CO460 - Assignment 0

Pin, Valgrind, Perf, gprof Tools

Instructions:

1. This is a team assignment - not more than 2 in a team.
2. Submit your code+report archive to co460.nitk@gmail.com by 18 - Jan - 2020, 9AM.

Assignment Questions:

Q1. Profile and analyze a family of 4 or more similar programs using the Intel PIN tool (https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads).
Example families: sorting programs, single (or all) source shortest path programs, tree traversal programs, compression/encryption programs, any other family of your current/past projects.

a) Collect instruction count, Instruction Address Trace, Memory Reference Trace.
b) Instruction mix must be collected with the total number of dynamic instructions, integer, floating point, load, store, branch.
c) Collect details on the total branches that are taken and total forward branches that are taken.
d) Collect Read-After-Write(RAW), Write-After-Write(WAW) and Write-After-Read(WAR) distribution in the dynamic instruction stream.

Help:
Launch and instrument an application

$ pin -t pintool -- application
pin - instrumentation engine(provided in the kit)
pintool - instrumentation tool(Write your own, or use one provided in the kit)
application - your benchmarks(binary file)

Q2. Write a program to find the largest eigenvalue of an NxN real symmetric matrix using the Power Iteration algorithm, then analyze the cache and branch prediction using **Valgrind**.

The Program
Given matrix M, a unit eigenvector v and its eigenvalue λ are defined by

$$M\ \hat{v} = b = \lambda\ \hat{v}$$

Initialize the matrix M with random values between 0 and 1. Since M should be symmetric, you only need to fill in half, using M[x][y] = M[y][x] to fill in the other half.

For the power iteration algorithm to find the largest eigenvalue and its eigenvector, you first initialize a vector, b, with random values. For each iteration, the current approximate eigenvalue is the length of b, and the current approximate eigenvector is b normalized to unit length:

$$\lambda_i = \|b_i\|$$

$$v_i = b_i / \lambda_i$$

The next approximation for b is the matrix multiplication of M with the current eigenvector:

$$b_{i+1} = M\ \hat{v}_i$$

Stop if the eigenvalue in any iteration is within 10^{-6} of the value in the previous iteration.

Valgrind
A few tutorials are: http://pages.cs.wisc.edu/~bart/537/valgrind.html, http://cs.ecs.baylor.edu/~donahoo/tools/valgrind/, https://valgrind.org/docs/manual/quick-start.html.
Compile the program, run using a command similar to:

$ valgrind --tool=cachegrind --cache-sim=yes --branch-sim=yes *program(binary file)*

There are two ways you could implement a matrix multiply for a symmetric matrix:

1. b[x] += M[x][y] * v[y]
2. b[x] += M[y][x] * v[y]

Report the cache and branch statistics for both, using a matrix size of N=1000. Reason the cache hit/miss behaviour in both the variants. Profile the code timewise and identify the hotspots using Valgrind/Pin/Combination of both.

Q3. Matrix Chain Multiplication
Matrix multiplication satisfies Associative law. i.e., (ABCD) = A(BCD) = (AB)(CD) = (ABC)D = ...
The number of arithmetic operations required for producing the product depends on how the matrices have been parenthesised.
Let, A is 5 x 10 matrix, B is a 10 x 15 matrix and C is a 15 x 20 matrix.
The number of arithmetic operations needed for the following multiplication are as follows:
(AB)C = 5*10*15 + 5*15*20 = 2250
A(BC) = 10*15*20 + 5*10*20 = 4000
First arrangement of matrices requires lesser number of multiplications.

Write a program using i) recursion, and ii) dynamic programming techniques to find the minimum number of multiplications needed to multiply a chain of matrices. Report the Performance counter stats for both the programs using the **perf** profiler. Explain the behaviour of the I/D cache hits/misses, and other interesting stats from the perf tool.

Input: An array A[] = {5,10,15,20,30} meaning, there are four arrays of dimensions 5x10, 10x15, 15x20 and 20x30.

Output: The minimum number of matrix multiplications required.

Perf tool

Install the perf package on the machine. Run the object file of your compiled program using the command below:
perf stat -e task-clock,cycles,instructions,cache-references,cache-misses ./hello

Q4. Write a program to solve travelling salesman problem using recursive functions. Profile the program using "**gprof**" tool to analyse the flat profile and call graphs of the functions used.

gprof usage: Install the gprof profiler.
1) compile the code with the below options:
gcc -Wall -pg test.c -o test
2) execute the program
./test
3) profile the program using gprof profiler
gprof test gmon.out > prof_output