# Optimizing Matrix chain multiplication and Floyd warshell Algo

**2019101098**
**Pravalika mukkiri**

Optimizing the code for Matrix chain multiplication and Floydwarshell algorithm to find the shortest distance.

Task 0:

Understanding perf,cachegrind,prof and clock_gettime.

1. perf
   - perf is a performance analyzer in linux.
   - Makes use of data from the Performance Monitoring Unit (PMU) in CPUs.
   - To install:
     - sudo apt-get install linux-tools-’kernel version’
     - Eg : sudo apt-get install linux-tools-4.15.0.136
2. cachegrind
   - cachegrind simulates how your program interacts with a machine's cache hierarchy and (optionally) branch predictor. It simulates a machine with independent first-level instruction and data caches (I1 and D1), backed by a unified second-level cache (L2).
   - Usage:
     - valgrind --tool=cachegrind <program>
     - 
3. gprof
   - Gprof is a free profiler from GNU
   - determine where most of the execution time is spent
   - it locates code regions suited for optimization
   - analyzes connections between individual functions
   - Usage:
     - gcc -pg test_gprof.c -o test_gprof
       - -pg allows for profiling
     - ./test_gprof
       - Run once to get the gnom.out file

    - gprof test_gprof gmon.out > analysis.txt
        - Run with gprof to get the analysis report in the output file

4. Clock_gettime()
    - This is used to calculate time taken for the entire program to run. we record start and stop time and calculate the difference between them to print the total time taken by the program to run.

Task 1:
Matrix chain multiplication
For 4 matrices with result size 1000 1000

```
Time taken without optimization:  34.32 seconds
```

Optimizations used:

1. Changing the loop order to improve the chances of getting cache hit. Previous order was i,j,k. I changed it to i,k,j.

    ```
    Time taken:  9.88 seconds
    ```

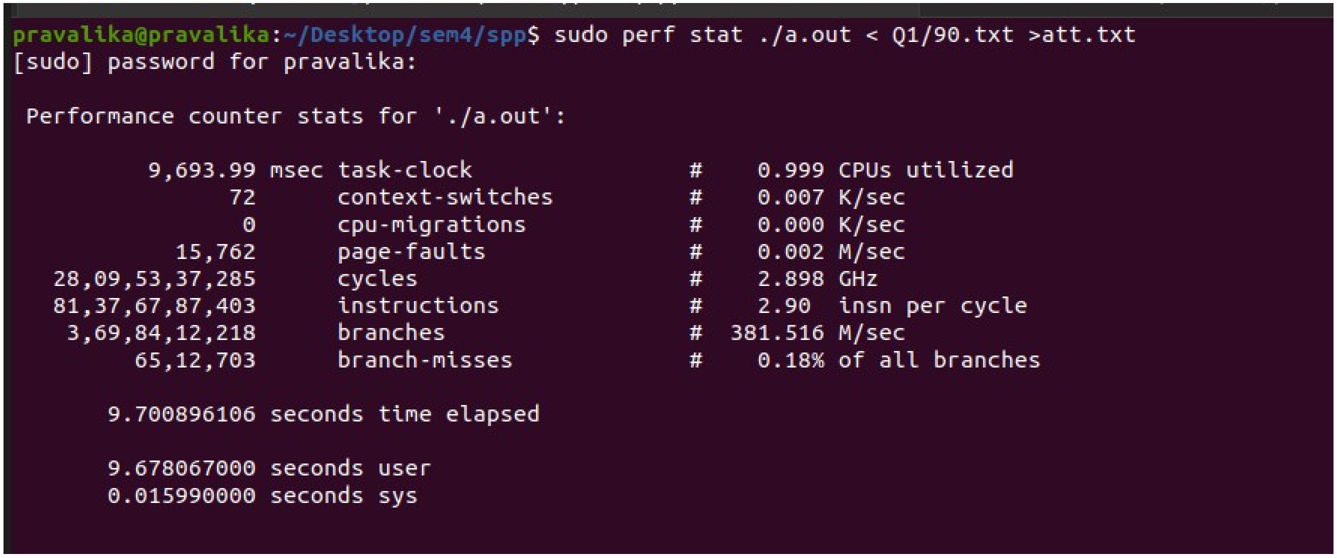
2. Pre increment over post increment Pre-increment is faster than post-increment because post increment keeps a copy of previous (existing) value and adds 1 in the existing value while pre-increment is simply adds 1 without keeping the existing value.

    ```
    Time taken:  9.51 seconds
    ```

3. Storing in 1D array  The best optimisation in terms of time, as now only 1D arrays are used instead of 2D referencing which is most time consuming.

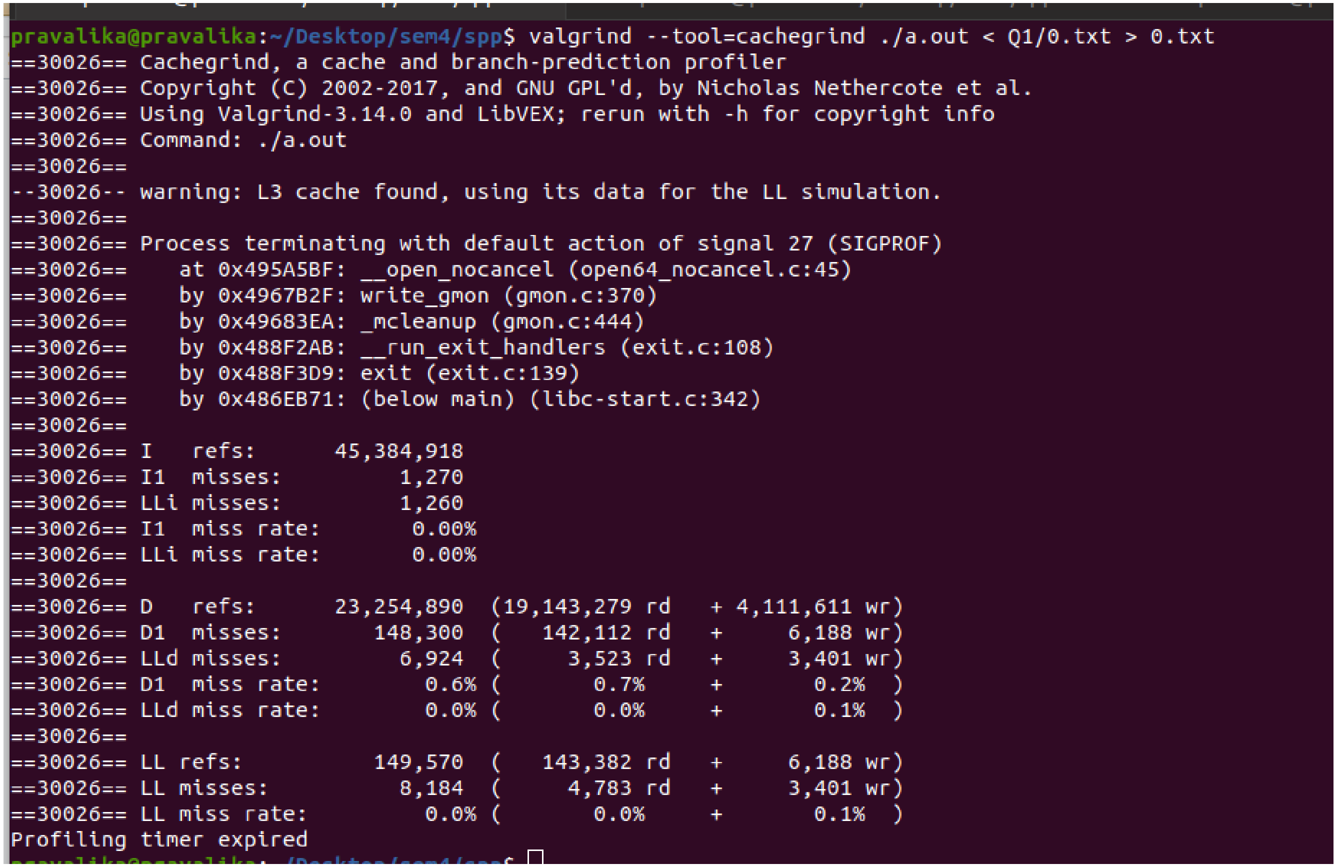
    ```
      Time taken:  9.20 seconds
    ```

perf:



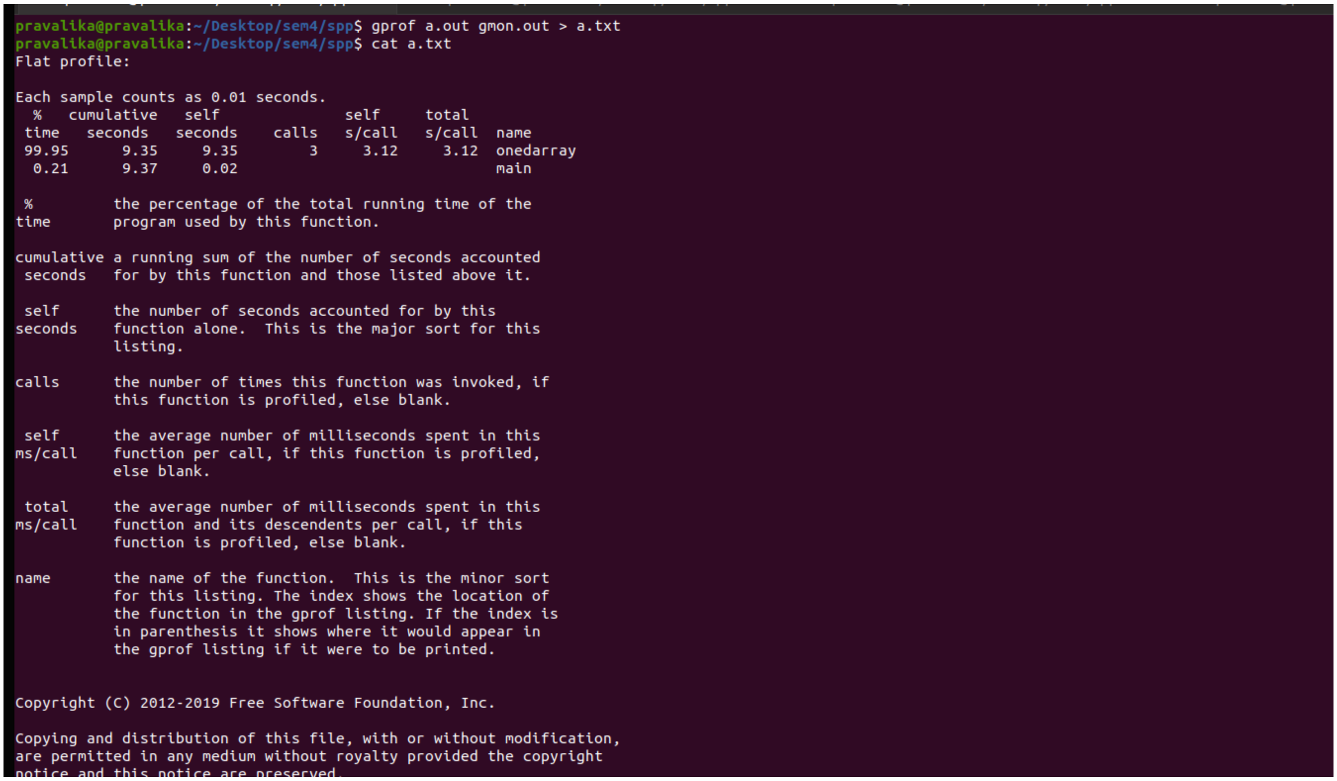
```
pravalika@pravalika:~/Desktop/sem4/spp$ sudo perf stat ./a.out < Q1/90.txt >att.txt
[sudo] password for pravalika:

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

          9,693.99 msec task-clock                #    0.999 CPUs utilized
                72      context-switches          #    0.007 K/sec
                 0      cpu-migrations            #    0.000 K/sec
            15,762      page-faults               #    0.002 M/sec
   28,09,53,37,285      cycles                    #    2.898 GHz
   81,37,67,87,403      instructions              #    2.90  insn per cycle
    3,69,84,12,218      branches                  #  381.516 M/sec
         65,12,703      branch-misses             #    0.18% of all branches

       9.700896106 seconds time elapsed

       9.678067000 seconds user
       0.015990000 seconds sys
```

Cachegrind



```
pravalika@pravalika:~/Desktop/sem4/spp$ valgrind --tool=cachegrind ./a.out < Q1/0.txt > 0.txt
==30026== Cachegrind, a cache and branch-prediction profiler
==30026== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==30026== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==30026== Command: ./a.out
==30026==
--30026-- warning: L3 cache found, using its data for the LL simulation.
==30026==
==30026== Process terminating with default action of signal 27 (SIGPROF)
==30026==    at 0x495A5BF: __open_nocancel (open64_nocancel.c:45)
==30026==    by 0x4967B2F: write_gmon (gmon.c:370)
==30026==    by 0x49683EA: _mcleanup (gmon.c:444)
==30026==    by 0x488F2AB: __run_exit_handlers (exit.c:108)
==30026==    by 0x488F3D9: exit (exit.c:139)
==30026==    by 0x486EB71: (below main) (libc-start.c:342)
==30026==
==30026== I   refs:      45,384,918
==30026== I1  misses:         1,270
==30026== LLi misses:         1,260
==30026== I1  miss rate:       0.00%
==30026== LLi miss rate:       0.00%
==30026==
==30026== D   refs:      23,254,890  (19,143,279 rd   + 4,111,611 wr)
==30026== D1  misses:       148,300  (   142,112 rd   +     6,188 wr)
==30026== LLd misses:         6,924  (     3,523 rd   +     3,401 wr)
==30026== D1  miss rate:        0.6% (       0.7%     +       0.2%  )
==30026== LLd miss rate:        0.0% (       0.0%     +       0.1%  )
==30026==
==30026== LL refs:          149,570  (   143,382 rd   +     6,188 wr)
==30026== LL misses:          8,184  (     4,783 rd   +     3,401 wr)
==30026== LL miss rate:         0.0% (       0.0%     +       0.1%  )
Profiling timer expired
pravalika@pravalika:~/Desktop/sem4/spp$
```

gprof



```
pravalika@pravalika:~/Desktop/sem4/spp$ gprof a.out gmon.out > a.txt
pravalika@pravalika:~/Desktop/sem4/spp$ cat a.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.95      9.35     9.35        3     3.12     3.12  onedarray
  0.21      9.37     0.02                             main

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2019 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

Task 2:
Floyd Warshall Algorithm
For a graph of 1912 173429 size

`Time taken without optimization:  26.28 seconds`

Optimizations used:

1. Storing in 1D array The best optimisation in terms of time, as now only 1D arrays are used instead of 2D referencing which is most time consuming.

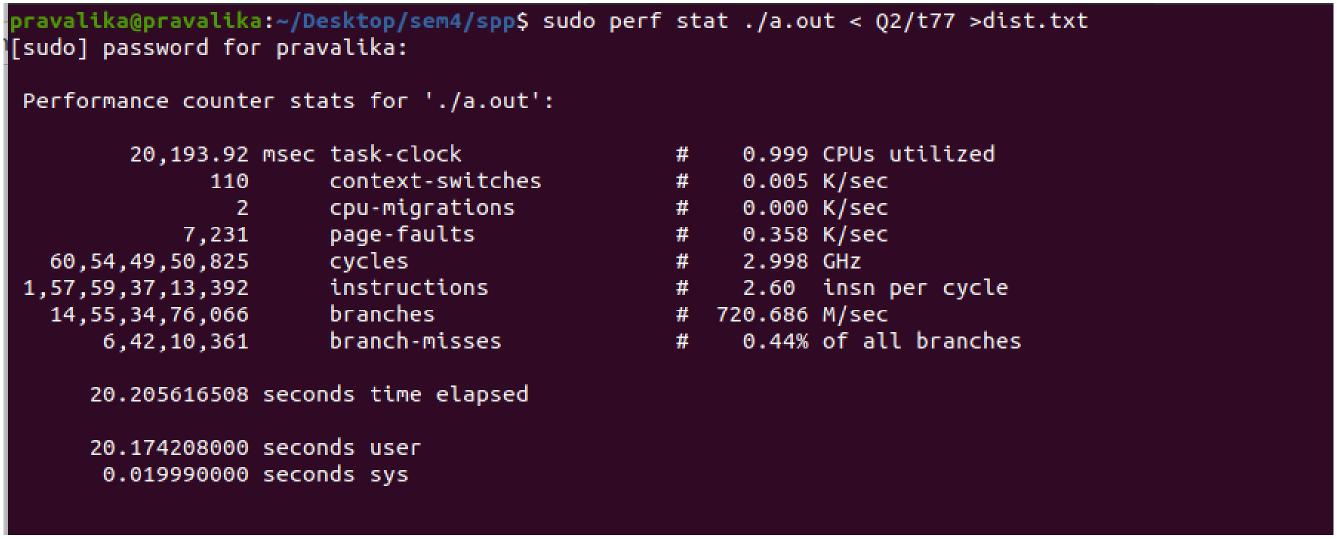
   `Time taken:  19.53 seconds`

2. Pre increment over post increment Pre-increment is faster than post-increment because post increment keeps a copy of previous (existing) value

and adds 1 in the existing value while pre-increment is simply adds 1 without keeping the existing value.

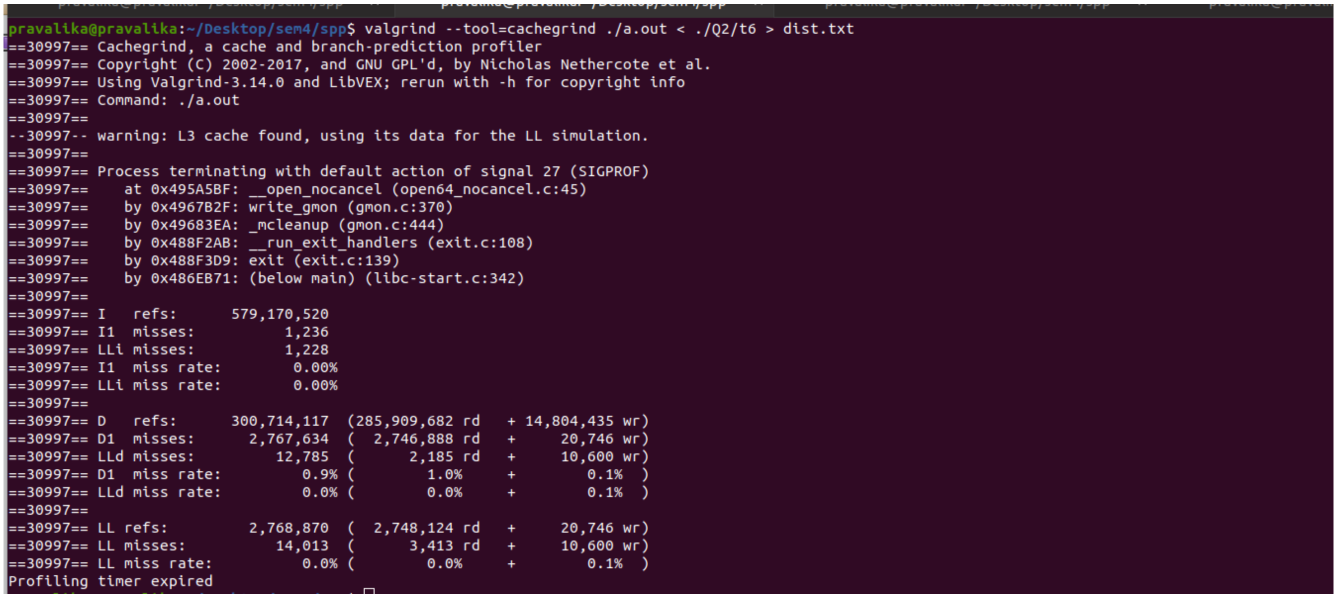
```
Time taken:  19.25 seconds
```

perf



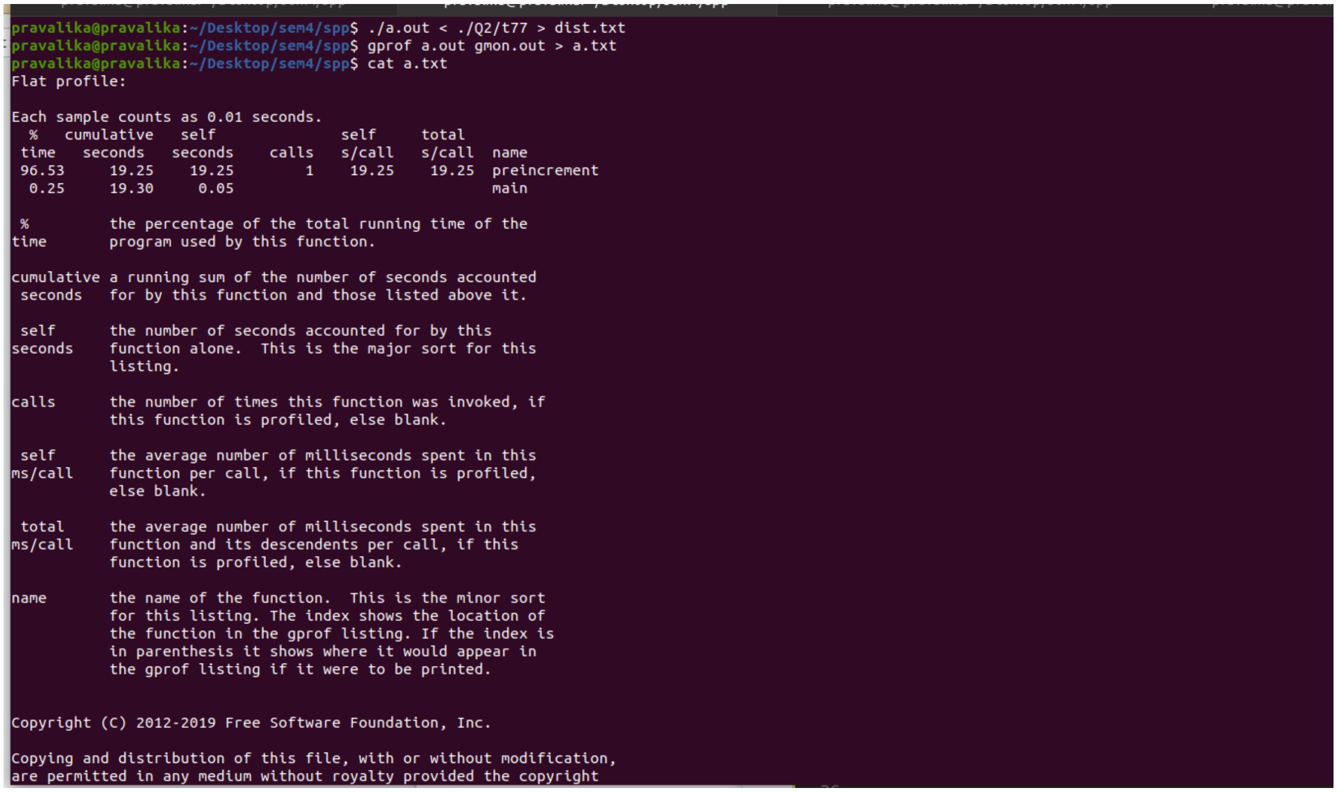
```
pravalika@pravalika:~/Desktop/sem4/spp$ sudo perf stat ./a.out < Q2/t77 >dist.txt
[sudo] password for pravalika:

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

         20,193.92 msec task-clock                #    0.999 CPUs utilized
               110      context-switches          #    0.005 K/sec
                 2      cpu-migrations            #    0.000 K/sec
             7,231      page-faults               #    0.358 K/sec
   60,54,49,50,825      cycles                    #    2.998 GHz
 1,57,59,37,13,392      instructions              #    2.60  insn per cycle
   14,55,34,76,066      branches                  #  720.686 M/sec
       6,42,10,361      branch-misses             #    0.44% of all branches

      20.205616508 seconds time elapsed

      20.174208000 seconds user
       0.019990000 seconds sys
```

cachegrind



```
pravalika@pravalika:~/Desktop/sem4/spp$ valgrind --tool=cachegrind ./a.out < ./Q2/t6 > dist.txt
==30997== Cachegrind, a cache and branch-prediction profiler
==30997== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==30997== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==30997== Command: ./a.out
==30997==
--30997-- warning: L3 cache found, using its data for the LL simulation.
==30997==
==30997== Process terminating with default action of signal 27 (SIGPROF)
==30997==    at 0x495A5BF: __open_nocancel (open64_nocancel.c:45)
==30997==    by 0x4967B2F: write_gmon (gmon.c:370)
==30997==    by 0x49683EA: _mcleanup (gmon.c:444)
==30997==    by 0x488F2AB: __run_exit_handlers (exit.c:108)
==30997==    by 0x488F3D9: exit (exit.c:139)
==30997==    by 0x486EB71: (below main) (libc-start.c:342)
==30997==
==30997== I   refs:      579,170,520
==30997== I1  misses:          1,236
==30997== LLi misses:          1,228
==30997== I1  miss rate:        0.00%
==30997== LLi miss rate:        0.00%
==30997==
==30997== D   refs:      300,714,117  (285,909,682 rd   + 14,804,435 wr)
==30997== D1  misses:      2,767,634  (  2,746,888 rd   +     20,746 wr)
==30997== LLd misses:         12,785  (      2,185 rd   +     10,600 wr)
==30997== D1  miss rate:         0.9% (        1.0%     +        0.1%  )
==30997== LLd miss rate:         0.0% (        0.0%     +        0.1%  )
==30997==
==30997== LL refs:         2,768,870  (  2,748,124 rd   +     20,746 wr)
==30997== LL misses:          14,013  (      3,413 rd   +     10,600 wr)
==30997== LL miss rate:          0.0% (        0.0%     +        0.1%  )
Profiling timer expired
```

# gprof



```
pravalika@pravalika:~/Desktop/sem4/spp$ ./a.out < ./Q2/t77 > dist.txt
pravalika@pravalika:~/Desktop/sem4/spp$ gprof a.out gmon.out > a.txt
pravalika@pravalika:~/Desktop/sem4/spp$ cat a.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls   s/call   s/call  name    
 96.53     19.25    19.25        1    19.25    19.25  preincrement
  0.25     19.30     0.05                             main

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.


Copyright (C) 2012-2019 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
```