CS 201: Computer Systems

Assignment 4: Optimization

Academic Integrity

You may NOT, under any circumstances, begin a programming assignment by looking for completed code on StackOverflow or Chegg or any such website, which you can claim as your own. Please check out the Student Code of Conduct at PCC.

The only way to learn to code is to do it yourself. The assignments will be built from examples during the lectures, so ask for clarification during class if something seems confusing. If you start with code from another source and just change the variable names or other content to make it look original, you will receive a zero on the assignment.

I may ask you to explain your assignment verbally. If you cannot satisfactorily explain what your code does, and answer questions about why you wrote it in a particular way, then you should also expect a zero.

Purpose

The purpose of this assignment is to be able to make programs run faster via several different types of program optimization.

After completing this assignment you will be able to:

- Explore how to make programs run faster via several different types of program optimization.
- Understand the capabilities and limitations of optimizing compilers.
- Describe a variety of techniques for improving code performance.

Task

Part A: Vowel Count

- Download the file `vowel_count.c` and upload this to the Linux PCC server.

- Complete the below activities and record your answers either in a separate text file or in comments inside your modified **`vowel_count.c`** file.
- This file contains a function called **`countAllVowels()`** which takes an array of strings that are of random size with random letters in them. It then counts all the vowels (a,e,i,o,u) that are contained in all strings.
- The **`createLines()`** creates an array of random sized strings and then fills each string with lower-case letters chosen at random.
- Let's look at this program in gprof. Compile the program using the following line:
  `gcc -Og -pg vowel_count.c -o prog`

- Then run it to generate a file called **`gmon.out`**:
  `./prog`

- ***NOTE:*** it takes about 1 1/2 minutes to run so be patient. Now run gprof and redirect this to a text file so you can capture the output:
  `gprof prog > out.txt`

- **Answer the question:** What function has the most of the run time spent in this program?
- Next, we'll be using the time command to test how fast the unmodified program runs.
- Use the command:
  `time ./prog`

- Now run the program three times and record the user values; convert them to seconds. Then, find the average of these three values.
- There are some performance problems with this code. Note that although the string size is different for each string, each string doesn't change it's size while the inner loop is running.
- So start by moving the strlen() call somewhere where it isn't being called for every letter in the array (just for every string being processed).
- Now, re-compile with the same line as above and run it using time command:
  `time ./prog`

- Run the program three times and record the user time values; find the average of these three values.
- **Answer the question:** By what factor did the speed change? Use the formula (#3 average) / (#4 average). Write this in your answer sheet or in comments in your code file.

- There is also one more unnecessary call to a function inside this loop. Try getting rid of this call altogether (it's only doing a series of comparisons) and do the same thing as above.
- How much of a decrease is this from the above modified (the value from #4)?

Criteria for Success

- ❏ For information on how to do this assignment, see Module 10 materials on optimization in the D2L shell.

- ❏ Submit your modified source code files and your answers either in a separate text file or in comments inside your modified **vowel_count.c** file per the instructions in the D2L Assignment folder for assignment 4 by the due date. ***Some instructors require you to turn in your assignment on D2L while some others like it turned into the Linux server. Be sure to read the instructions in the assignment folder.***

- ❏ You must not change any part of the code except the function countAllVowels().

- ❏ The number of times each for loop runs should not change.
- ❏ The vowel count must be correct.

Part B: Memory Usage in For Loops

- Download the file **`addition.c`** and upload this to the Linux PCC server.
- Complete the below activities and record your answers either in a separate text file or in comments inside your modified **`addition.c`** file.
- This file contains a function called `addition()` which takes a two dimensional array of random numbers. It then adds all numbers within this array.
- Note that `createTable()` is there to just create the array; it has negligible effect on run time.
- Also note that in main, the `addition()` function is being called 1000 times; this is to get enough time to pass to see the difference between the original and the modified function.
- Compile the program using the following line:
  `gcc -Og addition.c -o prog`
- Then run it use time command:
  `time ./prog`
- ***NOTE:*** it takes about 1 minute to run so be patient.
- Do this three times and record the user time value each time. Average the values you get and record this in your answer sheet or in comments in your **`addition.c`** file.
- Also, look at the nested loops in addition() function and write down the **stride** for accessing this array.
- See the Optimization materials in week 11 in D2L or textbook chapter 6.2.1. These show some examples of what stride is.
- There are some performance problems with this code. Note that the stride is very large and could be greatly improved.
- Change the indexing so that the stride is much better improved.
- Now, re-compile with the same line as above and run it using time command three times and record the user time values. Average the values you get and record this in your answer sheet or in comments in your **`addition.c`** file.
- Answer the question: By what factor did the speed change? Write this in your answer sheet or in comments in **`addition.c`**.

Criteria for Success

- ❏ For information on how to do this assignment, see week 11 materials on optimization in the D2L shell.
- ❏ Submit your modified source code files and your answers either in a separate text file or in comments inside your modified **addition.c** file per the instructions in the D2L Assignment folder for assignment 4 by the due date. ***Some instructors require you to turn in your assignment on D2L while some others like it turned into the Linux server. Be sure to read the instructions in the assignment folder.***
- ❏ You must not change any part of the code except the function addition().
- ❏ The number of times each for loop runs should not change.
- ❏ The sum must be correct.

Grading Criteria

Part A:

- Compiles in gcc without errors.
- Program runs without run-time errors
- Follows directions as stated in the instructions
- Changes none of the code where comments indicate not to change.
- The for loops run the same number of times for all runs.
- The vowel count is accurate.

Part B:

- Compiles in gcc without errors.
- Program runs without run-time errors
- Changes none of the code where comments indicate not to change.
- The for loops run the same number of times for all runs.
- The sum is accurate.