# CS6383: Assignment 2
# Index Set Splitting
# Due Friday February 24th, 2023 at 11:59 PM

**Introduction** Index set splitting is the method of partitioning the iteration space of a loop into multiple loops such that the original iteration space is fully traversed by the set of new loops. There should be a semantic equivalence between the original and the new loops.

For example, loop in **Listing 1** can be transformed into loop in **Listing 2** by splitting the interation space of loop in 3 parts.

```
for(int i = 0; i < N; i++) {
  A[i] = A[i] / 2;
}
```

Listing 1: Sample Input

```
for(int i = 0; i < N/3; i++) {
  A[i] = A[i] / 2;
}

for(int i = N/3; i < 2N/3; i++) {
  A[i] = A[i] / 2;
}

for(int i = 2N/3; i < N; i++) {
  A[i] = A[i] / 2;
}
```

Listing 2: Sample Output

Note: CFGs corresponding to Listing 1 and 2 are shown at the end of the document.

## Part 1 [85 marks]: Index set splitting pass implementation

**Problem Statement:** You are supposed to implement index set splitting in LLVM 14.0.6.

**Procedure:**

- **Step1:**
  1. Create a new transformation pass named *-loop-splitting* with the same debug type, and define an argument *-loop-partitions=X* where X is the number of partitions of the original loop.
  2. Write a function pass for the same.

3. Loop simplify pass is a prerequisite pass for loop transformations, it will insert preheaders for the loops. You can use any of the three methods to run the prerequisite transformation passes as discussed in transformation passes lecture. (command line, addRequiredID, PassManager)

- **Step 2:**

  1. Find the inner loop
  2. We advise you to clone the loop as many times as required for the loop partitioning and then update the index bounds for each loop accordingly. However, you are free to proceed with your own implementation.

  Note: You can use *llvm/lib/Transforms/Utils/CloneFunction.cpp* for cloning the loop.

- **Step 3:**

  1. Join the individual cloned loops together by adding control flow edges. Use *-view-cfg* option to view the CFG from command line. You can also use F.viewGraph() in the function pass to view CFG in the middle of the program for debugging.

- **Step 4:**

  1. Make sure to generate valid LLVM IR so that any other LLVM transformations can be performed after your loop splitting pass. This will require invalidating any analysis information you are using in your pass. (use addPreserved cautiously).

The above procedure is for your reference in implementing ISS, you are free to follow your own approach.

**Assumptions**

1. Though index set splitting is valid on multi-dimensional loops, for this Assignment, you may consider only innermost loops.
2. Default partition number to 3 if *-loop-partitions=X* is not provided as compiler argument.

**Implementation Guidelines** Create a new directory *LoopSplitting* in lib/Transforms/ folder of the LLVM source tree. All your implementations should be in this directory as this directory will be part of your submission. You need to put header files under include/llvm/Transforms/LoopSplitting directory. Register your pass as *loop-splitting* and the module as `LoopSplitting.so` so that it can be run using scripts.

*opt -enable-new-pm=0 -load $LLVM_BUILD/lib/LoopSplitting.so -loop-splitting -loop-partitions=3 test.ll* should work.

Strictly follow the below directory structure while submitting your code. Submit new and modified files only.

```
./
└─llvm
   ├─lib
   │  └─Transforms
   │     └─LoopSplitting/
   └─include
      └─llvm
         └─Transforms
            └─LoopSplitting/
```

```
│
│
│
│
├── test/
└── README
```

**Testing** You are supposed to write non-trivial test cases to test your pass. The test cases should vary in complexity from simple to more complex ones. You need to submit at least 5 test cases in C/C++.

## Part 2 [15 marks]: Questions on Index set splitting pass

Answer the below question-based on your understanding after implementing Index set splitting pass:

1. Can Index Set Splitting (ISS) pass optimize the performance of the loop? When it is helpful, give some examples.
2. Can ISS pass result in performance degradation? Give some examples.
3. Why do we need Loop simplify pass before ISS pass? (Do not write what Loop Simplify does. It is given in above steps :)
4. Assume Loop Simplify pass is not available. Now, how can you implement ISS pass? Write an algorithm.
5. In general, can ISS expose more opportunities for optimizations? Which optimizations would benefit by performing ISS?

Add answers to the above question in your README file.

**Submission** Your submission should be a tar.gz archive with name *Asgn2_ROLLNO.tar.gz* containing the source code, header files, test-cases directory and a README file in the specified format. The README should mention all the materials that you have read/used for this assignment including LLVM documentation and source files. Mention the status of your submission, in case some part of it is incomplete. You can also include feedback, like what was challenging and what was trivial. Include files that you have added or modified. Do not include the binaries in your submission.

**Evaluation** We will test your code using a set of 50 test cases and you will be evaluated based on the number of test cases passed. Also proper commenting, code formatting along with well structured code will be part of the evaluation and fetch you more points.
Note: Non-compliance to the submission guidelines will attract strict penalty.

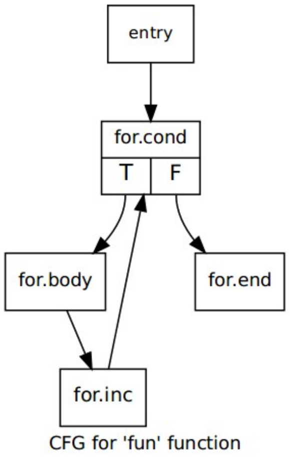

Figure 1: CFG of the original loop

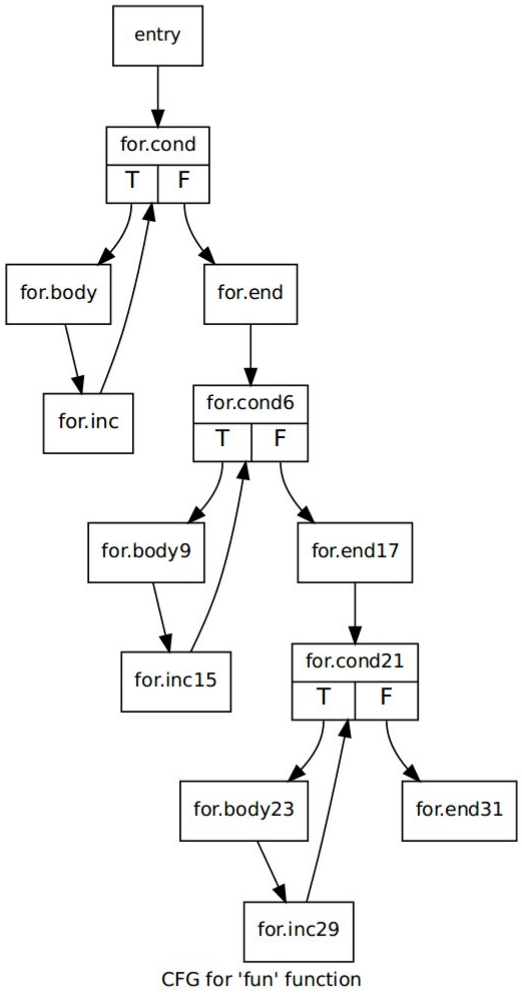

Figure 2: CFG of the splitted loops