ASO LAB Seminar
1
LLVM Tutorial

엄소은

CONTENTS

1. LLVM

LLVM Project

- collection of modular and reusable compiler and toolchain
- More than just a compiler, provides framework for code analysis, transformation, optimization
- LLVM Compiler Infrastructure
  1. Clang에 의한 AST(추상 구문 트리) 생성
  2. LLVM IR 로의 전환
  3. 기계 코드로의 변환
- LLVM Optimization techniques

1. LLVM

Step1 : IR 코드 포맷 (*.ll,*. bc)

- *.ll(LLVM Assembly Language) : 사람이 읽을 수 있는 텍스트 포맷, IR을 어셈블리 언어와 유사한 형태로 표현한 것
- *.bc(LLVM bitcode) : 바이너리 포맷, 컴퓨터가 빠르게 읽고 쓸 수 있는 형태로 저장된 LLVM.
- llvm-as, llvm-dis 를 사용하여 상황에 따라 *.ll <-> *.bc 변환 가능
- llc(LLVM backend compile)로 LLVM bitcode를 머신 어셈블리로 변환

1. LLVM

Step1 : LLVM IR을 명령어 수준에서 처리

- cpp 로 작성한 ReadIR.cpp 를 컴파일 후 HelloWorld.ll 과 HelloWorld.bc 를 ReadIR 내에서 로드해보기

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_1 (0.087s)
./ReadIR HelloWorld.ll

Success reading & parsing the IR file.
The module name is "HelloWorld.ll"
```

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_1 (0.084s)
./ReadIR HelloWorld.bc

Success reading & parsing the IR file.
The module name is "HelloWorld.bc"
```

1. LLVM

Step2 : LLVM Module Structure

- llvm::Module -> llvm::Function -> llvm::BasicBlock -> llvm:::Instruction 계층 구조
- 전역변수(llvm::GlobalVariable)은 llvm::Module 내에서 따로 관리

```
unmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial
./PrintInst Test.ll
func1
        %1 = alloca i32, align 4
        store i32 4, i32* %1, align 4
        %2 = load i32, i32* %1, align 4
        ret i32 %2
main
        %1 = alloca i32, align 4
        store i32 0, i32* %1, align 4
        ret i32 0
```

1. LLVM

Step3 : 프로그램 내 특정 명령어의 수를 세는 정적 프로파일러

- ADD opcode 세기 (최적화 전)

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_3 (0.123s)
./CountInst ./Test.ll
The Number of ADD  Instructions in the Module ./Test.ll is 7
```

- ADD opcode 세기 (-O3 으로 높은 수준 최적화 적용)
  - Loop Unrolling

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_3 (0.162s)
clang -O3 -emit-llvm -S Test.c -o Test.optimized.ll
```

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_3 (0.101s)
./CountInst ./Test.optimized.ll
The Number of ADD  Instructions in the Module ./Test.optimized.ll is 22
```

1. LLVM

Step3 : 프로그램 내 특정 명령어의 수를 세는 정적 프로파일러

- 명령어 세기
  - 최적화 ll 은 function inlining 이 된 것을 볼 수 있음

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_3 (0.112s)
./CountInst ./Test.ll

Function call counts in the module ./Test.ll:
Function FuseAddMul was called 1 times.
Function VectorAdd was called 1 times.
Function init was called 1 times.
Function printf was called 1 times.
```

```
uhmturks@CASSLAB-Server15 ~/tutorial/llvm-tutorial/Step_3 (0.109s)
./CountInst ./Test.optimized.ll

Function call counts in the module ./Test.optimized.ll:
Function printf was called 10 times.
```

1. LLVM

Step4 : IR 레벨에서 프로그램 내의 명령어 삭제 / 새로 생성하여 삽입

- ADD 명령어를 삭제하려면,

  1) ADD 명령어 이전에 SUB 명령어가 수행되도록 변경

  2) 생성한 Sub 명령어가 Add 명령어가 사용되는 곳에 대신하여 사용되도록 코드를 변경

  3) ADD 명령어 삭제

- (2)번 단계를 생략하면,<badref> 오류가 발생 -> add한 결과물을 다른 곳에 저장하도록 해야 함 !!

```
clang ./Test.Error.ll -o TestError

./Test.Error.ll:31:19: error: expected type
  %17 = sext i32 <badref> to i64
                 ^

1 error generated.
```

```
  %16 = load i32, i32* %15, align 4
  %subtmp = sub i32 %12, %16
  %17 = sext i32 <badref> to i64
  %18 = load i64, i64* %8, align 8
  %subtmp1 = sub i64 %17, %18
  %19 = trunc i64 <badref> to i32
  %20 = load i32*, i32** %7, align 8
  %21 = load i64, i64* %8, align 8
  %22 = getelementptr inbounds i32, i32* %20, i64 %21
  store i32 %19, i32* %22, align 4
  %23 = load i32*, i32** %7, align 8
  %24 = load i64, i64* %8, align 8
  %25 = getelementptr inbounds i32, i32* %23, i64 %24
  %26 = load i32, i32* %25, align 4
  ret i32 %26
}
```

1. LLVM

Step4 : IR 레벨에서 프로그램 내의 명령어 삭제 / 새로 생성하여 삽입

- A+B+C -> (A*B)-C 로 변경해보기

```
  %16 = load i32, i32* %15, align 4
  %17 = add nsw i32 %12, %16
  %18 = sext i32 %17 to i64
  %19 = load i64, i64* %8, align 8
  %20 = add i64 %18, %19
  %21 = trunc i64 %20 to i32
  %22 = load i32*, i32** %7, align 8
  %23 = load i64, i64* %8, align 8
  %24 = getelementptr inbounds i32, i32* %22, i64 %23
  store i32 %21, i32* %24, align 4
  %25 = load i32*, i32** %7, align 8
  %26 = load i64, i64* %8, align 8
  %27 = getelementptr inbounds i32, i32* %25, i64 %26
  %28 = load i32, i32* %27, align 4
  ret i32 %28
}
```

Test.ll

```
29   %16 = load i32, i32* %15, align 4
30   %multmp = mul i32 %12, %16      // A*B
31   %17 = sext i32 %multmp to i64   // sign extend %17
32   %18 = load i64, i64* %8, align 8  // Load C
33   %subtmp2 = sub i64 %17, %18     // (A*B) - C
34   %19 = trunc i64 %subtmp2 to i32
35   %20 = load i32*, i32** %7, align 8
36   %21 = load i64, i64* %8, align 8
37   %22 = getelementptr inbounds i32, i32* %20, i64 %21
38   store i32 %19, i32* %22, align 4
39   %23 = load i32*, i32** %7, align 8
40   %24 = load i64, i64* %8, align 8
41   %25 = getelementptr inbounds i32, i32* %23, i64 %24
42   %26 = load i32, i32* %25, align 4
43   ret i32 %26
44 }
```

newTest.ll

1. LLVM

Step4 : IR 레벨에서 프로그램 내의 명령어 삭제 / 새로 생성하여 삽입

- A+B+C -> (A*B)-C 로 변경해보기

```
./newTest

Input N (0~5)
3
...(4 MUL 5) SUB 3 = 17
```

1. LLVM

Step5 : 명령어 메모리 종속성 파악

- load/store 의 종속성 문제는 컴파일러 단계에서 알기 어려움
- store 명령어들을 마지막 명령어 이전의 위치로 이동시키면 컴파일은 정상적으로 되지만 결과값 오류 발생

```
./Test
a(3+5) = 8, b(a+4) = 12

uhmturks@CASSLAB-Server15 ~/tutorial
./Test.Processed
a(3+5) = 0, b(a+4) = 32766
```

1. LLVM

Project1 : 수학적 최적화 (A+0, A*1 삭제)

- Opcode == Add 일때, operand가 0이면 교체
- Opcode == Mul 일 때, operand가 1이면 교체

```
for( llvm::BasicBlock::iterator BBIter = BB->begin(); BBIter != BB->end(); ++BBIter )
{
    llvm::Instruction* Inst = llvm::cast<llvm::Instruction>(BBIter);

    if (Inst->getOpcode() == llvm::Instruction::Add) {
        if (isConstantIntZero(Inst->getOperand(0))) {
            // e.g) %6 = add i32 0, %5 이면 %6을 사용하는 모든 operand 를 %5로 교체해야 한다.
            Inst->replaceAllUsesWith(Inst->getOperand(1));
            ToRemove.push_back(Inst);
        } else if (isConstantIntZero(Inst->getOperand(1))) {
            Inst->replaceAllUsesWith(Inst->getOperand(0));
            ToRemove.push_back(Inst);
        }
    }

    if (Inst->getOpcode() == llvm::Instruction::Mul) {
        if (isConstantIntOne(Inst->getOperand(0))) {
            Inst->replaceAllUsesWith(Inst->getOperand(1));
            ToRemove.push_back(Inst);
        } else if (isConstantIntOne(Inst->getOperand(1))) {
            Inst->replaceAllUsesWith(Inst->getOperand(0));
            ToRemove.push_back(Inst);
        }
    }
}
```

```
for (int i = 0, Size = ToRemove.size(); i < Size; ++i) {
    ToRemove[i]->eraseFromParent();
}
```

1. LLVM

Project1 : 수학적 최적화 (A+0, A*1 삭제)

- 불필요한 명령어 삭제됨

```
  %4 = alloca float, align 4
  store i32 0, i32* %1, align 4
  store i32 3, i32* %2, align 4
  store i32 4, i32* %3, align 4
  store float 3.000000e+00, float* %4, align 4
  %5 = load i32, i32* %2, align 4
  %6 = add nsw i32 %5, 0 ;remove
-------------------------------------------------------------------------------
  store i32 %6, i32* %2, align 4
  %7 = load i32, i32* %2, align 4
  %8 = add nsw i32 0, %7 ;remove
  store i32 %8, i32* %2, align 4
  %9 = load i32, i32* %3, align 4
  %10 = mul nsw i32 %9, 1 ; remove
  store i32 %10, i32* %3, align 4
  %11 = load i32, i32* %3, align 4
  %12 = mul nsw i32 1, %11 ;remove
  store i32 %12, i32* %3, align 4
  %13 = load float, float* %4, align 4
  %14 = fmul float %13, 1.000000e+00
  store float %14, float* %4, align 4
  ret i32 0
}
```

```
11  %4 = alloca float, align 4
12  store i32 0, i32* %1, align 4
13  store i32 3, i32* %2, align 4
14  store i32 4, i32* %3, align 4
15  store float 3.000000e+00, float* %4, align 4
16  %5 = load i32, i32* %2, align 4
17  store i32 %5, i32* %2, align 4
18  %6 = load i32, i32* %2, align 4
19  store i32 %6, i32* %2, align 4
20  %7 = load i32, i32* %3, align 4
21  store i32 %7, i32* %3, align 4
22  %8 = load i32, i32* %3, align 4
23  store i32 %8, i32* %3, align 4
24  %9 = load float, float* %4, align 4
25  %10 = fmul float %9, 1.000000e+00
26  store float %10, float* %4, align 4
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
    -------------------------------------------------------------------------------
27  ret i32 0
28 }
29
```

1. LLVM

Project2 : 동적 프로파일러 개발

- 동적으로 실제로 실행되는 명령어 카운트
- add_inst_count는 전역변수로 모듈에서 관리
- 현재 명령어 이전으로 SetInsertPoint
- add_inst_count load한것과 1을 더해서 저장

```
./Test.Processed
Number of ADD : 20
```

```cpp
// Traverse Instructions in TheModule
void TraverseModule(void)
{
    llvm::IRBuilder<> Builder(*TheContext);
    llvm::Type* i32Type = llvm::Type::getInt32Ty(*TheContext);
    llvm::GlobalVariable* addInstCount = TheModule->getNamedGlobal("add_inst_count");

    for (auto &F : *TheModule)
    {
        for (auto &BB : F)
        {
            for (auto &I : BB)
            {
                if (llvm::isa<llvm::BinaryOperator>(I) && I.getOpcode() == llvm::Instruction::Add)
                {
                    Builder.SetInsertPoint(&I);
                    Builder.CreateStore(
                        Builder.CreateAdd(
                            Builder.CreateLoad(i32Type, addInstCount),
                            llvm::ConstantInt::get(i32Type, 1)
                        ),
                        addInstCount
                    );
                }
            }
        }
    }
}
```