PROGRAMMING ASSIGNMENT 2

Rasesh Kumar Rout

rkr5351@psu.edu

CSE 521

Prof. Mahmut Kandemir

1. INTRODUCTION

The goal of this assignment is to implement a compiler pass using the LLVM Framework that can print various information about loops in a program.

Platform Used: Linux(Ubuntu

19.04) LLVM: Version Used 9

Followed the following link to install llvm in VMWare installed Ubuntu:

https://apt.llvm.org/

2. APPROACH

Created a test.c file to include my own functions having different

loop nesting structures. I used the following commands to execute

the passes.

1. make

2. clang-9 -S -emit-llvm test.c

3. llvm-as-9 test.ll

4. opt-9 test.ll --time-passes

5. opt-9 -load ./LoopInfoPass.so -function-info < test.bc

LoopInfoPass.cpp defines the compiler pass that I'm making.

Printing the Loop Information

For this I followed the steps on the provided starter code.

1. To find the function name, I used the getName() inside the runOnFunction method.
2. For finding depth of the loop.
   First we assign the outer loops in the function that we got via LoopWrapperInfo analysis with a depth of 0. Then we iterate over each children and assign them a depth of parent depth + 1. We do this in a recursive fashion.
3. To find if a loop contains subloops, we use the getSubLoops() to get the children loops and see size of the vector. If size is 0, we return false else true.
4. To find top-level basic blocks in a loop, we find the basic blocks number for that loop and subtract the basic block number of the immediate children loops.
5. To find number of instructions for a loop, we find all the basic blocks and iterate over all instructions in each basic block to get the count.
6. To find the number of atomic operations, we perform similar operation as for finding instruction but we only count the ones for which isAtomic() returns true.
7. To find the top-level branch instructions, we have to perform similar steps as for instruction, then increment the count only when the dynamic casting to BranchInst

is not null. We subtract all instructions for the immediate children loops to get only the top-level Branch instructions.

The test function I included in the code looks like:

```
int func1 (int a, int b, int c)
{
  int i;
  int ret = 0;
  for (i = a; i < b; i++) {
   c = c+1;
  }

  for (i = a; i<b; i++){
   c = c+2;
  }
  return ret + c;
}

int func2 (int a, int b, int c){
 int ret = 0;
 int i,j,k;
 for(i=0; i<10; i++){
   int d = 0;
   if(d<10){
     d++;
   }else{
     d--;
   }
   int atm = 0;
   __sync_fetch_and_add(&atm, 0x00000001);
   for(j=0; j<20; j++){
     for(k=0; k<10; k++){
      c = c+i+j+k;
     }
   }
 }
 return ret+c;
}

int func3(int a, int b){
  int ret = 0;
  int i,j,k;
  for(i=0; i<10; i++){
    for(j=0; j<10; j++){
      ret = ret + i + j;
    }
  }
  for(i=0; i<10; i++){
    for(j=0; j<10; j++){
      for(k=0; k<10; k++){
        ret = ret + i + j - k;
  int atm = 0;
  __sync_fetch_and_add(&atm, 0x00000001);
      }
    }
  }
  return ret;
}
```

OUTPUT:

```
CSE521 Loop Information Pass
<0>: func=func1, depth=0, subLoops=false, BBs=3, instrs=12, atomics=0, branches
=3
<1>: func=func1, depth=0, subLoops=false, BBs=3, instrs=12, atomics=0, branches
=3
0x244ee58
0x244edc0
<2>: func=func2, depth=0, subLoops=true, BBs=7, instrs=50, atomics=1, branches=
7
<3>: func=func2, depth=1, subLoops=true, BBs=4, instrs=26, atomics=0, branches=
4
<4>: func=func2, depth=2, subLoops=false, BBs=3, instrs=16, atomics=0, branches
=3
0x244eef0
0x244ee58
0x244edc0
<5>: func=func3, depth=0, subLoops=true, BBs=4, instrs=38, atomics=1, branches=
4
<6>: func=func3, depth=1, subLoops=true, BBs=4, instrs=28, atomics=1, branches=
4
<7>: func=func3, depth=2, subLoops=false, BBs=3, instrs=18, atomics=1, branches
=3
<8>: func=func3, depth=0, subLoops=true, BBs=4, instrs=24, atomics=0, branches=
4
<9>: func=func3, depth=1, subLoops=false, BBs=3, instrs=14, atomics=0, branches
=3
rash@ubuntu:~/cse521/PA2$
```