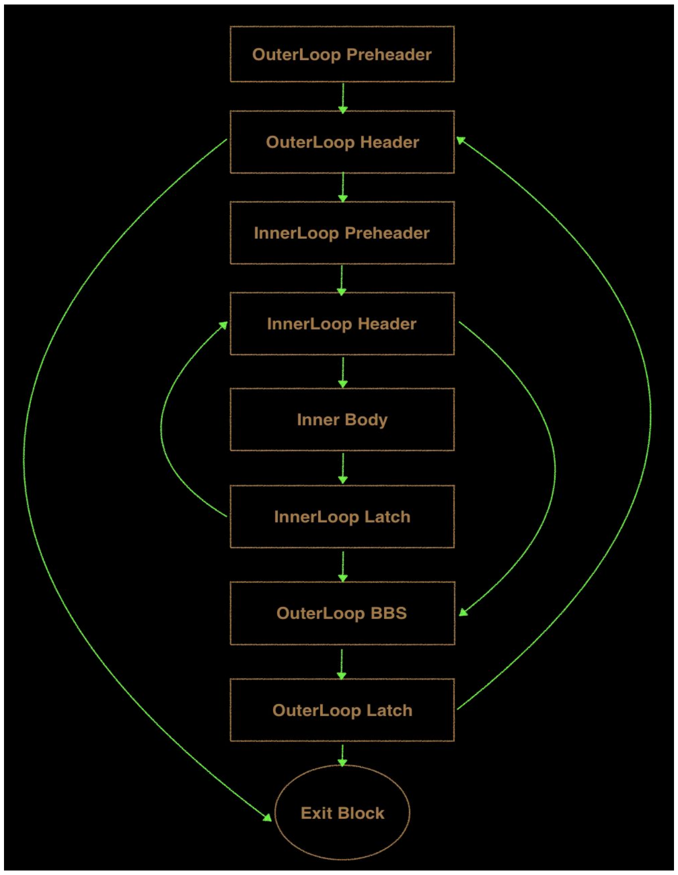
CSE 504 Homework : #3

Prateek Roy SBUID : 111481907 NetID: praroy Email: praroy@cs.stonybrook.edu

1) **Perfectly Nested Loop** : Below is the control flow diagram for a nested loop



For perfectly/tightly nested loop, there should not any basic block between Outer Loop Header and Inner Loop Preheader. Similarly, there should not be any basic block between Outer Loop bbs and Outer Loop latch. This definition is found on the llvm git page.
Also I have added another condition that there should not any statement in between the Outer Loop and Inner Loop to be perfectly nested.

Implementation:

perfectlyNested(): It uses Outer Loop Header, Inner Loop PreHeader Basic Block* to validate that the only successor of Outer Loop Header is the Inner Loop PreHeader and there are no basic blocks present in between them.

2) **Dependence between loop index variables** : To find out the dependence between index variables of Outer Loop and Inner Loop, we need to first find out index variable of the Outer Loop. The index variable is always present in the previous instruction of the terminator instruction of the outer preheader.
After finding the index variable of the outer loop, we search for the use of index variable in the inner preheader block, inner header block and outer loop latch block. If there is no use of outer index variable in those blocks, then we can say that the loops are not dependent. If we find the use of outer index variable in above three blocks, then we conclude that the loop index variables are dependent.

Implementation:

areLoopDependent(): First, we find Outer Loop Index Variable using the Basic Block* Outer Loop Header. Then we use the api *sUsedInBasicBlock(Basic Block*)* to find whether the index variable is used in inner preheader block, inner header block and outer loop latch block using their BasicBlock*. If we find any use, then there the loops are dependent.

Some sample examples and output are attached in Part 1,2 folder.

3) **Loop Interchange :** To interchange the loops there are three steps:

- Extract the index variables pointers and values of the inner and outer loops which are present inside the Inner Loop Preheader and Outer Loop Preheader blocks. Interchange the index variable pointer as well as the value stored to them.
- Interchange the Outer Loop Header and Inner Loop Header blocks, Outer Loop Latch and Inner Loop Latch blocks.
- Adjust the branch labels of all the blocks which we have swapped so that they point to correct basic blocks.

An example code. IR generated after loop interchange is attached in Part3 - LoopInterchange folder.

Implementation:

interchangeLoops(): It involves three steps, first interchange loop index variable pointer and value, interchange the basic blocks of outer loop header <-> inner loop header, inner loop latch <-> outer loop latch. Final step is to make the branch labels correct after swapping the basic blocks. Below are the helper functions used to implement the above steps.

interchangeLoopIndex(): Interchange the loop index variable pointer and value by extracting the User* from the instruction before the terminator inst of the outer loop preheader and inner loop preheader. Then we do setOperand on the User* to interchange the values.

makeBranch(BasicBlock src, BasicBlock* dest)*: It sets the branch instruction of src basic block to point to the destination basic block.

makeBranches(BasicBlock src, BasicBlock* dest1, BasicBlock* dest2)*: It sets the two branch labels of src basic block to point to dest1 and dest2 basic blocks.

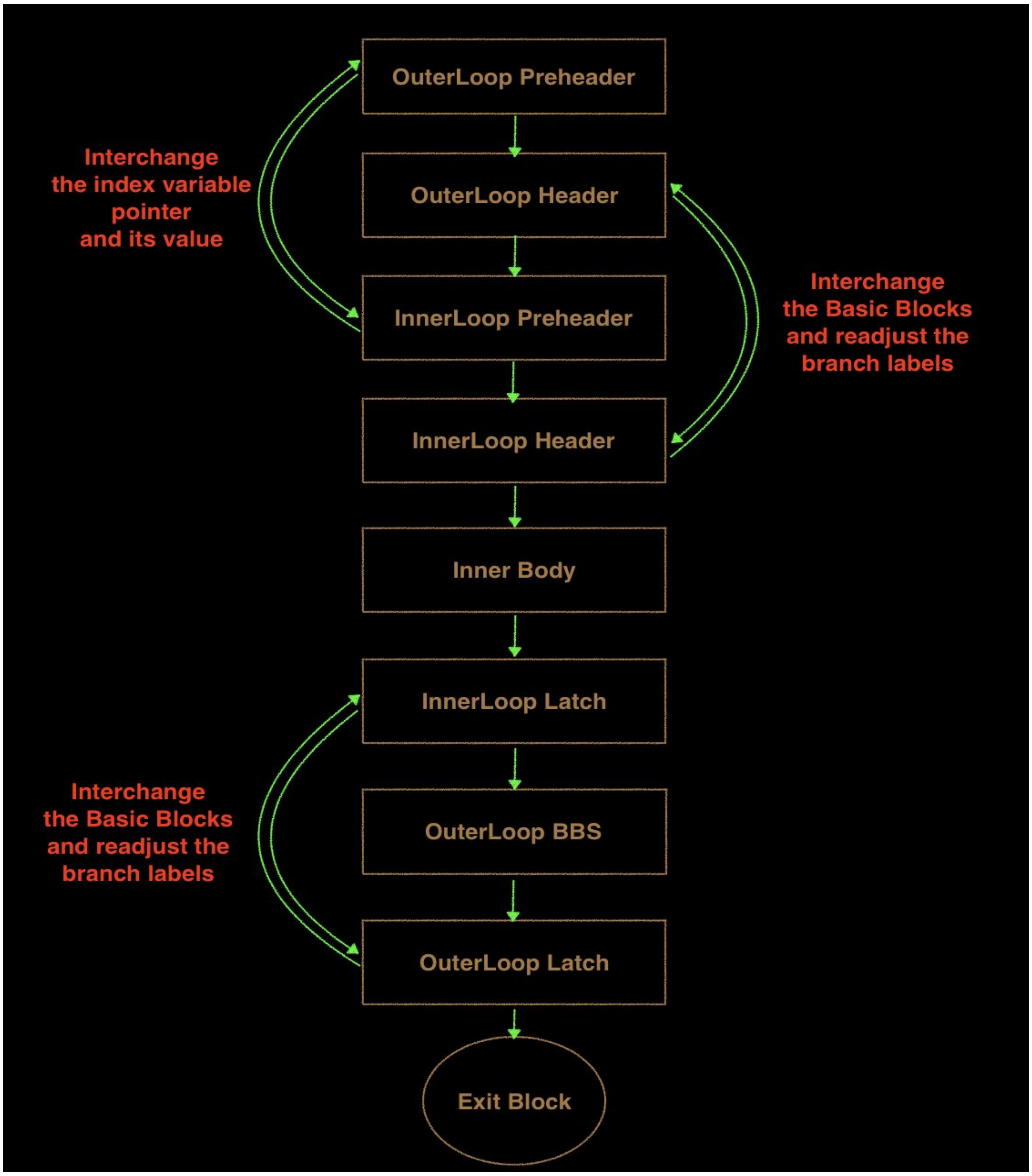
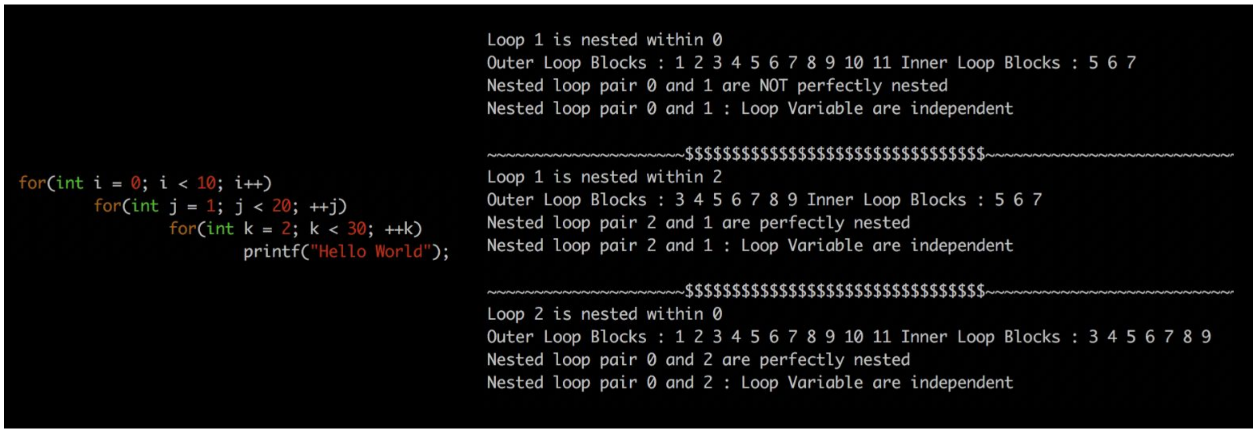

OuterLoop Preheader
OuterLoop Header
InnerLoop Preheader
InnerLoop Header
Inner Body
InnerLoop Latch
OuterLoop BBS
OuterLoop Latch
Exit Block
Interchange the index variable pointer and its value
Interchange the Basic Blocks and readjust the branch labels
Interchange the Basic Blocks and readjust the branch labels

Diagram to represent the implementation

4) The sample code, original IR of the code, modified IR after running mypass and interchanging the independent loops are attached in the folder.

<u>Result of pass:</u>

<u>Example1.c (Part4 - Example 1 folder, originalIR and the modifiedIR are also attached)</u>



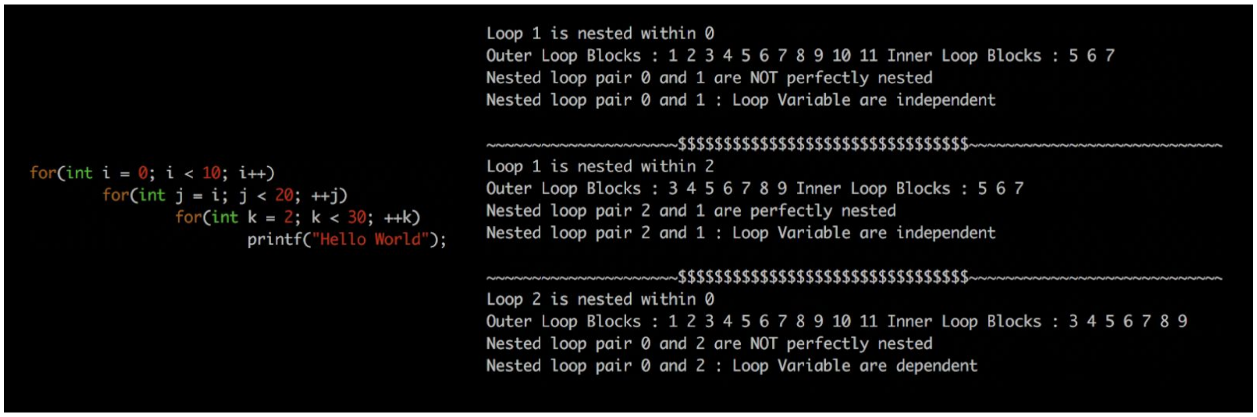
```
for(int i = 0; i < 10; i++)
        for(int j = 1; j < 20; ++j)
                for(int k = 2; k < 30; ++k)
                        printf("Hello World");
```

```
Loop 1 is nested within 0
Outer Loop Blocks : 1 2 3 4 5 6 7 8 9 10 11 Inner Loop Blocks : 5 6 7
Nested loop pair 0 and 1 are NOT perfectly nested
Nested loop pair 0 and 1 : Loop Variable are independent

~~~~~~~~~~~~~~~~~~~~~$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$~~~~~~~~~~~~~~~~~~~~~~~~~~
Loop 1 is nested within 2
Outer Loop Blocks : 3 4 5 6 7 8 9 Inner Loop Blocks : 5 6 7
Nested loop pair 2 and 1 are perfectly nested
Nested loop pair 2 and 1 : Loop Variable are independent

~~~~~~~~~~~~~~~~~~~~~$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$~~~~~~~~~~~~~~~~~~~~~~~~~~
Loop 2 is nested within 0
Outer Loop Blocks : 1 2 3 4 5 6 7 8 9 10 11 Inner Loop Blocks : 3 4 5 6 7 8 9
Nested loop pair 0 and 2 are perfectly nested
Nested loop pair 0 and 2 : Loop Variable are independent
```

<u>Example2.c (Part4 - Example 2 folder, originalIR and the modifiedIR are also attached)</u>



```
for(int i = 0; i < 10; i++)
        for(int j = i; j < 20; ++j)
                for(int k = 2; k < 30; ++k)
                        printf("Hello World");
```

```
Loop 1 is nested within 0
Outer Loop Blocks : 1 2 3 4 5 6 7 8 9 10 11 Inner Loop Blocks : 5 6 7
Nested loop pair 0 and 1 are NOT perfectly nested
Nested loop pair 0 and 1 : Loop Variable are independent

~~~~~~~~~~~~~~~~~~~~~$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$~~~~~~~~~~~~~~~~~~~~~~~~~~
Loop 1 is nested within 2
Outer Loop Blocks : 3 4 5 6 7 8 9 Inner Loop Blocks : 5 6 7
Nested loop pair 2 and 1 are perfectly nested
Nested loop pair 2 and 1 : Loop Variable are independent

~~~~~~~~~~~~~~~~~~~~~$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$~~~~~~~~~~~~~~~~~~~~~~~~~~
Loop 2 is nested within 0
Outer Loop Blocks : 1 2 3 4 5 6 7 8 9 10 11 Inner Loop Blocks : 3 4 5 6 7 8 9
Nested loop pair 0 and 2 are NOT perfectly nested
Nested loop pair 0 and 2 : Loop Variable are dependent
```

5) Advantages of **Loop Interchange** are :

- **Improves cache performance**: It is often done to ensure that the elements of a multi-dimensional array are accessed in the order in which they are present in memory, improving locality of reference. In C programming language, array elements in the same row are stored consecutively in memory (a[1,1], a[1,2], a[1,3]) − in **row-major order**. On the other hand, FORTRAN programs store array elements from the same column together (a[1,1], a[2,1], a[3,1]), using **column-major**. Thus the order of two iteration variables in the nested for loops will affect cache performance.

- **Automatic Vectorization :** It also helps automatic vectorization of the array assignments. Modern computers typically have vector operations that simultaneously perform multiple add/subtract operations via SIMD or SPMD hardware by automatic vectorization of the array elements in length four block.

Overview of the pass :

I used the code of assignment 2 to start with. For assignment 2, I already detected the nested loops and the basic blocks which are associated with the loops.

Continuing from that, the entry point to the current assignment is **do_assignment3()** function. First, I recognized the important blocks which are required for loop interchange as shown in control flow diagram of part 1, i.e Outer Loop PreHeader, Outer Loop Header, Outer Loop BBS, Outer Loop Latch, Inner Loop PreHeader, Inner Loop Header, Inner Loop Latch and stored them in respective pointers.
Then to check for perfectly nested loop, **perfectlyNested()** does the implementation. The details of this function is given above in implementation of part 1.

For checking loop dependency between the index variables, **areLoopDependent()** does the implementation. It also uses all the saved pointers of the basic blocks in the do_assignment3() function. Details of this function is given above in implementation of part 2.

For loop interchange, **interchangeLoops()** does the implementation, details are given in implementation of part 3.
We return true from the pass as we modify the IR.

Steps to run the Source code:

- In case you want to just see the output of Part 1 and Part 2, there is a function in the end of file LoopInterChange.cpp - bool changeIR(int run); just return false from it. It won't change the IR.
- In order to generate IR of a specific nested loop in case of more than 2 loops, just put a condition for if(run == order of nested loop for which you want to generate the IR) , return true. It will only change the IR for that order.
- It may happen for more than 2 loops, a pair of perfectly nested loop changed the IR, so the next pair can become not perfectly nested which originally was meant to be perfectly nested. To avoid such scenarios, you can just allow a single run to change the IR inside the changeIR function.

References:
https://github.com/llvm-mirror/llvm/blob/master/lib/Transforms/Scalar/LoopInterchange.cpp
https://en.wikipedia.org/wiki/Loop_interchange