# 

# Department of Computing

**CS 354: Compiler Construction**

**Class:** BSCS-8AB

# Lab [11]: Loop Optimization

**Date:** 17th Dec, 2021

# Time: Friday (10:00 – 1:00 & 02:00 - 5:00)

# Instructor: Dr Syed Atif Ali Shah

**Lab Instructor: M . Danyal Sadiq**

# Lab [11]: Loop Optimization

**Introduction**

Loops in a typical program takes about 90% of the total program time. This is why, loop optimization becomes extremely important in code optimization. Code Optimization is a program transformation technique, which tries to improve the code by making it consume less resources (i.e. CPU, Memory) and deliver high speed. In optimization, high-level general programming constructs are replaced by very efficient low-level programming codes. A code optimizing process in a compiler must follow the three rules given below:

a) The output code must not, in any way, change the meaning of the program.

b) Optimization should increase the speed of the program and if possible, the program should demand less number of resources.

c) Optimization should itself be fast and should not delay the overall compiling process.

**Objectives**

1. Successful implementation of programs for loop optimization techniques for compiler construction.

**Tools/Software Requirement**

1. GCC on Linux or Windows platform

**Description**

In computing, an optimizing compiler is a compiler that tries to minimize or maximize some attributes of an executable computer program. The most common requirement is to minimize the time taken to execute a program; a less common one is to minimize the amount of memory occupied. The growth of portable computers has created a market for minimizing the power consumed by a program.

Compiler optimization is generally implemented using a sequence of optimizing transformations, algorithms which take a program and transform it to produce a semantically equivalent output program that uses fewer resources. It has been shown that some code optimization problems are NP-complete, or even undecidable. In practice, factors such as the programmer's willingness to wait for the compiler to complete its task place upper limits on the optimizations that a compiler implementer might provide. (Optimization is generally a very CPU- and memory-intensive process.) In the past, computer memory limitations were also a major factor in limiting which optimizations could be performed. Because of all these factors, optimization rarely produces "optimal" output in any sense, and in fact an "optimization" may impede performance in some cases; rather, they are heuristic methods for improving resource usage in typical programs.

Widely used loop optimization techniques are:

1. Loop invariant detection and code motion
2. Induction variable elimination
3. Strength reduction in loops
4. Loop unrolling
5. Loop peeling
6. Loop fusion

Sources: <https://en.wikipedia.org/wiki/Optimizing_compiler>

<https://en.wikipedia.org/wiki/Loop_optimization>

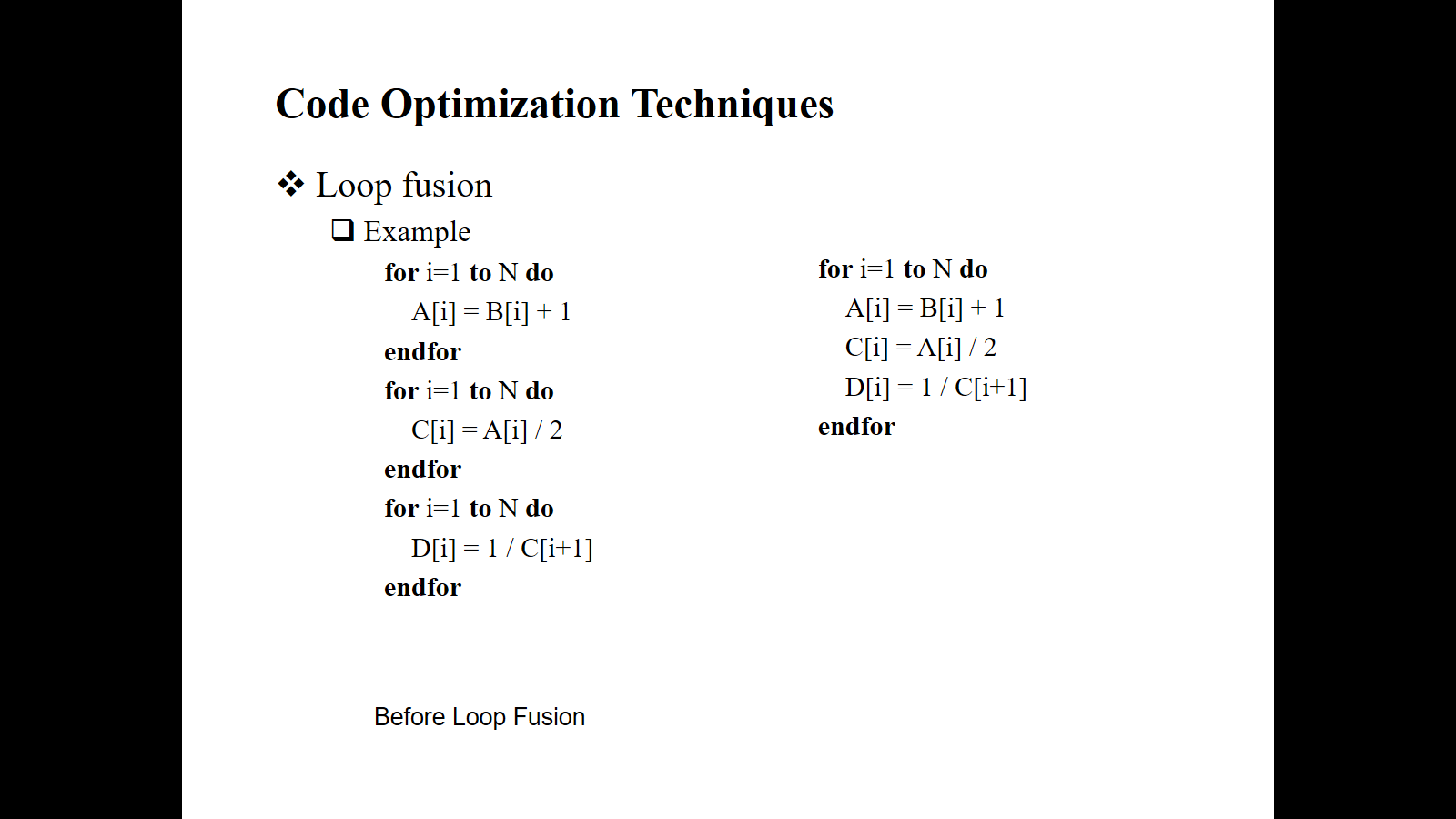
**Lab Tasks**

Figure 1. Before and after loop fusion

1. Write the code for the execution of for the programs one with loops (as given in Fig. 1) and the second one with the loop fusion, avoiding any error that may arise.

1. Write a program for the execution of the two routines so that you can compare the time complexity, executing the both routines separately, may be a million times.

**Deliverables**

You are required to upload your task (Source code and screenshot ) using the link created on LMS followed by a viva.