CS5544 Homework 3 Writeup Report

ZENG TAO

1.Immediate Dominators

Class DomTreeNode and DomTree has been used to store the dominance relationship of basic blocks inside a loop.

In the Class DomTreeNode, the field parent is the immediate dominator of current node, the field children are the successors of current node in the dominance tree.

In the Class DomTree, the field root denotes the root of the current dominance tree, the field nodes contains all the DomTreeNode in the dominance tree.

The data flow framework in the assignment2 has been reused here to construct an analysis pass to compute the dominance relationship of basic block inside a loop. For the dominators analysis pass, the following data-flow algorithm has been adopted:

	Dominators
Domain	The power set of N
Direction	Forwards
Transfer function	$f_B(x) = x \cup \{B\}$
Boundary	$\text{OUT}[\text{ENTRY}] = \{\text{ENTRY}\}$
Meet ($\wedge$)	$\cap$
Equations	$\text{OUT}[B] = f_B(\text{IN}[B])$ $\text{IN}[B] = \bigwedge_{P,pred(B)} \text{OUT}[P]$
Initialization	$\text{OUT}[B] = N$

After running the dominator pass, the OUT of each basic block is the set of basic blocks that dominate current block. According to the information in the OUT set, we can find the immediate dominator of current node through a reverse breadth first search.

The pseudo code of the algorithm is showing as flowing:

```
Initialize a worklist queue with current node.
Initialize a visited set.
While !worklist.empty() then
    node = worklist.pop_front();
    If visited[current] then
       Continue;
    If node != current && OUT[node] == TRUE then
       Return node;
    visited[node] = TRUE;
    For each predecessor of node:
       Worklist.push( predecessor)
    end
```

end

With the immediate dominators relationship, the dominator tree can be constructed in the following way: The root node is the node whose immediate dominance is NULL, iterate all the nodes and added all the nodes immediately dominated the root to its children. Take its children as the root node iteratively and repeat the process until all the nodes have been added to the tree.

At last, we created a DominatorPass derived from the LoopPass. The runOnLoop function has been overrided. In the runOnLoop function, the above data flow algorithm has been executed to generate the dominator tree. The information of immediate dominator of each basic in the loop is printed.

Immediate Dominator Test 1: basic_main.c

```c
int main(void){
  int x = 5;
  int r = 10;
  for (int i = 0; i < 42*x; i++) {
    int y = 5;
    r += y;
  }
  return 0;
};
```

Transformed Bitcode:

```
; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main() #0 {
  br label %1

1:                                                ; preds = %6, %0
  %.01 = phi i32 [ 10, %0 ], [ %5, %6 ]
  %.0 = phi i32 [ 0, %0 ], [ %7, %6 ]
  %2 = mul nsw i32 42, 5
  %3 = icmp slt i32 %.0, %2
  br i1 %3, label %4, label %8

4:                                                ; preds = %1
  %5 = add nsw i32 %.01, 5
  br label %6

6:                                                ; preds = %4
  %7 = add nsw i32 %.0, 1
  br label %1

8:                                                ; preds = %1
  ret i32 0
}
```

Immediate dominators:

```
user@user-VB:~/Documents/llvm_assignment3/Dominators/tests$ opt -enable-new-pm=0 -load ../dominator.so --dominators m2r_basic_main.bc -o out

Loop<depth = 1>
for.cond idom entry
for.body idom for.cond
for.inc idom for.body
```

Immediate Dominator Test 2: nested_basic_main.c

```c
int main(void){
  int x = 5;
  int y = 1;
  int r = 10;
  for (int i = 0; i < 2*x; i++) {
    for (int j = 0; j < 12345*y; j++) {
      r += y;
    }
  }
  return r;
};
```

Transformed Bitcode:

```
; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main() #0 {
  br label %1

1:                                                ; preds = %13, %0
  %.02 = phi i32 [ 10, %0 ], [ %.1, %13 ]
  %.01 = phi i32 [ 0, %0 ], [ %14, %13 ]
  %2 = mul nsw i32 2, 5
  %3 = icmp slt i32 %.01, %2
  br i1 %3, label %4, label %15

4:                                                ; preds = %1
  br label %5

5:                                                ; preds = %10, %4
  %.1 = phi i32 [ %.02, %4 ], [ %9, %10 ]
  %.0 = phi i32 [ 0, %4 ], [ %11, %10 ]
  %6 = mul nsw i32 12345, 1
  %7 = icmp slt i32 %.0, %6
  br i1 %7, label %8, label %12

8:                                                ; preds = %5
  %9 = add nsw i32 %.1, 1
  br label %10

10:                                               ; preds = %8
  %11 = add nsw i32 %.0, 1
  br label %5

12:                                               ; preds = %5
  br label %13

13:                                               ; preds = %12
  %14 = add nsw i32 %.01, 1
  br label %1

15:                                               ; preds = %1
  ret i32 %.02
}
```

Immediate Dominators:

```
user@user-VB:~/Documents/llvm_assignment3/Dominators/tests$ opt -enable-new-pm=0 -load ../dominator.so --dominators m2r_nested_main.bc -o out

Loop<depth = 2>
for.cond1 idom for.body
for.body4 idom for.cond1
for.inc idom for.body4

Loop<depth = 1>
for.cond idom entry
for.body idom for.cond
for.cond1 idom for.body
for.end idom for.cond1
for.inc5 idom for.end
for.body4 idom for.cond1
for.inc idom for.body4
```

Immediate Dominator Test 3: double_nested_basic_main.c

```c
int main(void){
  int x = 5;
  int y = 1;
  int r = 10;
  for (int i = 0; i < 2*x; i++) {
    for (int j = 0; j < 12345*y; j++) {
      for (int k = 0; k < 5; k++) {
        int a = 5;
        r += a + y;
      }
    }
  }
  return r;
}
```

Transformed Bitcode:

```
; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main() #0 {
  br label %1

1:                                                ; preds = %20, %0
  %.03 = phi i32 [ 10, %0 ], [ %.1, %20 ]
  %.02 = phi i32 [ 0, %0 ], [ %21, %20 ]
  %2 = mul nsw i32 2, 5
  %3 = icmp slt i32 %.02, %2
  br i1 %3, label %4, label %22

4:                                                ; preds = %1
  br label %5

5:                                                ; preds = %17, %4
  %.1 = phi i32 [ %.03, %4 ], [ %.2, %17 ]
  %.01 = phi i32 [ 0, %4 ], [ %18, %17 ]
  %6 = mul nsw i32 12345, 1
  %7 = icmp slt i32 %.01, %6
  br i1 %7, label %8, label %19

8:                                                ; preds = %5
  br label %9

9:                                                ; preds = %14, %8
  %.2 = phi i32 [ %.1, %8 ], [ %13, %14 ]
  %.0 = phi i32 [ 0, %8 ], [ %15, %14 ]
  %10 = icmp slt i32 %.0, 5
  br i1 %10, label %11, label %16

11:                                               ; preds = %9
  %12 = add nsw i32 5, 1
  %13 = add nsw i32 %.2, %12
  br label %14

14:                                               ; preds = %11
  %15 = add nsw i32 %.0, 1
  br label %9

16:                                               ; preds = %9
  br label %17

17:                                               ; preds = %16
  %18 = add nsw i32 %.01, 1
  br label %5
```

Immediate Dominators:

```
user@user-VB:~/Documents/llvm_assignment3/Dominators/tests$ opt -enable-new-pm=0 -load ../dominator.so --dominators m2r_double_nested_main.bc -o out

Loop<depth = 3>
for.cond5 idom for.body4
for.body7 idom for.cond5
for.inc idom for.body7

Loop<depth = 2>
for.cond1 idom for.body
for.body4 idom for.cond1
for.cond5 idom for.body4
for.end idom for.cond5
for.inc9 idom for.end
for.body7 idom for.cond5
for.inc idom for.body7

Loop<depth = 1>
for.cond idom entry
for.body idom for.cond
for.cond1 idom for.body
for.end11 idom for.cond1
for.inc12 idom for.end11
for.body4 idom for.cond1
for.cond5 idom for.body4
for.end idom for.cond5
for.inc9 idom for.end
for.body7 idom for.cond5
for.inc idom for.body7
```

2. Dead Code Elimination

To identify the faint variables, we first identify the set of strongly live variables.

A variable is strongly live if
⮝ it is used in a statement other than assignment statement, or (same as simple liveness)
⮝ it is used in an assignment statement defining a variable that is strongly live (different from simple liveness)

The set of faint variables will be the complementary set of strongly live variables. If a variable is not included in the strongly live variable set, then it is a faint variable.

The KillSet and GenSet of Strongly live variable analysis are defined as:
•Killing: An assignment statement, an input statement, or BI (this is same as killing in simple liveness)
•Generation: A direct use or a use for defining values that are strongly live (this is different from generation in simple liveness)

The data flow equation of SLVA is defined as:

$$IN_n = f_n(OUT_n)$$

$$OUT_n = \begin{cases} BI & n \text{ is } End \\ \bigcup_{s \in succ(n)} IN_s & \text{otherwise} \end{cases}$$

BI denotes the boundary information of the block n.

The Transfer function of SLVA is defined as:

$$f_n(X) = \begin{cases} (X - \{y\}) \cup (Opd(e) \cap \mathbb{V}\text{ar}) & n \text{ is } y = e, e \in \mathbb{E}\text{xpr},\ y \in X \\ X - \{y\} & n \text{ is } input(y) \\ X \cup \{y\} & n \text{ is } use(y) \\ X & \text{otherwise} \end{cases}$$

The SLVA analysis is a backward analysis, the meet operation is union. The initial condition for all blocks is: In[block] = ∅. The boundary condition for exit is: In[Exit] = ∅.

After the SLVA analysis pass has been executed, we iterate the basic blocks in a post-order. In each of the basic block, we iterate all the instructions backward, if the expression is not included in the IN set of the current block, it is identified as the dead code and removed later.

DCE Benchmarks:

The source code and the descriptions for the DCE benchmarks will be listed in the appendix. The summary results of DCE are shown as following:

Benchmark	Dynamic instructions before DCE	Dynamic instructions after DCE
dce_bench1.c	7	5
dce_bench2.c	1110	908
dce_bench3.c	7009	6008
dce_bench4.c	20	16

3. Loop Invariant Code Motion

The Loop Invariant Code Motion pass has been implemented as a transformation pass derived from the LoopPass in LLVM. The runOnLoop function has been overridden to check all the invariant variables in the loop. In the LICM pass, we also added the two required passes: LoopInfoPass and DominatorPass. The pseudo code of the algorithm is shown as following:

A breakdown of invariant check is shown as following:

- **isSafeToSpeculativelyExecute()** – Determine if executing the given instruction is safe even though it may nor be used.
- **!inst->mayReadFromMemory()** -In the multithreading environment or the interrupts, the variable may change during the execution.
- **!isa<LandingPadInst>(inst)** -If it is a target of a jump or a new conditional branch.
- The operands of this instruction are invariant, which means that they were defined outside the loop, or they are already identified as invariant, or they are not instructions.

There are some helper functions of hoisting the invariant instructions.

BasicBlock *getOrCreateHoistedBlock(BasicBlock *BB);

This function will check if the current block is conditional based on a pending branch. If it is not involved in a pending branch, then we simply hoist the instruction to the loop preheader. Otherwise, we create new hoisted version of the branch blocks, which are not contained in the HoistDestinationMap, and link those new blocks with the branches. Finally, we place the branch instruction at the end of the target hoist block and return the block.

void registerPossiblyHoistableBranch(BranchInst *BI);
This function will keep track of the branch instructions, which are possibly valid for hoisting.

bool hoistRegion(LoopInfo *LI, DominatorTree *DT,Loop *CurLoop);
The function will iterate through the blocks in the loop in reverse post-order, and check each instruction in the block to see if it is invariant. If it is, then the function getOrCreateHoistedBlock will be called to create a holding block to the invariant instruction. If we hoisted instructions to a conditional block that may not dominate their uses that weren't hoisted (such as phis where some operands are not loop invariant), then we make them unconditional by moving them to their immediate dominator. We iterate through the instructions in reverse order which ensures that when we rehoist an instruction we rehoist its operands, and also keep track of where in the block we are rehoisting to, to make sure that we rehoist instructions before the instructions that use them.

static void hoist(Instruction &I, const DominatorTree *DT, const Loop *CurLoop, BasicBlock *Dest);
When an instruction is found to only use loop invariant operands inside of the loop that is safe to hoist, this function is called to do the dirty work.

LICM Benchmarks:
The source code and the descriptions for the LICM benchmarks will be listed in the appendix. The summary results of LICM are shown as following:

Benchmark	Dynamic instructions before LICM	Dynamic instructions after LICM
licm_bench1.c	4294	4024
licm_bench2.c	29882	23484
licm_bench3.c	623	513

Appendix:

DCE Benchmarks

DCE Benchmark 1:

dce_bench1.c

```c
int deadvar(int a, int b) {
        int c = 2 + b;
        int z = 2 * c;
  int x = z - 2;
        return c + a;
}

int main(int argc, char const *argv[]) {
  int a = deadvar(3, 4);
  return a;
}
```

Benchmark1 bitcode before DCE:

```
; ModuleID = 'm2r_dce_bench1.bc'
source_filename = "dce_bench1.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @deadvar(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  %mul = mul nsw i32 2, %add
  %sub = sub nsw i32 %mul, 2
  %add1 = add nsw i32 %add, %a
  ret i32 %add1
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @deadvar(i32 noundef 3, i32 noundef 4)
  ret i32 %call
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math
"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+
x87" "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"uwtable", i32 1}
!2 = !{i32 7, !"frame-pointer", i32 2}
!3 = !{!"clang version 14.0.0 (https://github.com/llvm/llvm-project.git ce5b04cc048aa9b82355bbea21a062462553eb47
)"}
```

Benmark1 dynamic instructions before DCE:

```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter m2r_dce_bench1.bc
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

44 bitcode-reader - Number of Metadata records loaded
 4 bitcode-reader - Number of MDStrings loaded
 7 interpreter    - Number of dynamic instructions executed
```

Benchmark1 bitcode after DCE:

```
; ModuleID = 'out_dce_bench1'
source_filename = "dce_bench1.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @deadvar(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  %add1 = add nsw i32 %add, %a
  ret i32 %add1
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @deadvar(i32 noundef 3, i32 noundef 4)
  ret i32 %call
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math
"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+
x87" "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"uwtable", i32 1}
!2 = !{i32 7, !"frame-pointer", i32 2}
!3 = !{!"clang version 14.0.0 (https://github.com/llvm/llvm-project.git ce5b04cc048aa9b82355bbea21a062462553eb47
)"}
```

Benchmark1 dynamic instructions after DCE:

```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter out_dce_bench1
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

44 bitcode-reader - Number of Metadata records loaded
 4 bitcode-reader - Number of MDStrings loaded
 5 interpreter    - Number of dynamic instructions executed
```

DCE Benchmark 2:

dce_bench2.c

```c
int dcebench2(int a, int b) {
        int c = 2 + b;
        int z = 2 * c;
        int x = z - 2;

        for(int i = 0; i < 100; i++) {
                x = c + i;
                if (b < 0)
                        c = c + 1;
                else
                        c = c + a;

                z = x + c;
        }

        return c + a;
}

int main(int argc, char const *argv[]) {
        dcebench2(2, 3);
        return 0;
}
```

Benchmark2 bitcode before DCE:

```
; ModuleID = 'm2r_dce_bench2.bc'
source_filename = "dce_bench2.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @dcebench2(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  %mul = mul nsw i32 2, %add
  %sub = sub nsw i32 %mul, 2
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %c.0 = phi i32 [ %add, %entry ], [ %c.1, %for.inc ]
  %i.0 = phi i32 [ 0, %entry ], [ %inc, %for.inc ]
  %cmp = icmp slt i32 %i.0, 100
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %add1 = add nsw i32 %c.0, %i.0
  %cmp2 = icmp slt i32 %b, 0
  br i1 %cmp2, label %if.then, label %if.else

if.then:                                          ; preds = %for.body
  %add3 = add nsw i32 %c.0, 1
  br label %if.end

if.else:                                          ; preds = %for.body
  %add4 = add nsw i32 %c.0, %a
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %c.1 = phi i32 [ %add3, %if.then ], [ %add4, %if.else ]
  %add5 = add nsw i32 %add1, %c.1
  br label %for.inc

for.inc:                                          ; preds = %if.end
  %inc = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !4

for.end:                                          ; preds = %for.cond
  %add6 = add nsw i32 %c.0, %a
  ret i32 %add6
                                                                              11,1          Top
```

```
; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @dcebench2(i32 noundef 2, i32 noundef 3)
  ret i32 0
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"="
true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "
tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"uwtable", i32 1}
!2 = !{i32 7, !"frame-pointer", i32 2}
!3 = !{!"clang version 14.0.0 (https://github.com/llvm/llvm-project.git ce5b04cc048aa9b82355bbea21a062462553eb47)"}
!4 = distinct !{!4, !5}
!5 = !{!"llvm.loop.mustprogress"}
                                                                                64,1          Bot
```

Dynamic instructions of benchmark2 before DCE:

```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter m2r_dce_bench2.bc
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

  49 bitcode-reader - Number of Metadata records loaded
   5 bitcode-reader - Number of MDStrings loaded
1110 interpreter    - Number of dynamic instructions executed
```

Benchmark2 bitcode after DCE:

```
; ModuleID = 'out_dce_bench2'
source_filename = "dce_bench2.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @dcebench2(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %c.0 = phi i32 [ %add, %entry ], [ %c.1, %for.inc ]
  %i.0 = phi i32 [ 0, %entry ], [ %inc, %for.inc ]
  %cmp = icmp slt i32 %i.0, 100
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %cmp2 = icmp slt i32 %b, 0
  br i1 %cmp2, label %if.then, label %if.else

if.then:                                          ; preds = %for.body
  %add3 = add nsw i32 %c.0, 1
  br label %if.end

if.else:                                          ; preds = %for.body
  %add4 = add nsw i32 %c.0, %a
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %c.1 = phi i32 [ %add3, %if.then ], [ %add4, %if.else ]
  br label %for.inc

for.inc:                                          ; preds = %if.end
  %inc = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !4

for.end:                                          ; preds = %for.cond
  %add6 = add nsw i32 %c.0, %a
  ret i32 %add6
}
```

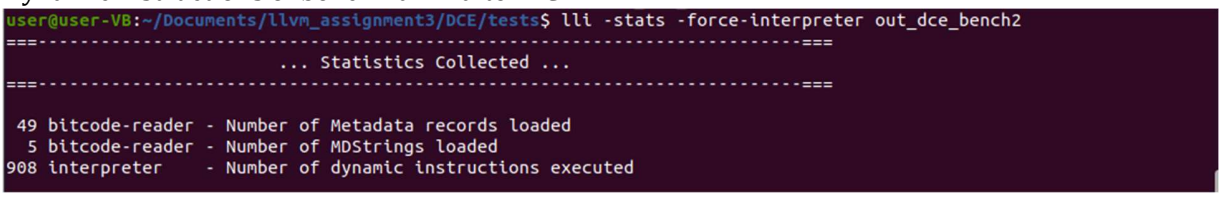
```
; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @dcebench2(i32 noundef 2, i32 noundef 3)
  ret i32 0
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"="
true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "
tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"uwtable", i32 1}
!2 = !{i32 7, !"frame-pointer", i32 2}
!3 = !{!"clang version 14.0.0 (https://github.com/llvm/llvm-project.git ce5b04cc048aa9b82355bbea21a062462553eb47)"}
!4 = distinct !{!4, !5}
!5 = !{!"llvm.loop.mustprogress"}
                                                                                60,1          Bot
```

Dynamic instructions of benchmark2 after DCE:

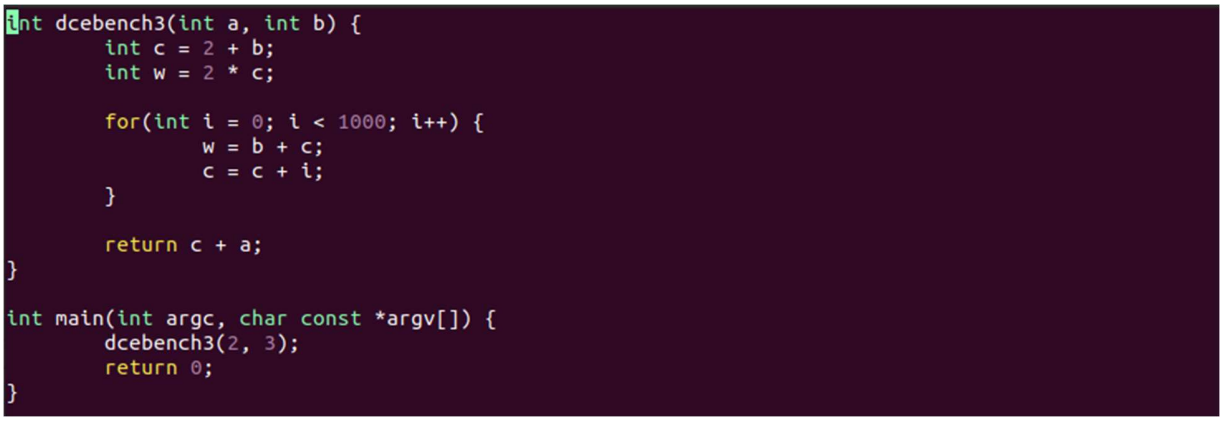


```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter out_dce_bench2
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

 49 bitcode-reader - Number of Metadata records loaded
  5 bitcode-reader - Number of MDStrings loaded
908 interpreter    - Number of dynamic instructions executed
```

DCE Benchmark 3:

dce_bench3.c



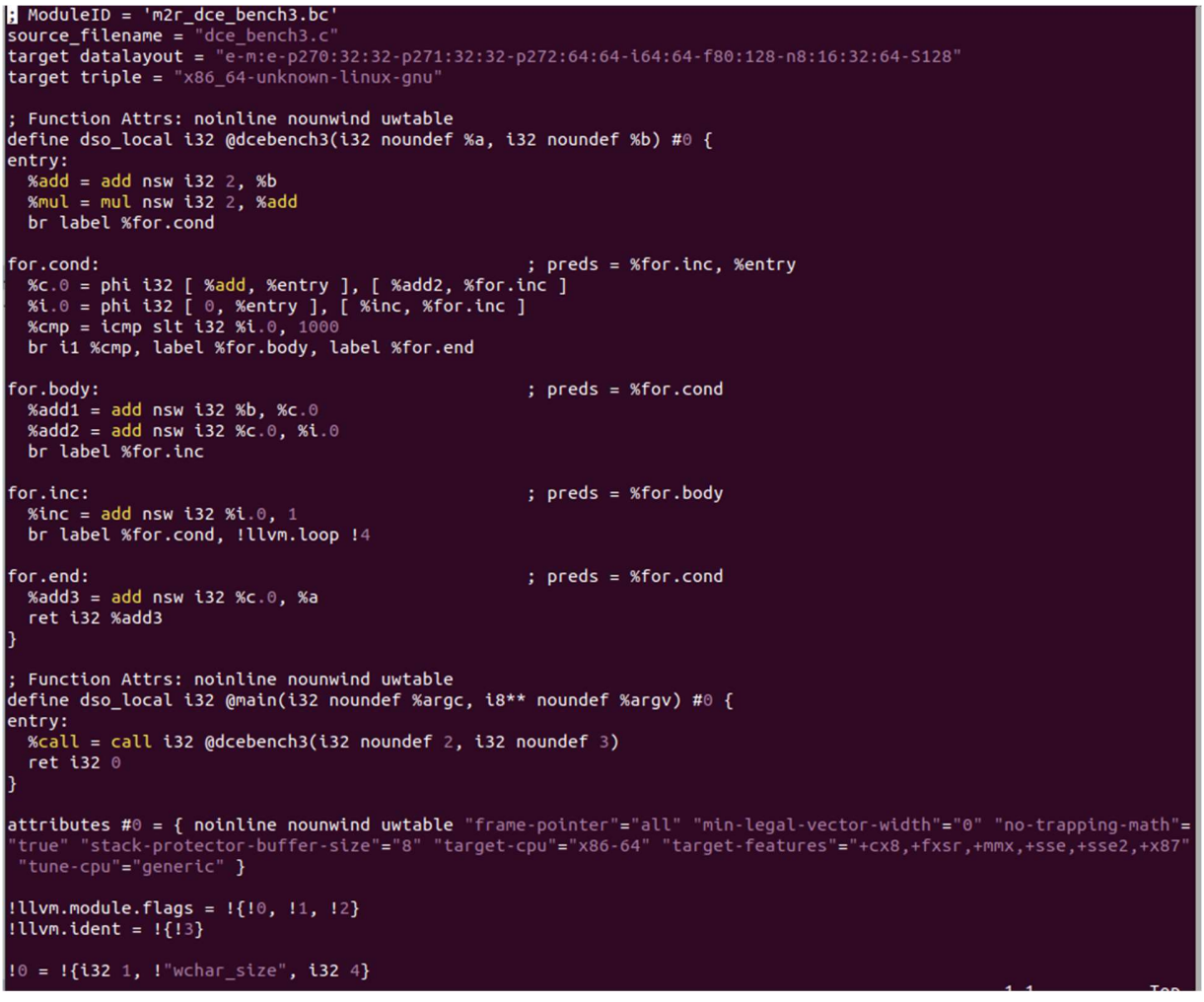
```
int dcebench3(int a, int b) {
        int c = 2 + b;
        int w = 2 * c;

        for(int i = 0; i < 1000; i++) {
                w = b + c;
                c = c + i;
        }

        return c + a;
}

int main(int argc, char const *argv[]) {
        dcebench3(2, 3);
        return 0;
}
```

Benchmark3 bitcode before DCE:



```
; ModuleID = 'm2r_dce_bench3.bc'
source_filename = "dce_bench3.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @dcebench3(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  %mul = mul nsw i32 2, %add
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %c.0 = phi i32 [ %add, %entry ], [ %add2, %for.inc ]
  %i.0 = phi i32 [ 0, %entry ], [ %inc, %for.inc ]
  %cmp = icmp slt i32 %i.0, 1000
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %add1 = add nsw i32 %b, %c.0
  %add2 = add nsw i32 %c.0, %i.0
  br label %for.inc

for.inc:                                          ; preds = %for.body
  %inc = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !4

for.end:                                          ; preds = %for.cond
  %add3 = add nsw i32 %c.0, %a
  ret i32 %add3
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @dcebench3(i32 noundef 2, i32 noundef 3)
  ret i32 0
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"=
"true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87"
 "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
```

Benchmark3 dynamic instructions before DCE:

```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter m2r_dce_bench3.ll
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

7009 interpreter - Number of dynamic instructions executed
```

Benchmark3 bitcode after DCE:

```
; ModuleID = 'out_dce_bench3'
source_filename = "dce_bench3.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @dcebench3(i32 noundef %a, i32 noundef %b) #0 {
entry:
  %add = add nsw i32 2, %b
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %c.0 = phi i32 [ %add, %entry ], [ %add2, %for.inc ]
  %i.0 = phi i32 [ 0, %entry ], [ %inc, %for.inc ]
  %cmp = icmp slt i32 %i.0, 1000
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %add2 = add nsw i32 %c.0, %i.0
  br label %for.inc

for.inc:                                          ; preds = %for.body
  %inc = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !4

for.end:                                          ; preds = %for.cond
  %add3 = add nsw i32 %c.0, %a
  ret i32 %add3
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @dcebench3(i32 noundef 2, i32 noundef 3)
  ret i32 0
}

attributes #0 = { noinline nounwind uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"=
"true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87"
 "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}
```

Benchmark3 dynamic instructions after DCE:

```
user@user-VB:~/Documents/llvm_assignment3/DCE/tests$ lli -stats -force-interpreter out_dce_bench3
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

  49 bitcode-reader - Number of Metadata records loaded
   5 bitcode-reader - Number of MDStrings loaded
6008 interpreter    - Number of dynamic instructions executed
```

LICM Benchmarks

LICM Bechmark1:

Licm_bench1.c

```c
int loop (int a, int b, int c)
{
  int i;
  int ret = 0;
  for (i = a; i < b; i++) {
    int x = c + 4;   // should bubble above i loop
    int j, k;
    for (j = i; j < b; j++) {
      int y = x + i; // should bubble above j loop
      int z = x + a; // should bubble above i loop
      ret += (i * j) + y;
    }
    for (k = i; k < c; k++) {
      ret += 1;
    }
  }
  return ret;
}

int main(int argc, char const *argv[]) {
  int a = loop(4, 20, 40);
  return a;
}
```

Benchmark1 bitcode before LICM:

```
; ModuleID = 'm2r_licm_bench1.bc'
source_filename = "licm_bench1.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  br label %for.cond

for.cond:                                         ; preds = %for.inc15, %entry
  %i.0 = phi i32 [ %a, %entry ], [ %inc16, %for.inc15 ]
  %ret.0 = phi i32 [ 0, %entry ], [ %ret.2, %for.inc15 ]
  %cmp = icmp slt i32 %i.0, %b
  br i1 %cmp, label %for.body, label %for.end17

for.body:                                         ; preds = %for.cond
  %add = add nsw i32 %c, 4
  br label %for.cond1

for.cond1:                                        ; preds = %for.inc, %for.body
  %ret.1 = phi i32 [ %ret.0, %for.body ], [ %add7, %for.inc ]
  %j.0 = phi i32 [ %i.0, %for.body ], [ %inc, %for.inc ]
  %cmp2 = icmp slt i32 %j.0, %b
  br i1 %cmp2, label %for.body3, label %for.end

for.body3:                                        ; preds = %for.cond1
  %add4 = add nsw i32 %add, %i.0
  %add5 = add nsw i32 %add, %a
  %mul = mul nsw i32 %i.0, %j.0
  %add6 = add nsw i32 %mul, %add4
  %add7 = add nsw i32 %ret.1, %add6
  br label %for.inc

for.inc:                                          ; preds = %for.body3
  %inc = add nsw i32 %j.0, 1
  br label %for.cond1, !llvm.loop !4

for.end:                                          ; preds = %for.cond1
  br label %for.cond8

for.cond8:                                        ; preds = %for.inc12, %for.end
  %ret.2 = phi i32 [ %ret.1, %for.end ], [ %add11, %for.inc12 ]
  %k.0 = phi i32 [ %i.0, %for.end ], [ %inc13, %for.inc12 ]
  %cmp9 = icmp slt i32 %k.0, %c
  br i1 %cmp9, label %for.body10, label %for.end14
```

```
for.body10:                                       ; preds = %for.cond8
  %add11 = add nsw i32 %ret.2, 1
  br label %for.inc12

for.inc12:                                        ; preds = %for.body10
  %inc13 = add nsw i32 %k.0, 1
  br label %for.cond8, !llvm.loop !6

for.end14:                                        ; preds = %for.cond8
  br label %for.inc15

for.inc15:                                        ; preds = %for.end14
  %inc16 = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !7

for.end17:                                        ; preds = %for.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark1 dynamic instructions before LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter out_licm_bench1
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

  53 bitcode-reader - Number of Metadata records loaded
   5 bitcode-reader - Number of MDStrings loaded
4024 interpreter    - Number of dynamic instructions executed
```

Benchmark1 bitcode after LICM:

```
; ModuleID = 'out_licm_bench1'
source_filename = "licm_bench1.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  %add = add nsw i32 %c, 4
  %add5 = add nsw i32 %add, %a
  br label %for.cond

for.cond:                                         ; preds = %for.inc15, %entry
  %i.0 = phi i32 [ %a, %entry ], [ %inc16, %for.inc15 ]
  %ret.0 = phi i32 [ 0, %entry ], [ %ret.2, %for.inc15 ]
  %cmp = icmp slt i32 %i.0, %b
  br i1 %cmp, label %for.body, label %for.end17

for.body:                                         ; preds = %for.cond
  %add4 = add nsw i32 %add, %i.0
  br label %for.cond1

for.cond1:                                        ; preds = %for.inc, %for.body
  %ret.1 = phi i32 [ %ret.0, %for.body ], [ %add7, %for.inc ]
  %j.0 = phi i32 [ %i.0, %for.body ], [ %inc, %for.inc ]
  %cmp2 = icmp slt i32 %j.0, %b
  br i1 %cmp2, label %for.body3, label %for.end

for.body3:                                        ; preds = %for.cond1
  %mul = mul nsw i32 %i.0, %j.0
  %add6 = add nsw i32 %mul, %add4
  %add7 = add nsw i32 %ret.1, %add6
  br label %for.inc

for.inc:                                          ; preds = %for.body3
  %inc = add nsw i32 %j.0, 1
  br label %for.cond1, !llvm.loop !4

for.end:                                          ; preds = %for.cond1
  br label %for.cond8

for.cond8:                                        ; preds = %for.inc12, %for.end
  %ret.2 = phi i32 [ %ret.1, %for.end ], [ %add11, %for.inc12 ]
  %k.0 = phi i32 [ %i.0, %for.end ], [ %inc13, %for.inc12 ]
  %cmp9 = icmp slt i32 %k.0, %c
  br i1 %cmp9, label %for.body10, label %for.end14
```

```
for.body10:                                       ; preds = %for.cond8
  %add11 = add nsw i32 %ret.2, 1
  br label %for.inc12

for.inc12:                                        ; preds = %for.body10
  %inc13 = add nsw i32 %k.0, 1
  br label %for.cond8, !llvm.loop !6

for.end14:                                        ; preds = %for.cond8
  br label %for.inc15

for.inc15:                                        ; preds = %for.end14
  %inc16 = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !7

for.end17:                                        ; preds = %for.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark1 dynamic instructions after LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter m2r_licm_bench1.bc
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

  53 bitcode-reader - Number of Metadata records loaded
   5 bitcode-reader - Number of MDStrings loaded
4294 interpreter    - Number of dynamic instructions executed
```

LICM Bechmark2:

Licm_bench2.c

```c
int loop (int a, int b, int c)
{
  int i;
  int ret = 0;
  for (i = 0; i < a; i++) {
    int x = c + 4;   // should bubble above i loop
    int j, k;
    for (j = 0; j < b; j++) {
      int y = x + i; // should bubble above j loop
      int z = x + a; // should bubble above i loop
      ret += (i * j) + y;
      for (k = 0; k < c; k++) {
        int w = (i + j) * c; // should bubble above k
        int r = z + k;
        ret *= r;
      }
    }
  }
  return ret;
}

int main(int argc, char const *argv[]) {
  int a = loop(4, 20, 40);
  return a;
}
~
```

Benchmark2 bitcode before LICM:

```
; ModuleID = 'm2r_licm_bench2.bc'
source_filename = "licm_bench2.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  br label %for.cond

for.cond:                                         ; preds = %for.inc18, %entry
  %i.0 = phi i32 [ 0, %entry ], [ %inc19, %for.inc18 ]
  %ret.0 = phi i32 [ 0, %entry ], [ %ret.1, %for.inc18 ]
  %cmp = icmp slt i32 %i.0, %a
  br i1 %cmp, label %for.body, label %for.end20

for.body:                                         ; preds = %for.cond
  %add = add nsw i32 %c, 4
  br label %for.cond1

for.cond1:                                        ; preds = %for.inc15, %for.body
  %ret.1 = phi i32 [ %ret.0, %for.body ], [ %ret.2, %for.inc15 ]
  %j.0 = phi i32 [ 0, %for.body ], [ %inc16, %for.inc15 ]
  %cmp2 = icmp slt i32 %j.0, %b
  br i1 %cmp2, label %for.body3, label %for.end17

for.body3:                                        ; preds = %for.cond1
  %add4 = add nsw i32 %add, %i.0
  %add5 = add nsw i32 %add, %a
  %mul = mul nsw i32 %i.0, %j.0
  %add6 = add nsw i32 %mul, %add4
  %add7 = add nsw i32 %ret.1, %add6
  br label %for.cond8

for.cond8:                                        ; preds = %for.inc, %for.body3
  %ret.2 = phi i32 [ %add7, %for.body3 ], [ %mul14, %for.inc ]
  %k.0 = phi i32 [ 0, %for.body3 ], [ %inc, %for.inc ]
  %cmp9 = icmp slt i32 %k.0, %c
  br i1 %cmp9, label %for.body10, label %for.end

for.body10:                                       ; preds = %for.cond8
  %add11 = add nsw i32 %i.0, %j.0
  %mul12 = mul nsw i32 %add11, %c
  %add13 = add nsw i32 %add5, %k.0
  %mul14 = mul nsw i32 %ret.2, %add13
  br label %for.inc

for.inc:                                          ; preds = %for.body10
  %inc = add nsw i32 %k.0, 1
  br label %for.cond8, !llvm.loop !4

for.end:                                          ; preds = %for.cond8
  br label %for.inc15

for.inc15:                                        ; preds = %for.end
  %inc16 = add nsw i32 %j.0, 1
  br label %for.cond1, !llvm.loop !6

for.end17:                                        ; preds = %for.cond1
  br label %for.inc18

for.inc18:                                        ; preds = %for.end17
  %inc19 = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !7

for.end20:                                        ; preds = %for.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark2 dynamic instructions before LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter m2r_licm_bench2.bc
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

   53 bitcode-reader - Number of Metadata records loaded
    5 bitcode-reader - Number of MDStrings loaded
29882 interpreter    - Number of dynamic instructions executed
```

Benchmark2 bitcode after LICM:

```
; ModuleID = 'out_licm_bench2'
source_filename = "licm_bench2.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  %add = add nsw i32 %c, 4
  %add5 = add nsw i32 %add, %a
  br label %for.cond

for.cond:                                         ; preds = %for.inc18, %entry
  %i.0 = phi i32 [ 0, %entry ], [ %inc19, %for.inc18 ]
  %ret.0 = phi i32 [ 0, %entry ], [ %ret.1, %for.inc18 ]
  %cmp = icmp slt i32 %i.0, %a
  br i1 %cmp, label %for.body, label %for.end20

for.body:                                         ; preds = %for.cond
  %add4 = add nsw i32 %add, %i.0
  br label %for.cond1

for.cond1:                                        ; preds = %for.inc15, %for.body
  %ret.1 = phi i32 [ %ret.0, %for.body ], [ %ret.2, %for.inc15 ]
  %j.0 = phi i32 [ 0, %for.body ], [ %inc16, %for.inc15 ]
  %cmp2 = icmp slt i32 %j.0, %b
  br i1 %cmp2, label %for.body3, label %for.end17

for.body3:                                        ; preds = %for.cond1
  %mul = mul nsw i32 %i.0, %j.0
  %add6 = add nsw i32 %mul, %add4
  %add7 = add nsw i32 %ret.1, %add6
  %add11 = add nsw i32 %i.0, %j.0
  %mul12 = mul nsw i32 %add11, %c
  br label %for.cond8

for.cond8:                                        ; preds = %for.inc, %for.body3
  %ret.2 = phi i32 [ %add7, %for.body3 ], [ %mul14, %for.inc ]
  %k.0 = phi i32 [ 0, %for.body3 ], [ %inc, %for.inc ]
  %cmp9 = icmp slt i32 %k.0, %c
  br i1 %cmp9, label %for.body10, label %for.end

for.body10:                                       ; preds = %for.cond8
  %add13 = add nsw i32 %add5, %k.0
  %mul14 = mul nsw i32 %ret.2, %add13
  br label %for.inc
```

```
for.inc:                                          ; preds = %for.body10
  %inc = add nsw i32 %k.0, 1
  br label %for.cond8, !llvm.loop !4

for.end:                                          ; preds = %for.cond8
  br label %for.inc15

for.inc15:                                        ; preds = %for.end
  %inc16 = add nsw i32 %j.0, 1
  br label %for.cond1, !llvm.loop !6

for.end17:                                        ; preds = %for.cond1
  br label %for.inc18

for.inc18:                                        ; preds = %for.end17
  %inc19 = add nsw i32 %i.0, 1
  br label %for.cond, !llvm.loop !7

for.end20:                                        ; preds = %for.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark2 dynamic instructions after LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter out_licm_bench2
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

   53 bitcode-reader - Number of Metadata records loaded
    5 bitcode-reader - Number of MDStrings loaded
23484 interpreter    - Number of dynamic instructions executed
```

LICM Benchmark3:

Licm_bench3.c

```c
int loop (int a, int b, int c)
{
  int ret = 0;
  while (a < (b + c)) {
    int x;
    if (a % 2) {
      x = 5 + b; // should get hoisted
    } else {
      x = 4 + c; // should get hoisted
    }
    ret += x;
    a++;
  }
  return ret;
}

int main(int argc, char const *argv[]) {
  int a = loop(4, 20, 40);
  return a;
}
```

Benchmark3 bitcode before LICM:

```
; ModuleID = 'out_licm_bench3'
source_filename = "licm_bench3.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  %add = add nsw i32 %b, %c
  %add1 = add nsw i32 5, %b
  %add2 = add nsw i32 4, %c
  br label %while.cond

while.cond:                                       ; preds = %if.end, %entry
  %ret.0 = phi i32 [ 0, %entry ], [ %add3, %if.end ]
  %a.addr.0 = phi i32 [ %a, %entry ], [ %inc, %if.end ]
  %cmp = icmp slt i32 %a.addr.0, %add
  br i1 %cmp, label %while.body, label %while.end

while.body:                                       ; preds = %while.cond
  %rem = srem i32 %a.addr.0, 2
  %tobool = icmp ne i32 %rem, 0
  br i1 %tobool, label %if.then, label %if.else

if.then:                                          ; preds = %while.body
  br label %if.end

if.else:                                          ; preds = %while.body
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %x.0 = phi i32 [ %add1, %if.then ], [ %add2, %if.else ]
  %add3 = add nsw i32 %ret.0, %x.0
  %inc = add nsw i32 %a.addr.0, 1
  br label %while.cond, !llvm.loop !4

while.end:                                        ; preds = %while.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark3 dynamic instructions before LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter m2r_licm_bench3.bc
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

 49 bitcode-reader - Number of Metadata records loaded
  5 bitcode-reader - Number of MDStrings loaded
623 interpreter    - Number of dynamic instructions executed
```

Benchmark3 bitcode after LICM:

```
; ModuleID = 'm2r_licm_bench3.bc'
source_filename = "licm_bench3.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @loop(i32 noundef %a, i32 noundef %b, i32 noundef %c) #0 {
entry:
  br label %while.cond

while.cond:                                       ; preds = %if.end, %entry
  %ret.0 = phi i32 [ 0, %entry ], [ %add3, %if.end ]
  %a.addr.0 = phi i32 [ %a, %entry ], [ %inc, %if.end ]
  %add = add nsw i32 %b, %c
  %cmp = icmp slt i32 %a.addr.0, %add
  br i1 %cmp, label %while.body, label %while.end

while.body:                                       ; preds = %while.cond
  %rem = srem i32 %a.addr.0, 2
  %tobool = icmp ne i32 %rem, 0
  br i1 %tobool, label %if.then, label %if.else

if.then:                                          ; preds = %while.body
  %add1 = add nsw i32 5, %b
  br label %if.end

if.else:                                          ; preds = %while.body
  %add2 = add nsw i32 4, %c
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %x.0 = phi i32 [ %add1, %if.then ], [ %add2, %if.else ]
  %add3 = add nsw i32 %ret.0, %x.0
  %inc = add nsw i32 %a.addr.0, 1
  br label %while.cond, !llvm.loop !4

while.end:                                        ; preds = %while.cond
  ret i32 %ret.0
}

; Function Attrs: noinline nounwind uwtable
define dso_local i32 @main(i32 noundef %argc, i8** noundef %argv) #0 {
entry:
  %call = call i32 @loop(i32 noundef 4, i32 noundef 20, i32 noundef 40)
  ret i32 %call
}
```

Benchmark3 dynamic instructions after LICM:

```
user@user-VB:~/Documents/llvm_assignment3/LICM/tests$ lli -stats -force-interpreter out_licm_bench3
===-------------------------------------------------------------------------===
                          ... Statistics Collected ...
===-------------------------------------------------------------------------===

 49 bitcode-reader - Number of Metadata records loaded
  5 bitcode-reader - Number of MDStrings loaded
513 interpreter    - Number of dynamic instructions executed
```