BITS PILANI, DUBAI CAMPUS II Semester 2018 – 2019 III Yr. C.S.

Course: CS F363 Compiler Construction

**Individual** LAB EXERCISE: Total Marks: 10 Weightage: **10 %**

Date for Demonstration and Record Submission:

**On or Before Monday 22 April 2019 Monday** Batch: Dr. BVK

**On or Before Wednesday 23 April 2019 Wednesday** Batch: Dr. AAJ

Record should contain **Source Code, Input** and **Output**. Record can be submitted **only after** successful demonstration.

Note: Delayed Submission will result in reduction in marks.

Copying not permissible. Issue Date: 20/04/19

**The students are required to work in their individual linux accounts in the linux server. The first statement in your program should contain your IDNO as a comment line.**

1. RUN the following program and observe the effects of different optimization levels:

and **show us the execution times with** and **without optimization**:

**#include <stdio.h>**

**#include <math.h>**

**double**

**powern (double d, unsigned n)**

**{**

**double x = 1.0;**

**unsigned j;**

**for (j = 1; j <= n; j++)**

**x \*= d;**

**return x;**

**}**

**int**

**main (void)**

**{**

**double sum = 0.0;**

**unsigned i;**

**for (i = 1; i <= 1000; i++)**

**{**

**sum += powern (i, i % 5);**

**}**

**printf ("sum = %g\n", sum);**

**return 0;**

**}**

The main program contains a loop calling the **powern** function. This function computes the ***n*-th power of a floating point number** by **repeated multiplication**--it has been chosen because it is suitable for both **inlining and loop-unrolling**. The run-time of the program can be measured using the **time** command in the GNU Bash shell.

**Test your program like this:**

**$ gcc -Wall -O0 test.c -lm**

**$ time ./a.out**

**real 0m13.388s**

**user 0m13.370s**

**sys 0m0.010s**

**$ gcc -Wall -O1 test.c -lm**

**$ time ./a.out**

**real 0m10.030s**

**user 0m10.030s**

**sys 0m0.000s**

**$ gcc -Wall -O2 test.c -lm**

**$ time ./a.out**

**real 0m8.388s**

**user 0m8.380s**

**sys 0m0.000s**

**$ gcc -Wall -O3 test.c -lm**

**$ time ./a.out**

**real 0m6.742s**

**user 0m6.730s**

**sys 0m0.000s**

**$ gcc -Wall -O3 -funroll-loops test.c -lm**

**$ time ./a.out**

**real 0m5.412s**

**user 0m5.390s**

**sys 0m0.000s**

The relevant entry in the output for comparing the speed of the resulting executables is the ‘user’ time, which gives the actual CPU time spent running the process. The other rows, ‘real’ and ‘sys’, record the total real time for the process to run (including times where other processes were using the CPU) and the time spent waiting for operating system calls. To confirm the results, run multiple times..

From the results it can be seen in this case that increasing the **optimization level** with **-O1**, **-O2** and **-O3** produces an increasing speedup, relative to the **unoptimized code compiled** with **-O0**. The additional option **-funroll-loops** produces a further speedup. The speed of the program is increased overall, when going from unoptimized code to the highest level of optimization.

**2. Take ANY one YACC program [say your previous lab exercise or calc.y the four function calculator program which you did before.**

**do these steps**

**yacc -v calc.y**

**more y.output**

**What did you observe? Write 1 para description in brief [5 to 6 sentences in your record].**

**yacc -g calc.y**

**more y.dot**

**What did you observe? Write 1 para description in brief [5 to 6 sentences in your record].**