**Assignment 1**

1. The charts below show the runtime (in cycles) to calculate the sum from 0 to 10000 (exclusive). They are bounded by 3 parameters:

* Is the -O2 flag enabled or not?
* Is the final sum printed?
* Is the input number 10000 hard coded or given at runtime?

|  |  |  |
| --- | --- | --- |
| **-O2 Flag Enabled** | | |
|  | **Sum Printed** | **Sum Not Printed** |
| **Hard Coded** | 77274 | 1302 |
| **User Input** | 94904 | 1289 |

|  |  |  |
| --- | --- | --- |
| **-O2 Flag Disabled** | | |
|  | **Sum Printed** | **Sum Not Printed** |
| **Hard Coded** | 137660 | 62871 |
| **User Input** | 138633 | 63010 |

Most of the discrepancy in the timings is mostly attributed to compiler optimizations. When the –O2 flag is enabled, the code is optimized to run faster. In the case where the value to sum to is hardcoded in, the –O2 flag will calculate part of the sum, if not all, during compile time. Otherwise, it will calculate only during runtime. This is why there is a difference between when the –O2 flag is enabled when is hardcoded versus when it is not enabled. When the –O2 flag is not enabled, the sum is calculated during runtime anyways. The –O2 flag also optimizes for the case when the print statement is enabled or not. When the print statement is disabled and optimization is enabled, it optimizes it such that it will not enter the loop because there is no call for the sum.

The most accurate way of timing the computation with or without optimization with the having a user input the number such that no calculation is done during compile time. The sum should also be printed such that the loop runs.

“argv[1]” is the second element of the input given to terminal when calling the program, in this case the number to sum to. “atoi(x)” converts string or char *x* to an integer. Thus “atoi(argv[1])” converts the terminal input for the number to sum until to an integer system it is a proper data type for the program to parse.