A0 Submission Report

By

- Dibyadarshan Hota 16CO154
- Omkar Prabhu 16CO233

Table of Contents

Q1

We have profiled the programs

1. Bubble Sort
2. Selection Sort
3. Merge Sort
4. Quick Sort

belonging to Sorting Family for:

Note: Instrumentation code written (p1-p4.cpp) and Trace obtained for each sorting program are present in folder "q1"

a) Instruction count, Instruction Address Trace, Memory Reference Trace

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p1.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Selection sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p1.out
Image /home/dibyadarshan/Desktop/HPC/hpc-assignments/a0/q1/a.out has 271 instructions
Image /lib64/ld-linux-x86-64.so.2 has 29322 instructions
Image [vdso] has 426 instructions
==> 0x00007f6b38ee9090
==> 0x00007f6b38ee9093
Write 0x00007fff72dbdb38
==> 0x00007f6b38ee9ea0
Write 0x00007fff72dbdb30
==> 0x00007f6b38ee9ea1
==> 0x00007f6b38ee9ea4
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p1.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Bubble sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p1.out
Image /home/dibyadarshan/Desktop/HPC/hpc-assignments/a0/q1/a.out has 270 instructions
Image /lib64/ld-linux-x86-64.so.2 has 29322 instructions
Image [vdso] has 426 instructions
==> 0x00007f6a6854a090
==> 0x00007f6a6854a093
Write 0x00007ffc56197ec8
==> 0x00007f6a6854aea0
Write 0x00007ffc56197ec0
==> 0x00007f6a6854aea1
==> 0x00007f6a6854aea4
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p1.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Merge sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p1.out
Image /home/dibyadarshan/Desktop/HPC/hpc-assignments/a0/q1/a.out has 443 instructions
Image /lib64/ld-linux-x86-64.so.2 has 29322 instructions
Image [vdso] has 426 instructions
==> 0x00007fc25409e090
==> 0x00007fc25409e093
Write 0x00007fff5f7681e8
==> 0x00007fc25409eea0
Write 0x00007fff5f7681e0
==> 0x00007fc25409eea1
==> 0x00007fc25409eea4
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p1.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Quick sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p1.out
Image /home/dibyadarshan/Desktop/HPC/hpc-assignments/a0/q1/a.out has 317 instructions
Image /lib64/ld-linux-x86-64.so.2 has 29322 instructions
Image [vdso] has 426 instructions
==> 0x00007f9e05730090
==> 0x00007f9e05730093
Write 0x00007ffc75811738
==> 0x00007f9e05730ea0
Write 0x00007ffc75811730
==> 0x00007f9e05730ea1
==> 0x00007f9e05730ea4
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

Analysis:
We have obtained the instruction count using the static image of the object file, thus counting only the total instruction present, for the 4 programs as shown in the screenshots above. The instruction address along with memory reference trace (for reads and writes to a memory address belonging to the instruction logged before it) is obtained placing instrumentation code before each executed instruction.

b) Instruction mix with the total number of dynamic instructions, integer, floating-point, load, store, branch.

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p2.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Selection sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p2.out
Dynamic Instruction Count: 201622
Total Loads: 48218
Total Stores: 12883
Integer Instruction Count: 73728
Float Instruction Count: 7
Branch Instruction Count: 43491
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p2.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Bubble sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p2.out
Dynamic Instruction Count: 201937
Total Loads: 48337
Total Stores: 12932
Integer Instruction Count: 73843
Float Instruction Count: 7
Branch Instruction Count: 43456
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p2.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Merge sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p2.out
Dynamic Instruction Count: 202581
Total Loads: 48573
Total Stores: 13065
Integer Instruction Count: 73937
Float Instruction Count: 7
Branch Instruction Count: 43526
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p2.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Quick sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p2.out
Dynamic Instruction Count: 201693
Total Loads: 48218
Total Stores: 12952
Integer Instruction Count: 73731
Float Instruction Count: 7
Branch Instruction Count: 43433
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

Analysis:
We obtained the dynamic instruction count by placing the instrumentation code before every instruction executed. For integer and floating operation we compared the instruction category of the instruction with appropriate ones. Load, stores are detected using for each instruction using Pin's inspection API's. Branch instructions also can be determined by using one of the API.

c) Total branches that are taken and total forward branches that are taken

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p3.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Selection sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p3.out
Total Taken Branches: 20382
Total Taken Forward Branches: 9039
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p3.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Bubble sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p3.out
Total Taken Branches: 20374
Total Taken Forward Branches: 9034
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p3.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Merge sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p3.out
Total Taken Branches: 20429
Total Taken Forward Branches: 9077
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p3.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Quick sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p3.out
Total Taken Branches: 20372
Total Taken Forward Branches: 9045
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

Analysis:
To detect taken branches we placed the instrumentation function to be present when a branch is taken in the executing program. For identifying forward branches we compare the current branch instruction address to the address taken by the branch.

d) Read-After-Write(RAW), Write-After-Write(WAW) and Write-After-Read(WAR) distribution in the dynamic instruction stream.

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p4.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Selection sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p4.out
RAW Dependency Count: 451883
WAR Dependency Count: 432941
WAW Dependency Count: 393581
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p4.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Bubble sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p4.out
RAW Dependency Count: 453490
WAR Dependency Count: 434757
WAW Dependency Count: 394036
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p4.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Merge sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p4.out
RAW Dependency Count: 454313
WAR Dependency Count: 436984
WAW Dependency Count: 395278
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

```
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/p4.so -- ~/Desktop/HPC/hpc-assignments/a0/q1/a
.out
Quick sort, sorted array:
0.500000 1.000000 2.300000 2.500000 6.400000 25.500000 25.600000
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ head p4.out
RAW Dependency Count: 451484
WAR Dependency Count: 433159
WAW Dependency Count: 393617
dibyadarshan@hota:~/Desktop/pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/ManualExamples$ |
```

Analysis:
To help keep track of all read and writes to a memory address, we keep a map indexed by memory address with each entry storing count of instruction in which the memory address was read and written to. For any address encountered next, if this address is present in the map we identify its total read and write dependencies using the map and increment the respective dependency count. Basically, the map acts as a DP approach to get dependencies rather than iterating through the instruction trace every time.

Q2

We wrote a program to find the highest eigenvalue of an NxN real symmetric matrix using the Power iteration algorithm.

Note: Valgrind output for each case for both sections along with the power iteration algorithm (with case-specific code commented) code are present in folder "q2"

a) Valgrind output showing cache and branch statistics for the two possibilities of matrix multiplication is shown below.

Case 1: b[x] += M[x][y] * v[y]

```
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ valgrind --tool=cachegrind --cache-sim=yes --branch-sim=yes --log-file=Case1 ./a.out
Largest Eigen Value 1.031383
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ cat Case1
==13517== Cachegrind, a cache and branch-prediction profiler
==13517== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==13517== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13517== Command: ./a.out
==13517== Parent PID: 12711
==13517==
--13517-- warning: L3 cache found, using its data for the LL simulation.
==13517==
==13517== I   refs:      3,756,401,248
==13517== I1  misses:            1,222
==13517== LLi misses:            1,211
==13517== I1  miss rate:          0.00%
==13517== LLi miss rate:          0.00%
==13517==
==13517== D   refs:      1,874,839,099  (1,700,466,448 rd   + 174,372,651 wr)
==13517== D1  misses:       21,513,963  (   21,024,439 rd   +     489,524 wr)
==13517== LLd misses:       21,149,917  (   21,012,775 rd   +     137,142 wr)
==13517== D1  miss rate:           1.1% (          1.2%     +         0.3%  )
==13517== LLd miss rate:           1.1% (          1.2%     +         0.1%  )
==13517==
==13517== LL refs:          21,515,185  (   21,025,661 rd   +     489,524 wr)
==13517== LL misses:        21,151,128  (   21,013,986 rd   +     137,142 wr)
==13517== LL miss rate:            0.4% (          0.4%     +         0.1%  )
==13517==
==13517== Branches:        174,722,814  (  174,220,726 cond +     502,088 ind)
==13517== Mispredicts:         207,892  (      207,702 cond +         190 ind)
==13517== Mispred rate:            0.1% (          0.1%     +         0.0%   )
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ |
```

Case 2: b[x] += M[y][x] * v[y]

```
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ valgrind --tool=cachegrind --cache-sim=yes --branch-sim=yes --log-file=Case2 ./a.out
Largest Eigen Value 1.031383
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ cat Case2
==13322== Cachegrind, a cache and branch-prediction profiler
==13322== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==13322== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13322== Command: ./a.out
==13322== Parent PID: 12711
==13322==
--13322-- warning: L3 cache found, using its data for the LL simulation.
==13322==
==13322== I   refs:      3,756,401,248
==13322== I1  misses:            1,222
==13322== LLi misses:            1,211
==13322== I1  miss rate:          0.00%
==13322== LLi miss rate:          0.00%
==13322==
==13322== D   refs:      1,874,839,099  (1,700,466,448 rd   + 174,372,651 wr)
==13322== D1  misses:      189,837,274  (  189,186,223 rd   +     651,051 wr)
==13322== LLd misses:       21,150,712  (   21,013,570 rd   +     137,142 wr)
==13322== D1  miss rate:          10.1% (         11.1%     +         0.4%  )
==13322== LLd miss rate:           1.1% (          1.2%     +         0.1%  )
==13322==
==13322== LL refs:         189,838,496  (  189,187,445 rd   +     651,051 wr)
==13322== LL misses:        21,151,923  (   21,014,781 rd   +     137,142 wr)
==13322== LL miss rate:            0.4% (          0.4%     +         0.1%  )
==13322==
==13322== Branches:        174,722,814  (  174,220,726 cond +     502,088 ind)
==13322== Mispredicts:         207,892  (      207,702 cond +         190 ind)
==13322== Mispred rate:            0.1% (          0.1%     +         0.0%   )
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q2$ |
```

Analysis

Case 1 has considerably lower L1 - D cache misses than Case 2. This is because Case 1 accesses elements in a row-wise manner and thus expected to have better cache performance due to the matrix being stored in row order format and hence elements in the locality of a referenced location would possibly be in the cache.

b) Timewise profile and hotspots are identified

Case 1: b[x] += M[x][y] * v[y]

```
 45,000                 addition += m[i][j] * v[j];
      .             }
      .
    250             b[i] = addition;
      .         }
      .
     11         while(1) {
      .             double new_lambda;
      .
     24             double add = 0;
  2,460             for(int i = 0; i < n; ++i) {
  7,200                 add += b[i] * b[i];
      .             }
     84             new_lambda = sqrtl(add);
     96  => /build/glibc-OTsEL5/glibc-2.27/math/w_sqrtl_compat.c:sqrtl (12x)
      .
  2,460             for(int i = 0; i < n; ++i) {
  5,400                 v[i] = b[i] / new_lambda;
      .             }
  2,460             for(int i = 0; i < n; ++i) {
      .
  1,200                 double addition = 0;
      .
123,000                 for(int j = 0; j < n; ++j) {
      .                     // addition += m[j][i] * v[j];
540,000                     addition += m[i][j] * v[j];
      .                 }
      .
  3,000                 b[i] = addition;
      .             }
      .
     84             if(fabs(new_lambda - cur_lambda) < 1e-6) {
      1                 break;
      .             }
      .             else {
     22                 cur_lambda = new_lambda;
      .             }
      .         }
      .
      7         printf("Largest Eigen Value %lf\n", cur_lambda);
 67,234  => /build/glibc-OTsEL5/glibc-2.27/stdio-common/printf.c:printf (1x)
      .
      2         return 0;
     11  }
```

Case 2: b[x] += M[y][x] * v[y]

```
      .          }
      .
    250          b[i] = addition;
      .      }
      .
     11      while(1) {
      .          double new_lambda;
      .
     24          double add = 0;
  2,460          for(int i = 0; i < n; ++i) {
  7,200              add += b[i] * b[i];
      .          }
     84          new_lambda = sqrtl(add);
     96  => /build/glibc-OTsEL5/glibc-2.27/math/w_sqrtl_compat.c:sqrtl (12x)
      .
  2,460          for(int i = 0; i < n; ++i) {
  5,400              v[i] = b[i] / new_lambda;
      .          }
  2,460          for(int i = 0; i < n; ++i) {
      .
  1,200              double addition = 0;
      .
123,000              for(int j = 0; j < n; ++j) {
540,000                  addition += m[j][i] * v[j];
      .                  // addition += m[i][j] * v[j];
      .              }
      .
  3,000              b[i] = addition;
      .          }
      .
     84          if(fabs(new_lambda - cur_lambda) < 1e-6) {
      1              break;
      .          }
      .          else {
     22              cur_lambda = new_lambda;
      .          }
      .      }
      .
      7      printf("Largest Eigen Value %lf\n", cur_lambda);
 67,234  => /build/glibc-OTsEL5/glibc-2.27/stdio-common/printf.c:printf (1x)
      .
      2      return 0;
     11  }
```

Analysis:

The hotspot in the power iteration program is the matrix multiplication in the iterative part of the code. Case 1 and Case 2 do not have any differences in the hotspot location as there is a difference only in the way the matrix is accessed and it does not change the number of times a part of code is accessed, unlike cache where locality makes a difference.

Q3

We wrote programs using i) recursion and ii) dynamic programming techniques for matrix chain multiplication. The output of perf having

- Task clock
- CPU cycles
- Instructions count
- Total cache (all levels) references and misses
- L1 D-cache loads, load-misses and stores
- L1 I-Cache load-misses

on these programs are as follows:

<u>Note:</u> Perf output along with the code for matrix chain multiplication using recursion and DP are present in folder “q3”

Recursion

```
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ gcc matrixChainRecursive.c
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ sudo perf stat -e  task-clock,cycles,instructions,cache-references,cache-misses ./a.out
Operations required: 796059

 Performance counter stats for './a.out':

         15.117860      task-clock (msec)         #    0.949 CPUs utilized
       2,55,69,939      cycles                    #    1.691 GHz
       6,86,57,533      instructions              #    2.69  insn per cycle
            64,749      cache-references          #    4.283 M/sec
            27,946      cache-misses              #   43.161 % of all cache refs

       0.015933323 seconds time elapsed

dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ sudo perf stat -e  L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-icache-load-misses ./a.out
Operations required: 796059

 Performance counter stats for './a.out':

       2,68,66,412      L1-dcache-loads
            15,085      L1-dcache-load-misses     #    0.06% of all L1-dcache hits
       1,29,46,155      L1-dcache-stores
            28,046      L1-icache-load-misses

       0.020607413 seconds time elapsed

dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$
```

DP

```
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ gcc matrixChainDP.c
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ sudo perf stat -e  task-clock,cycles,instructions,cache-references,cache-misses ./a.out
Operations required: 796059

 Performance counter stats for './a.out':

          0.517431      task-clock (msec)         #    0.615 CPUs utilized
          8,71,437      cycles                    #    1.684 GHz
          7,16,542      instructions              #    0.82  insn per cycle
            45,318      cache-references          #   87.583 M/sec
            18,769      cache-misses              #   41.416 % of all cache refs

       0.000841618 seconds time elapsed

dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$ sudo perf stat -e  L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-icache-load-misses ./a.out
Operations required: 796059

 Performance counter stats for './a.out':

          2,00,003      L1-dcache-loads
            13,066      L1-dcache-load-misses     #    6.53% of all L1-dcache hits
            87,735      L1-dcache-stores
            21,563      L1-icache-load-misses

       0.000990507 seconds time elapsed

dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q3$
```

Analysis

We can see that the recursive solution has considerably higher cycle count, instructions, L1 D-cache loads and stores, etc. as recomputing overlapping subproblems leads to greater accesses to memory than the DP solution. Although the L1 D-cache load miss rate of DP is higher than the recursive solution. This is because of memoization we reference the DP matrix to check whether this solution has been computed earlier or not, this reference doesn't exploit spatial or temporal locality and possibly leading to cache miss.

Q4

We wrote a recursive TSP program running on 12 nodes. Running gprof we obtained the following flat profile and call graphs

Note: *gprof output along with the code for tsp using recursion are present in folder "q4"*

Flat Profile

```
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q4$ gcc -Wall -pg tsp.c
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q4$ ./a.out
Number of Cities: 12
0 84 87 78 16 94 36 87 93 50 22 63
28 0 91 60 64 27 41 27 73 37 12 69
68 30 0 83 31 63 24 68 36 30 3 23
59 70 68 0 94 57 12 43 30 74 22 20
85 38 99 25 0 16 71 14 27 92 81 57
74 63 71 97 82 0 6 26 85 28 37 6
47 30 14 58 25 96 0 83 46 15 68 35
65 44 51 88 9 77 79 0 89 85 4 52
55 100 33 61 77 69 40 13 0 27 87 95
40 96 71 35 79 68 2 98 3 0 18 93
53 57 2 81 87 42 66 90 45 20 0 41
30 32 18 98 72 82 76 10 28 68 57 0
Cost: 183
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q4$ gprof a.out gmon.out > prof_output
dibyadarshan@hota:~/Desktop/HPC/hpc-assignments/a0/q4$ cat prof_output
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 98.93      3.88     3.88        1     3.88     3.88  travellingSalesmanProblem
  1.28      3.93     0.05                             frame_dummy
  0.13      3.93     0.01                             main
```

Time spent in the travellingSalesmanProblem function is shown above.

Call graph

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.25% of 3.93 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     98.7    0.01    3.88                 main [1]
                3.88    0.00       1/1           travellingSalesmanProblem [2]
-----------------------------------------------
                             108505111             travellingSalesmanProblem [2]
                3.88    0.00       1/1           main [1]
[2]     98.6    3.88    0.00       1+108505111 travellingSalesmanProblem [2]
                             108505111             travellingSalesmanProblem [2]
-----------------------------------------------
                                                 <spontaneous>
[3]      1.3    0.05    0.00                 frame_dummy [3]
-----------------------------------------------
```

For the *travellingSalesmanProblem function* is called recursively by itself 108505111 times and that function is called by *main* once.