_bb> <false_bb>`	6 for `true_bb` 1 for `false_bb`
Switch Instruction - `<val1>`, ... should be constant integers.	`switch <cond_val> <val1> <bb1> .. <default_bb>`	4

example

```
define i32 @main() {
  ; Initialize variables
  %i = add i32 1, 0
  %sum = add i32 0, 0

  ; Loop from 1 to 10
  br label %loop_cond

loop_cond:
  %cmp = icmp sle i32 %i, 10
  br i1 %cmp, label %loop_body, label %loop_exit

loop_body:
  %sum_temp = add i32 %sum, %i
  %i_inc = add i32 %i, 1
  br label %loop_cond

loop_exit:
  ; Print the sum
  %format_str = c"Sum: %d\n\00"
  call i32 (i8*, ...) @printf(i8* %format_str, i32 %sum)

  ret i32 0
}
```

```
define i32 @main() {
  ; Initialize variables
  %i = add i32 1, 0
  %sum = add i32 0, 0

  ; Loop from 1 to 10
  br label %loop_cond

loop_cond:
  %cmp = icmp sgt i32 %i, 10
  br i1 %cmp, label %loop_exit, label %loop_body

loop_body:
  %sum_temp = add i32 %sum, %i
  %i_inc = add i32 %i, 1
  br label %loop_cond

loop_exit:
  ; Print the sum
  %format_str = c"Sum: %d\n\00"
  call i32 (i8*, ...) @printf(i8* %format_str, i32 %sum)

  ret i32 0
}
```

아이디어 3

Registers over Stack over Heap

/* Heap memory is extra expensive! */

Heap의 cost는 바이트당 1024이며, load/store cost가 추가로 있다.

Stack은 cost가 없으며, load/store cost가 있다.

Register는 cost도 없고, load/store overhead도 없다!

스펙에 의해 33개의 general purpose register가 있으므로, 가능하다면 이를 최대한 활용한다.

Memory usage cost.

- The memory usage cost is 1024 times the maximum heap-allocated byte size at any moment.
- For example, the memory usage cost of

```
r1  = malloc 8
free r1
r2 = malloc 8
free r2
```

is 1024 * 8 = 8192, because the maximum memory usage is 8 bytes.

example

```
define i32 @main() {
  ; Allocate a block of 16 bytes on the heap and get a pointer to it
  %ptr = call i8* @malloc(i64 16)

  ; Cast the pointer to i32* so we can store integers in it
  %array_ptr = bitcast i8* %ptr to i32*

  ; Store integers in the four 4-byte chunks
  store i32 1, i32* %array_ptr
  store i32 2, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 1)
  store i32 3, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 2)
  store i32 4, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 3)

  ; Add up the integers in the array
  %sum1 = load i32, i32* %array_ptr
  %sum2 = load i32, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 1)
  %sum3 = load i32, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 2)
  %sum4 = load i32, i32* getelementptr inbounds (i32, i32* %array_ptr, i64 3)
  %sum = add i32 %sum1, %sum2
  %sum = add i32 %sum, %sum3
  %sum = add i32 %sum, %sum4

  ; Print the sum
  %format_str = c"Sum: %d\n\00"
  call i32 (i8*, ...) @printf(i8* %format_str, i32 %sum)

  ; Free the allocated memory
  call void @free(i8* %ptr)

  ret i32 0
}
```

```
define i32 @main() {
  ; Store integers in registers
  %num1 = alloca i32
  %num2 = alloca i32
  %num3 = alloca i32
  %num4 = alloca i32
  store i32 1, i32* %num1
  store i32 2, i32* %num2
  store i32 3, i32* %num3
  store i32 4, i32* %num4

  ; Add up the integers
  %sum1 = load i32, i32* %num1
  %sum2 = load i32, i32* %num2
  %sum3 = load i32, i32* %num3
  %sum4 = load i32, i32* %num4
  %sum = add i32 %sum1, %sum2
  %sum = add i32 %sum, %sum3
  %sum = add i32 %sum, %sum4

  ; Print the sum
  %format_str = c"Sum: %
  call i32 (i8*, ...) @p

  ret i32 0
}
```

```
define i32 @main() {
  ; Store integers in registers
  %num1 = add i32 1, 0
  %num2 = add i32 2, 0
  %num3 = add i32 3, 0
  %num4 = add i32 4, 0

  ; Add up the integers
  %sum = add i32 %num1, %num2
  %sum = add i32 %sum, %num3
  %sum = add i32 %sum, %num4

  ; Print the sum
  %format_str = c"Sum: %d\n\00"
  call i32 (i8*, ...) @printf(i8* %format_str, i32 %sum)

  ret i32 0
}
```

아이디어 4

Prefer mul/div over shifting

/* Never use shifts */

Shifting보다는 *2^n, /2^n을 이용

Shifting cost = 4

Mul / div cost = 1

Const 2^n을 곱하거나 나누는 방식이 더 효율적

Kind	Name	Cost

Integer Multiplication/Division	`<reg> = udiv <val1> <val2> <bw>`
	`<reg> = sdiv <val1> <val2> <bw>`
	`<reg> = urem <val1> <val2> <bw>`
	`<reg> = srem <val1> <val2> <bw>`
	`<reg> = mul  <val1> <val2> <bw>`
	`<bw> := 1|8|16|32|64`
		1

Integer Shift/Logical Operations
- shl: shift-left
- lshr: logical shift-right
- ashr: arithmetic shift-right
	`<reg> = shl  <val1> <val2> <bw>`
	`<reg> = lshr <val1> <val2> <bw>`
	`<reg> = ashr <val1> <val2> <bw>`
	`<reg> = and  <val1> <val2> <bw>`
	`<reg> = or   <val1> <val2> <bw>`
	`<reg> = xor  <val1> <val2> <bw>`
	`<bw> := 1|8|16|32|64`
		4

example

```
%after = shl i32 %before, 2
%after = shr i32 %before, 2
```

```
%after = mul i32 %before, 4
%after = udiv i32 %before, 4
```

아이디어 5

Aload when possible

/* Load is expensive, aload is cheap */

착안점

Load의 cost는 20/30, async load의 cost는 24/34이지만, waiting을 최소화하면 cost를 1로 줄일 수 있다.

Kind	Syntax	Base Cost
Load	`<reg> = load <size> <ptr>` `<size> := 1|2|4|8`	Stack area: 20 Heap area: 30 Cost reduced by 90% inside oracle
Store	`store <size> <val> <ptr>` `<size> := 1|2|4|8`	Stack area: 20 Heap area: 30 Cost reduced by 90% inside oracle
Async Load	`<reg> = aload <size> <ptr>` `<size> := 1|2|4|8`	Stack area: 1 Heap area: 1 Cost to resolve Stack area: 24 Heap area: 34 Cannot use inside oracle

example

예시 코드

```
r2 = aload 4 r1
r3 = icmp eq r2 5
r12 = add r10 r11 32
r13 = add r11 r12 32
r11 = sub r11 r10 32
r11 = mul r10 r11 32
r10 = incr r10
br r3 true_bb false_bb
```

Async load의 cost: 24 -> 24-5-5-5-1-1

```
r2 = aload 4 r1
r12 = add r10 r11 32
r13 = add r11 r12 32
r11 = sub r11 r10 32
r11 = mul r10 r11 32
r10 = incr r10
r3 = icmp eq r2 5 ; deferred
br r3 true_bb false_bb
```

아이디어 6

Use the Oracle

/* The Oracle makes load/stores cheap */

Oracle 함수의 특징은

- load/store가 저렴하며
- (함수이므로) 리턴값이 하나인 것이다

Condition

➢ 여러번의 load/store 끝에 결과값이 store되거나, 하나의 i64값으로 줄어드는 경우

Optimization

➢ Oracle 내부로 로직을 이동시킨다. (Function inlining의 반대 느낌)

필요 시, oracle들을 chaining하여, store/load를 통해 workload를 이어받을 수 있다.

(1-1) Oracle Function

Syntax:

```
    start oracle <Narg>:
      ... (basic blocks)
    end oracle
```

- A function named `oracle` is treated specially.
- Unlike other functions, `call oracle` always costs 40 regardless of the number of arguments
- In this function, all `load` / `store` cost only 10% of its original cost (reduced by 90%)
- The interpreter will crash if `call` is used inside `oracle`
- The interpreter will crash if `aload` is used inside `oracle`
- The compiler will crash if `oracle` contains more than 50 LLVM IR instructions (excluding basic block labels)

example

```
define i32 @sum4(i32* %ptr1, i32* %ptr2, i32* %ptr3, i32* %ptr4) {
  ; Load values from pointers
  %val1 = load i32, i32* %ptr1
  %val2 = load i32, i32* %ptr2
  %val3 = load i32, i32* %ptr3
  %val4 = load i32, i32* %ptr4

  ; Calculate sum
  %sum = add i32 %val1, %val2
  %sum = add i32 %sum, %val3
  %sum = add i32 %sum, %val4

  ; Return sum
  ret i32 %sum
}
```

```
define i32 @sum4(i32* %ptr1, i32* %ptr2, i32* %ptr3, i32* %ptr4) {

  ; Chain oracles together
  %result = call i64 @oracle(...) ; save temp results to memory
  %result = call i64 @oracle(...) ; load from memory, and continue
  %result = call i64 @oracle(...) ; and so on
  %result = call i64 @oracle(...)

  ; Return sum
  ret i32 %result
}
```

아이디어 7

Optimize in code level

/* Basic optimizations: CSE, DCE */

Common Subexpression Elimination

이전에 연산했던 expression 이 있다면 재사용한다.

Dead Code Elimination

Reachable 하지 않은 영역을 제거한다.

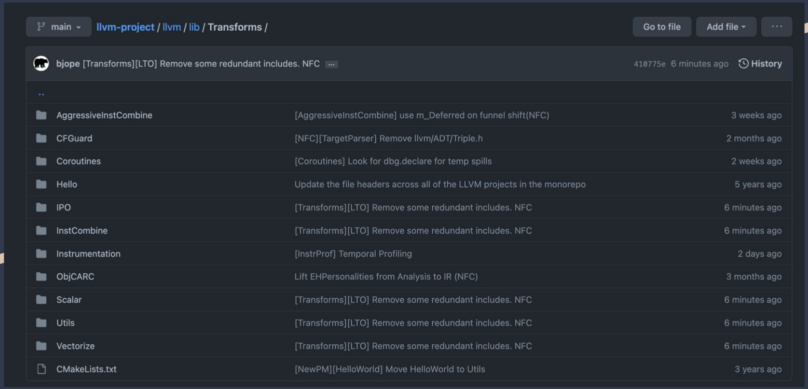
example

```
define i32 @example_function(i32 %a, i32 %b) {
  %c = add i32 %a, %b         ; Addition operation
  %d = mul i32 %c, 3         ; Multiplication operation
  %e = add i32 %c, %d        ; Redundant addition operation
  %f = mul i32 %e, %b        ; Unused multiplication operation
  ret i32 %f                 ; Return value
}

define i32 @example_function(i32 %a, i32 %b) {
  %c = add i32 %a, %b         ; Addition operation
  %d = mul i32 %c, 3         ; Multiplication operation
  %f = mul i32 %c, %b        ; Optimized multiplication operation
  ret i32 %f                 ; Return value
}
```

Other ideas...

https://github.com/llvm/llvm-project/tree/main/llvm/lib/Transforms



main llvm-project / llvm / lib / Transforms / Go to file Add file

bjope [Transforms][LTO] Remove some redundant includes. NFC 410775e 6 minutes ago History

..		
AggressiveInstCombine	[AggressiveInstCombine] use m_Deferred on funnel shift(NFC)	3 weeks ago
CFGuard	[NFC][TargetParser] Remove llvm/ADT/Triple.h	2 months ago
Coroutines	[Coroutines] Look for dbg.declare for temp spills	2 weeks ago
Hello	Update the file headers across all of the LLVM projects in the monorepo	5 years ago
IPO	[Transforms][LTO] Remove some redundant includes. NFC	6 minutes ago
InstCombine	[Transforms][LTO] Remove some redundant includes. NFC	6 minutes ago
Instrumentation	[InstrProf] Temporal Profiling	2 days ago
ObjCARC	Lift EHPersonalities from Analysis to IR (NFC)	3 months ago
Scalar	[Transforms][LTO] Remove some redundant includes. NFC	6 minutes ago
Utils	[Transforms][LTO] Remove some redundant includes. NFC	6 minutes ago
Vectorize	[Transforms][LTO] Remove some redundant includes. NFC	6 minutes ago
CMakeLists.txt	[NewPM][HelloWorld] Move HelloWorld to Utils	3 years ago

Loop unrolling

Function inlining

And other general optimization ideas are all possible