REPORT

CSO

ASSIGNMENT-2

BY Aryan Dubey(2019113014)

TASK 0

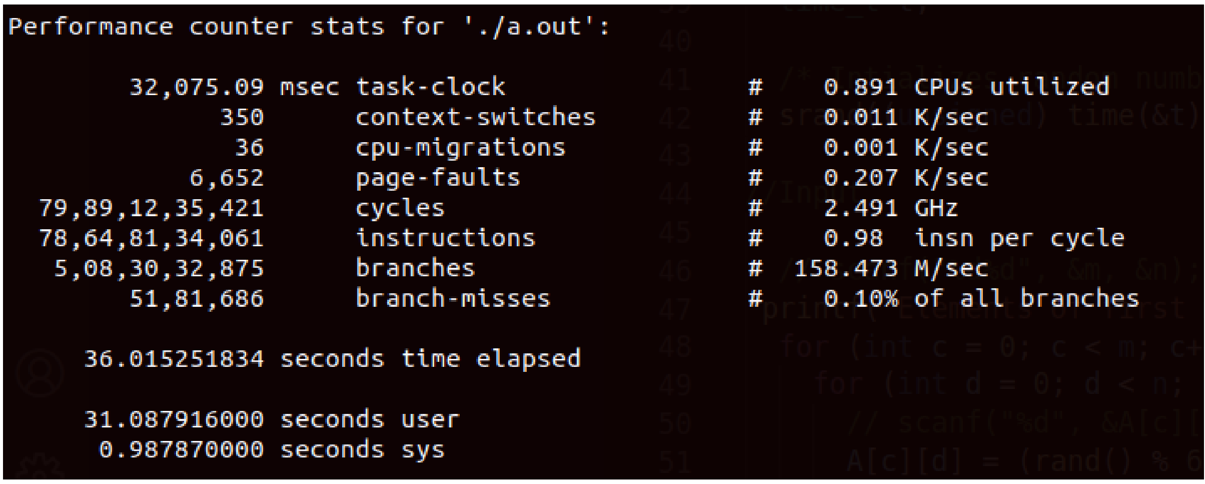
Follow the below link, it will open a pdf in which System Details are shown.

https://drive.google.com/file/d/1p8twVgngNb-dUnf_IDOnqErH5sfNR8SZ/view?usp=sharing

TASK 1

Here i am using 1500 * 1500 arrays.
Input format is - m,n,p,q

1) **Initial code**
    a) **Perf Stat**

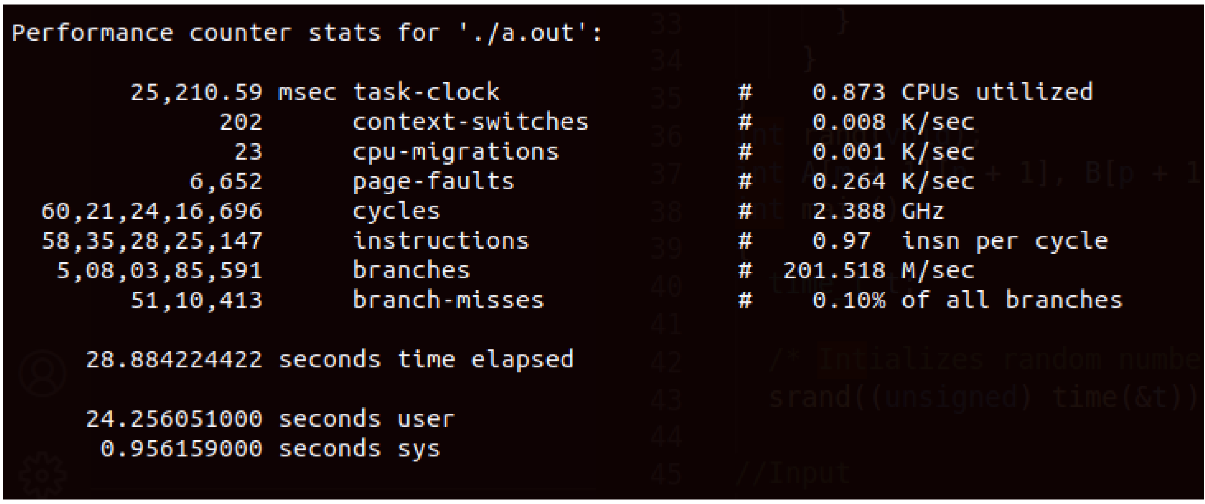


```
Performance counter stats for './a.out':

        32,075.09 msec task-clock                #    0.891 CPUs utilized
              350      context-switches          #    0.011 K/sec
               36      cpu-migrations            #    0.001 K/sec
            6,652      page-faults               #    0.207 K/sec
  79,89,12,35,421      cycles                    #    2.491 GHz
  78,64,81,34,061      instructions              #    0.98  insn per cycle
   5,08,30,32,875      branches                  #  158.473 M/sec
        51,81,686      branch-misses             #    0.10% of all branches

     36.015251834 seconds time elapsed

     31.087916000 seconds user
      0.987870000 seconds sys
```

2) **After register was used**

    When register was used, since registers are the fastest way to access memory. Thus the variable which was used many times when replaced by register int it gave a huge difference in time of around 7 secs. The perf of it is given below

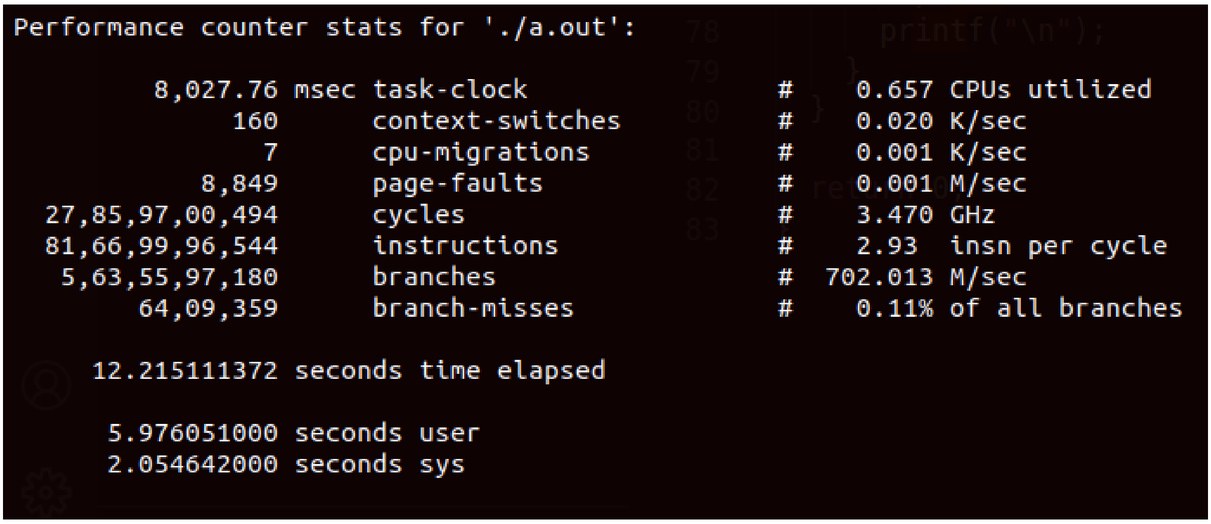
    a) **Perf Stat**



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

       25,210.59 msec task-clock                #    0.873 CPUs utilized
             202      context-switches          #    0.008 K/sec
              23      cpu-migrations            #    0.001 K/sec
           6,652      page-faults               #    0.264 K/sec
  60,21,24,16,696      cycles                   #    2.388 GHz
  58,35,28,25,147      instructions             #    0.97  insn per cycle
   5,08,03,85,591      branches                 #  201.518 M/sec
       51,10,413      branch-misses             #    0.10% of all branches

    28.884224422 seconds time elapsed

    24.256051000 seconds user
     0.956159000 seconds sys
```

3) Changing the loop order

I checked for almost all orders but the best one for my case was ckd, it reduces my execution time a lot. Since it is the fastest one.

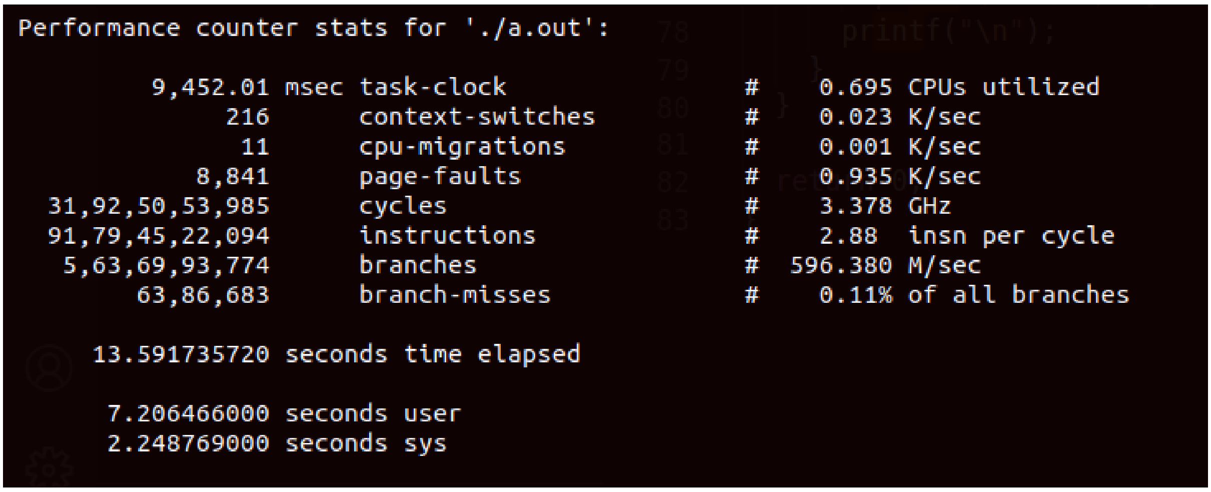


```
Performance counter stats for './a.out':

        8,027.76 msec task-clock                #    0.657 CPUs utilized
             160      context-switches          #    0.020 K/sec
               7      cpu-migrations            #    0.001 K/sec
           8,849      page-faults               #    0.001 M/sec
  27,85,97,00,494      cycles                   #    3.470 GHz
  81,66,99,96,544      instructions             #    2.93  insn per cycle
   5,63,55,97,180      branches                 #  702.013 M/sec
       64,09,359      branch-misses             #    0.11% of all branches

    12.215111372 seconds time elapsed

     5.976051000 seconds user
     2.054642000 seconds sys
```

4)Converting to 1D and unrolling

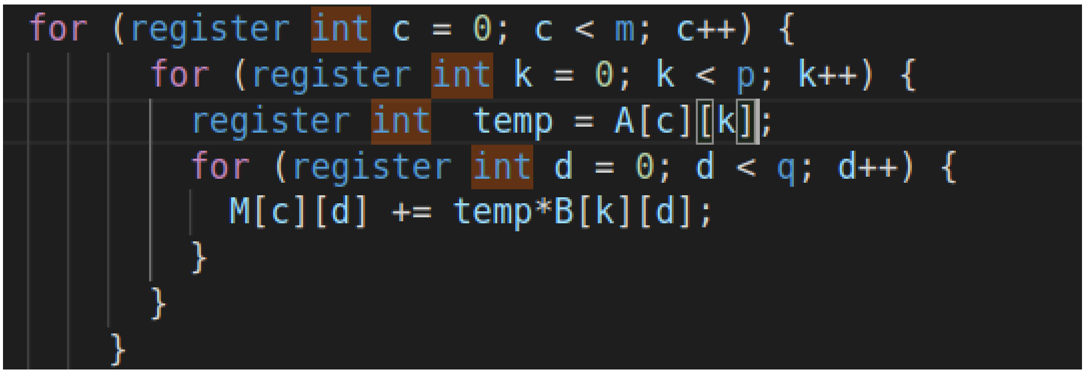
I also did optimisation by converting to 2d array to 1d array and also tried the unrolling of the array but in both cases i found the time to be slightly increased for my system.

a)Perf Stat

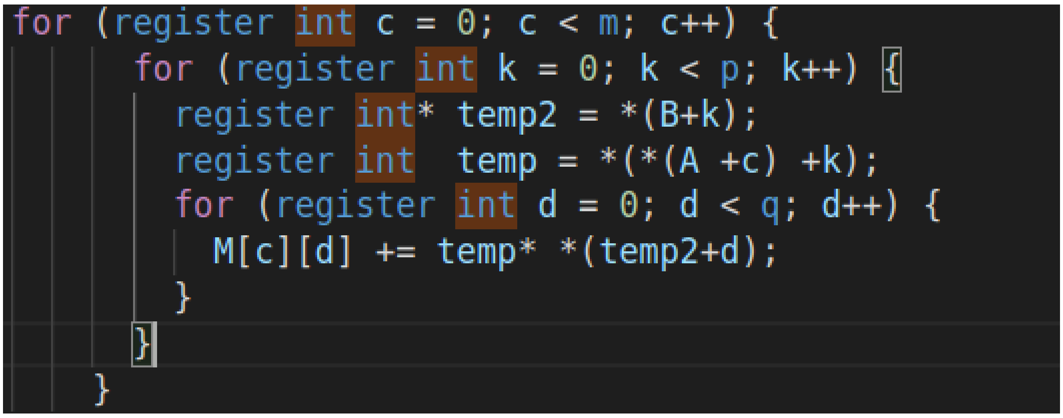


```
Performance counter stats for './a.out':

          9,452.01 msec task-clock                #    0.695 CPUs utilized
               216      context-switches          #    0.023 K/sec
                11      cpu-migrations            #    0.001 K/sec
             8,841      page-faults               #    0.935 K/sec
    31,92,50,53,985      cycles                   #    3.378 GHz
    91,79,45,22,094      instructions             #    2.88  insn per cycle
     5,63,69,93,774      branches                 #  596.380 M/sec
         63,86,683      branch-misses             #    0.11% of all branches

      13.591735720 seconds time elapsed

       7.206466000 seconds user
       2.248769000 seconds sys
```

5) Caching



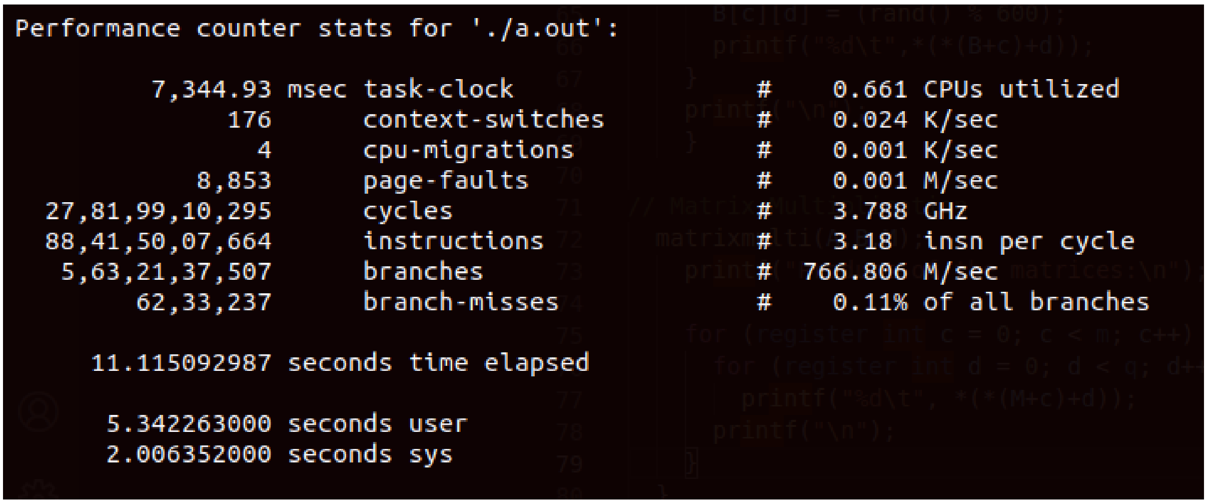
```
for (register int c = 0; c < m; c++) {
        for (register int k = 0; k < p; k++) {
          register int  temp = A[c][k];
          for (register int d = 0; d < q; d++) {
            M[c][d] += temp*B[k][d];
          }
        }
    }
```



```
for (register int c = 0; c < m; c++) {
        for (register int k = 0; k < p; k++) {
          register int* temp2 = *(B+k);
          register int  temp = *(*(A +c) +k);
          for (register int d = 0; d < q; d++) {
            M[c][d] += temp* *(temp2+d);
          }
        }
    }
```

Above we can see that in the first picture I used a temp variable for the A[c][k] because it was remaining constant in the inner loop. So I cached it. Similarly I did with the B matrix. It reduces my time by 1 min.

a) **Perf Stat**

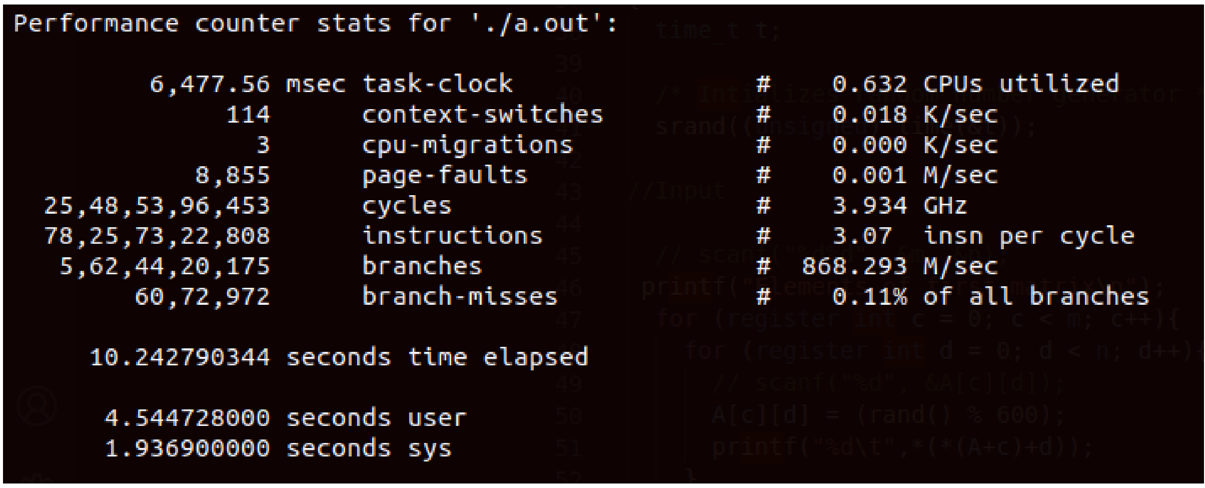


```
Performance counter stats for './a.out':

         7,344.93 msec task-clock                #    0.661 CPUs utilized
              176      context-switches          #    0.024 K/sec
                4      cpu-migrations            #    0.001 K/sec
            8,853      page-faults               #    0.001 M/sec
   27,81,99,10,295      cycles                    #    3.788 GHz
   88,41,50,07,664      instructions              #    3.18  insn per cycle
    5,63,21,37,507      branches                  #  766.806 M/sec
        62,33,237      branch-misses             #    0.11% of all branches

     11.115092987 seconds time elapsed

      5.342263000 seconds user
      2.006352000 seconds sys
```

6) Using pointers instead of arrays.

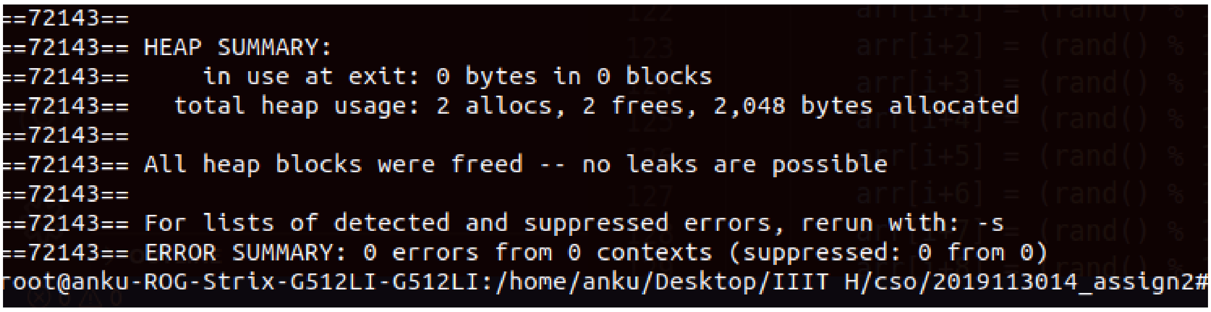
For my case after converting arrays to pointers the execution time is reduced by around 1 sec.

a) **Perf Stat**



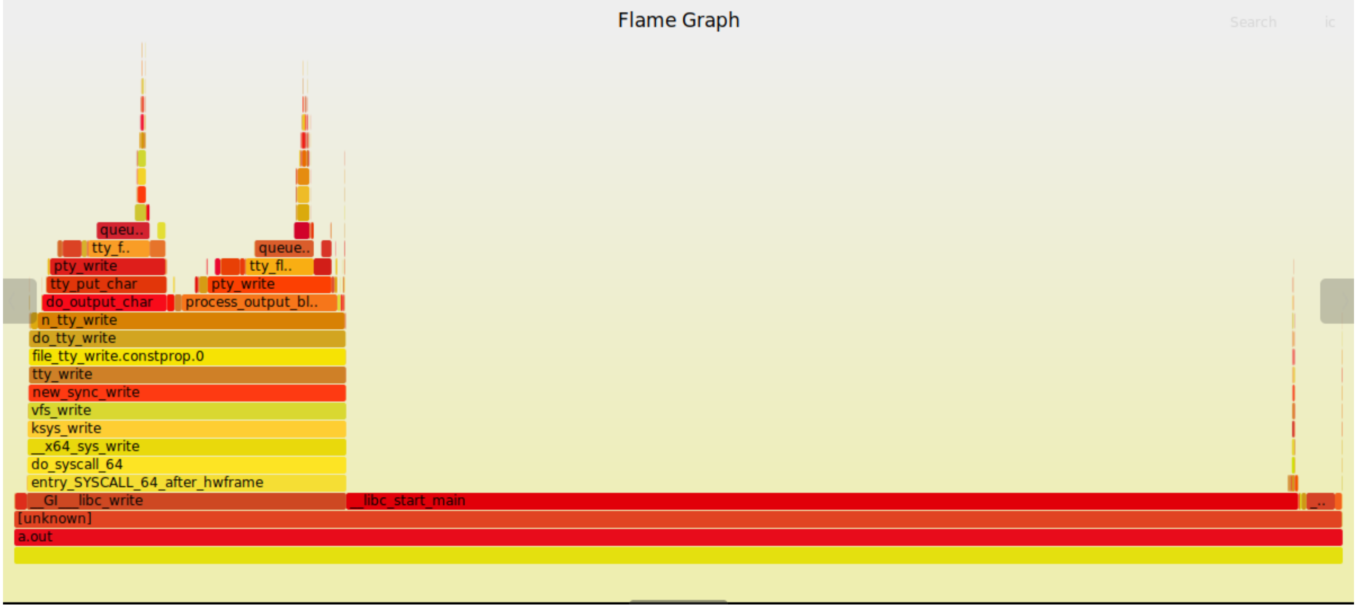
```
Performance counter stats for './a.out':

         6,477.56 msec task-clock                #    0.632 CPUs utilized
              114      context-switches          #    0.018 K/sec
                3      cpu-migrations            #    0.000 K/sec
            8,855      page-faults               #    0.001 M/sec
   25,48,53,96,453      cycles                    #    3.934 GHz
   78,25,73,22,808      instructions              #    3.07  insn per cycle
    5,62,44,20,175      branches                  #  868.293 M/sec
        60,72,972      branch-misses             #    0.11% of all branches

     10.242790344 seconds time elapsed

      4.544728000 seconds user
      1.936900000 seconds sys
```

Final Valgrind Report



```
==72143==
==72143== HEAP SUMMARY:
==72143==     in use at exit: 0 bytes in 0 blocks
==72143==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==72143==
==72143== All heap blocks were freed -- no leaks are possible
==72143==
==72143== For lists of detected and suppressed errors, rerun with: -s
==72143== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@anku-ROG-Strix-G512LI-G512LI:/home/anku/Desktop/IIIT H/cso/2019113014_assign2#
```

Flame graph

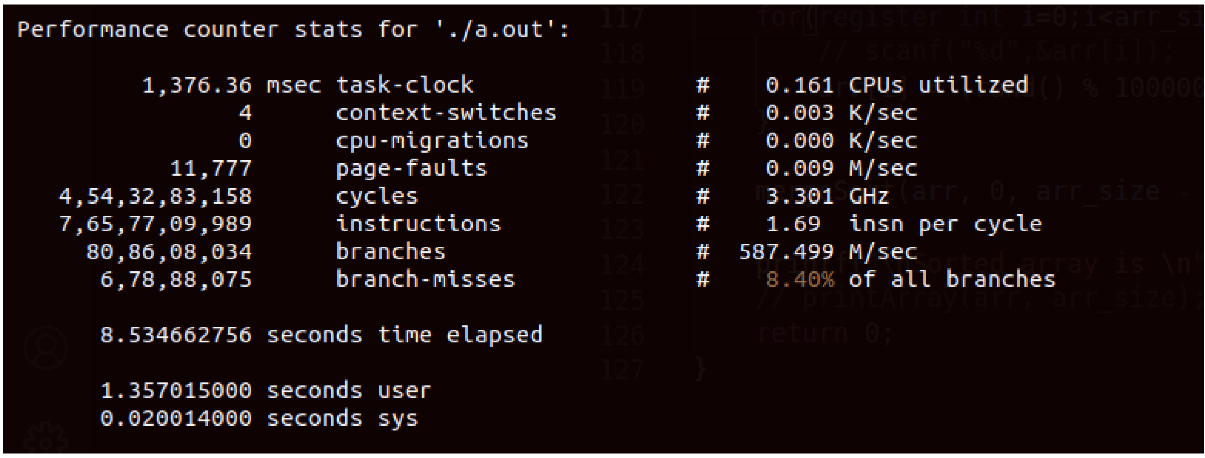


TASK 2

Here i am using array size = 6000000

1)Initial Code

a) Perf stat

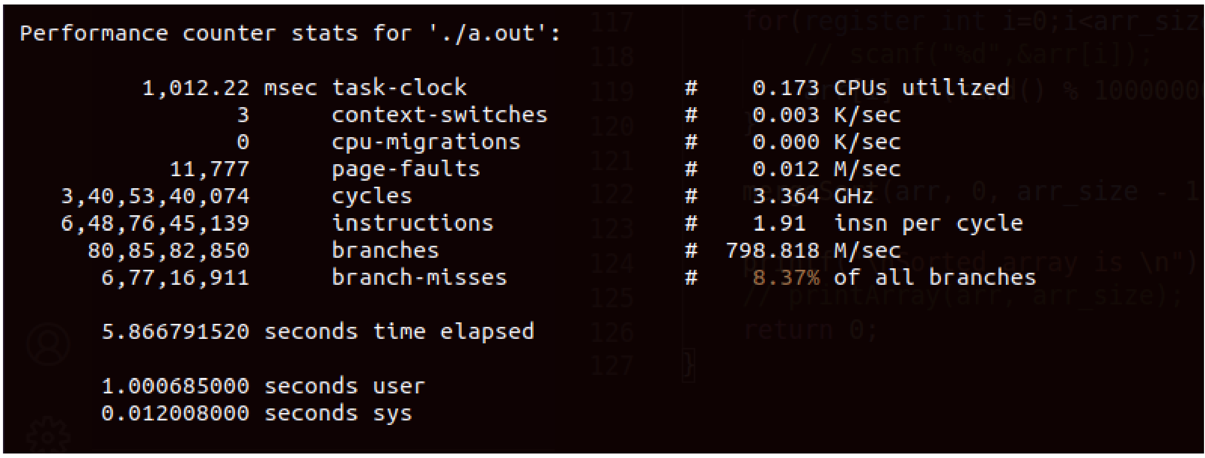


```
Performance counter stats for './a.out':

         1,376.36 msec task-clock                #    0.161 CPUs utilized
                4      context-switches          #    0.003 K/sec
                0      cpu-migrations            #    0.000 K/sec
           11,777      page-faults               #    0.009 M/sec
   4,54,32,83,158      cycles                    #    3.301 GHz
   7,65,77,09,989      instructions              #    1.69  insn per cycle
     80,86,08,034      branches                  #  587.499 M/sec
      6,78,88,075      branch-misses             #    8.40% of all branches

      8.534662756 seconds time elapsed

      1.357015000 seconds user
      0.020014000 seconds sys
```

2)After register was used

When register was used, since registers are the fastest way to access memory. Thus the variable which was used many times when replaced by register int it gave a huge difference in time of around 7 secs. The perf of it is given below

a)Perf Stat

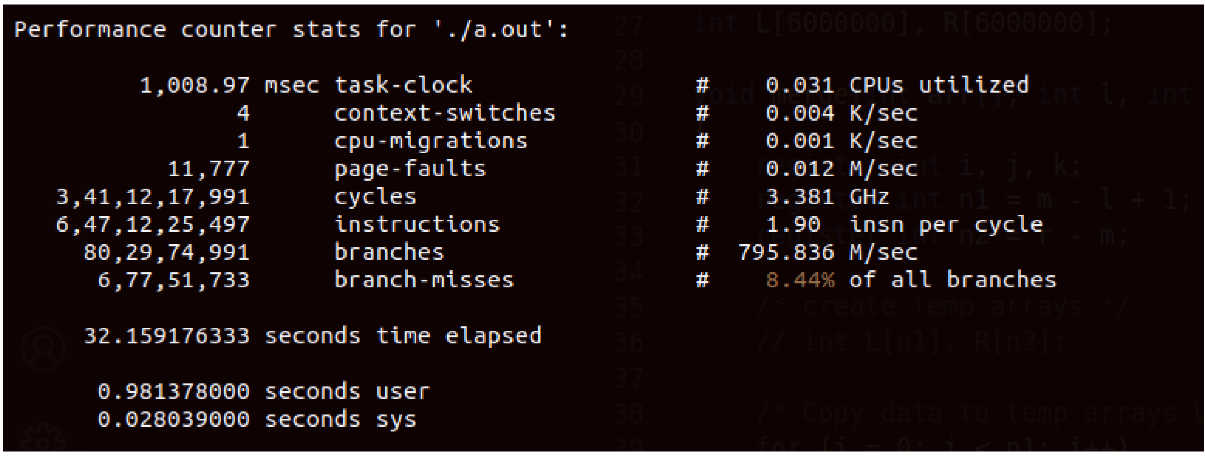


```
Performance counter stats for './a.out':

         1,012.22 msec task-clock                #    0.173 CPUs utilized
                3      context-switches          #    0.003 K/sec
                0      cpu-migrations            #    0.000 K/sec
           11,777      page-faults               #    0.012 M/sec
   3,40,53,40,074      cycles                    #    3.364 GHz
   6,48,76,45,139      instructions              #    1.91  insn per cycle
     80,85,82,850      branches                  #  798.818 M/sec
      6,77,16,911      branch-misses             #    8.37% of all branches

      5.866791520 seconds time elapsed

      1.000685000 seconds user
      0.012008000 seconds sys
```

3)After using unrolling

After using unrolling since loop check conditions are removed and hence it improves the execution time by some extent.

a) **Perf Stat**

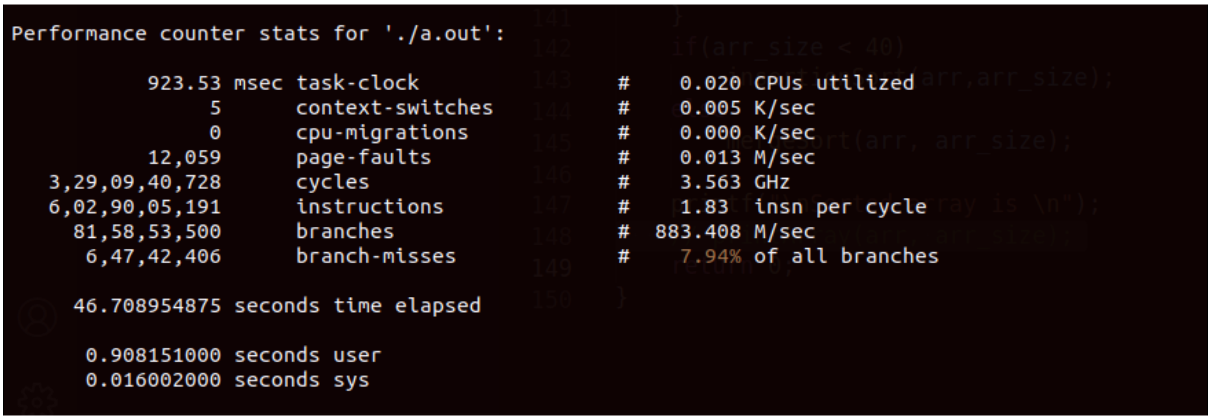


```
Performance counter stats for './a.out':

         1,008.97 msec task-clock                #    0.031 CPUs utilized
                4      context-switches          #    0.004 K/sec
                1      cpu-migrations            #    0.001 K/sec
           11,777      page-faults               #    0.012 M/sec
   3,41,12,17,991      cycles                    #    3.381 GHz
   6,47,12,25,497      instructions              #    1.90  insn per cycle
     80,29,74,991      branches                  #  795.836 M/sec
      6,77,51,733      branch-misses             #    8.44% of all branches

     32.159176333 seconds time elapsed

      0.981378000 seconds user
      0.028039000 seconds sys
```

4) After using iterative mergesort

It is because iterative merge sort avoids recursive function calls and thus avoiding overheads.

a)Perf Stat



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

           923.53 msec task-clock                #    0.020 CPUs utilized
                5      context-switches          #    0.005 K/sec
                0      cpu-migrations            #    0.000 K/sec
           12,059      page-faults               #    0.013 M/sec
   3,29,09,40,728      cycles                    #    3.563 GHz
   6,02,90,05,191      instructions              #    1.83  insn per cycle
     81,58,53,500      branches                  #  883.408 M/sec
      6,47,42,406      branch-misses             #    7.94% of all branches

     46.708954875 seconds time elapsed

      0.908151000 seconds user
      0.016002000 seconds sys
```

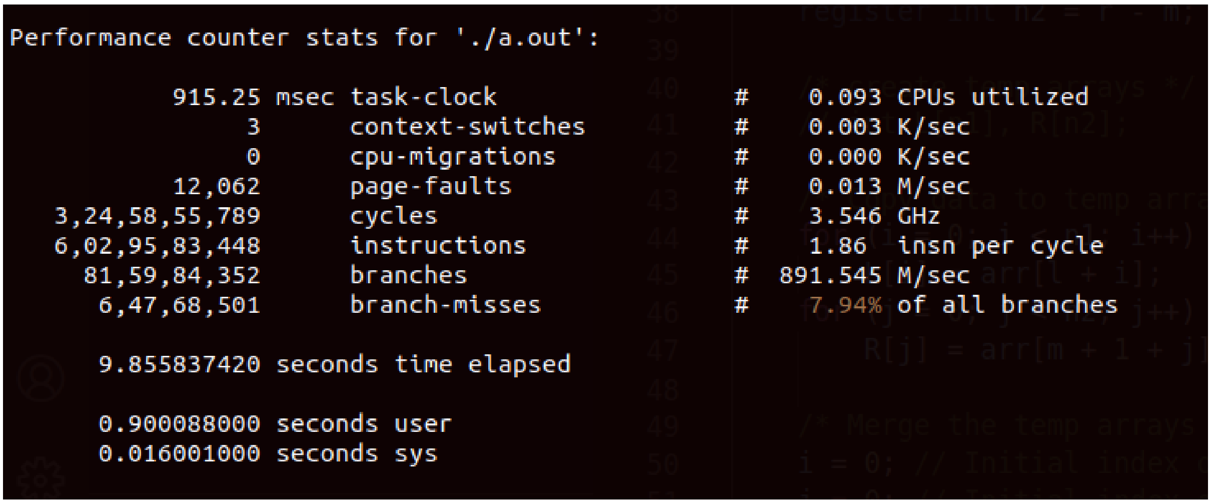
5) After using insertion sort for small size

Since insertion sort runs faster than merge sort for array size less around 40, so it will improve performance for lower array size.

6)After using pointers instead of array

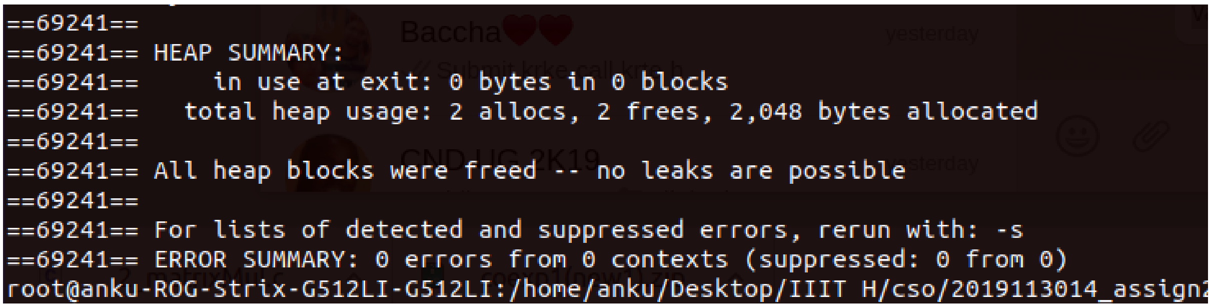
For my case pointers access memory faster than the array that’s why slight improvement was seen after using pointers instead of arrays.

a)Perf Stat



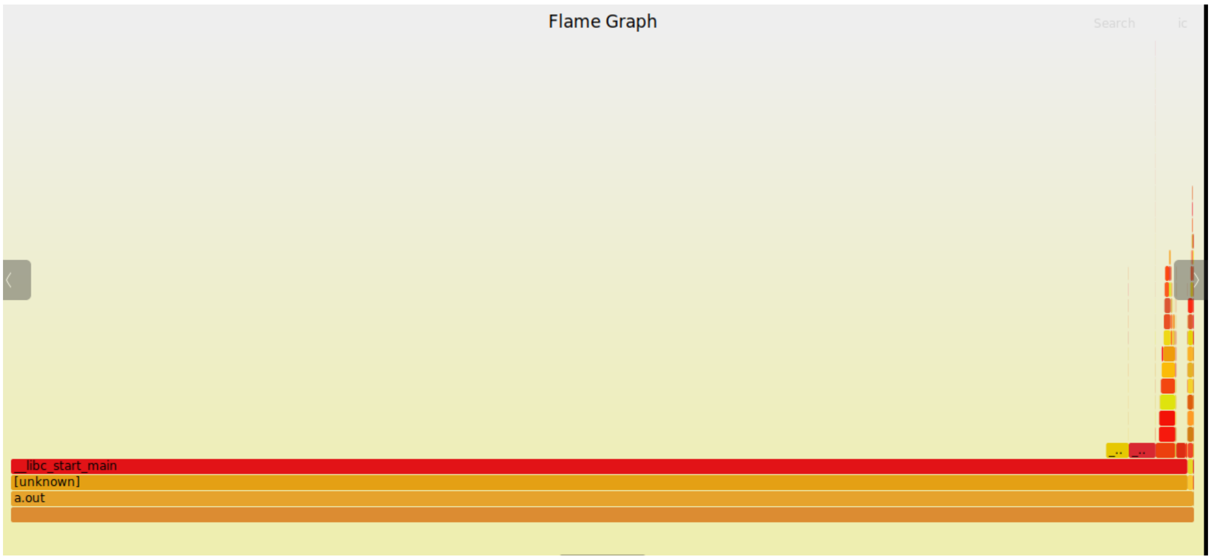
```
Performance counter stats for './a.out':

         915.25 msec task-clock                #    0.093 CPUs utilized
              3      context-switches          #    0.003 K/sec
              0      cpu-migrations            #    0.000 K/sec
         12,062      page-faults               #    0.013 M/sec
   3,24,58,55,789      cycles                  #    3.546 GHz
   6,02,95,83,448      instructions            #    1.86  insn per cycle
     81,59,84,352      branches                #  891.545 M/sec
      6,47,68,501      branch-misses           #    7.94% of all branches

      9.855837420 seconds time elapsed

      0.900088000 seconds user
      0.016001000 seconds sys
```

Final Valgrind Report



```
==69241==
==69241== HEAP SUMMARY:
==69241==     in use at exit: 0 bytes in 0 blocks
==69241==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==69241==
==69241== All heap blocks were freed -- no leaks are possible
==69241==
==69241== For lists of detected and suppressed errors, rerun with: -s
==69241== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@anku-ROG-Strix-G512LI-G512LI:/home/anku/Desktop/IIIT H/cso/2019113014_assign2
```

Flame graph


Flame Graph
Search
ic
_..
_..
__libc_start_main
[unknown]
a.out