Computer Organization

Sarvesh Anand Mankar
142203013
TY Comp Div-2, T4 Batch

Title: Study and experimentation using perf tool to observe different statistics of a program.

cpu-cycles:

The "cpu-cycles" hardware event calculates how many CPU clock cycles were used by a system or programme overall during a particular performance monitoring session.

Use Case: It offers a fundamental measure of the CPU's overall processing load and is helpful in evaluating the effectiveness of code execution and pinpointing potential performance bottlenecks associated with CPU utilisation.

For instance, keeping track of "cpu-cycles" can reveal areas of code that use the CPU excessively, enabling performance improvement attempts.

branches-instructions:

The "branch-instruction" hardware event keeps track of how many branch instructions a system or programme has executed over a performance monitoring session.

Use Case: This event is useful for comprehending the efficiency of branch prediction and programme control flow. It assists in determining the frequency of taking or not taking branches, which is essential for streamlining code execution and reducing branch mispredictions.

As an illustration, by keeping an eye on "branch-instructions," you can identify code segments with a lot of branches that might cause performance problems and optimise them to speed up overall execution.

bus-cycles:

The hardware event "bus-cycles" counts all bus cycles that occur during a performance monitoring session. Bus cycles describe the amount of time the CPU spends using the system bus to access data from memory or other peripherals.

Use Case: This event sheds light on memory access patterns and bottlenecks that are connected to memory. It aids in memory consumption optimisation and the detection of data access inefficiencies.

Example: Monitoring "bus-cycles" can assist you in identifying instances where the CPU spends an excessive amount of time waiting for data to be fetched from memory, which can be a performance bottleneck. Programming with cache awareness and memory optimisation will benefit from knowing this information.

instructions:

Description: The "instructions" hardware event counts the total number of instructions executed by a program or system during a performance monitoring session.

Use Case: This event provides a fundamental metric for understanding program execution. It helps in assessing the workload on the CPU and can be used to evaluate the efficiency of code execution.

Example: Monitoring "instructions" can be beneficial for profiling applications to identify code segments that consume a significant number of instructions. Optimizing such code sections can lead to improved overall performance and resource utilization.

branch-misses:

Description: The "branch-misses" hardware event tracks how often branch instructions during a performance monitoring session lead to incorrect predictions. Branch mispredictions happen when the CPU's branch predictor predicts the result of a branch instruction incorrectly.

Use Case: Analysing this incident is essential for determining how effective the CPU's branch prediction system is. High branch misprediction rates may point to portions of the code that require optimisation to enhance control flow and lower errors.

Example: Keeping track of "branch-misses," you can spot code portions that frequently forecast branches incorrectly. The software will run more effectively and perform better as a whole if certain areas are optimised.

cache-misses:

Description: The amount of memory access operations during a performance monitoring session that result in cache misses is counted by the "cache-misses" hardware event. When data is missing from the CPU's caches and must be fetched from a higher level memory hierarchy (such as main memory), this is known as a cache miss.

Use Case: Understanding memory access patterns and the efficiency of cache usage depends on keeping track of cache misses. Inefficient data access can result in high cache miss rates, which can cause performance bottlenecks.

Example: Keeping track of "cache-misses," you can identify the parts of the code that regularly seek data that is not cached. Programme performance can be greatly improved by optimising these parts by increasing data locality or minimising needless memory accesses.

cache-reference:

Description: The "cache-reference" hardware event counts the total number of memory access operations that reference the cache during a performance monitoring session. This event measures how often data is accessed in the CPU's cache.

Use Case: Monitoring cache references is essential for evaluating the efficiency of cache utilization. It provides insights into how frequently data is reused from the cache, which is crucial for optimizing memory access patterns.

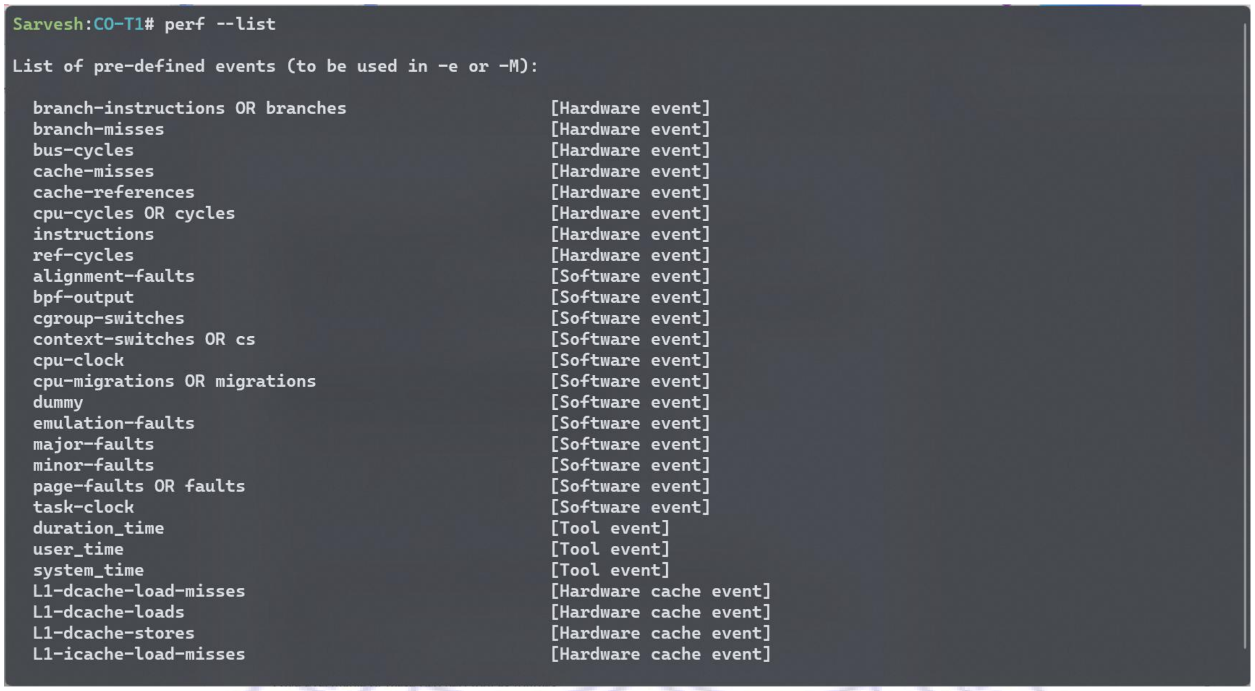
Example: By tracking "cache-reference" events, you can identify code sections that benefit from cache usage, as well as areas that may need improvement in terms of data reuse. Optimizing cache-friendly code can lead to better overall performance.

ref-cycles:

Description: The "ref-cycles" hardware event counts reference cycles, which represent the total number of clock cycles during which the CPU's resources are actively used for executing instructions or handling memory accesses.

Use Case: This event is valuable for understanding how efficiently the CPU is being utilized and can provide insights into overall system performance.

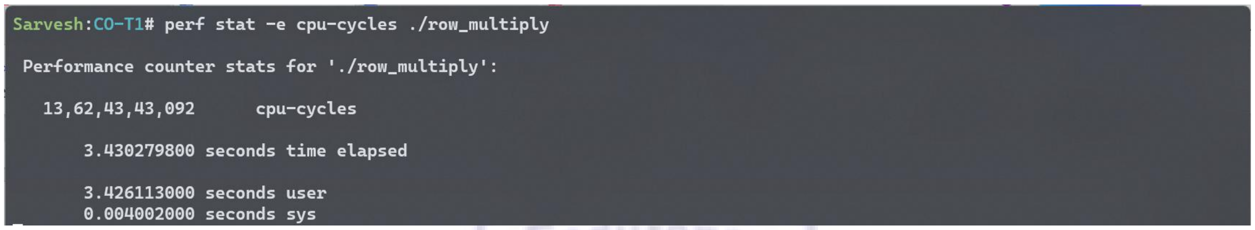
Example: Monitoring "ref-cycles" can help you assess whether the CPU is spending a significant amount of time idle or executing instructions. High reference cycle counts can indicate that the CPU is not fully utilized, while low counts may suggest efficient resource usage. This information is useful for optimizing code and improving system responsiveness.



```
Sarvesh:CO-T1# perf --list

List of pre-defined events (to be used in -e or -M):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]
  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  cgroup-switches                                    [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]
  duration_time                                      [Tool event]
  user_time                                          [Tool event]
  system_time                                        [Tool event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
```

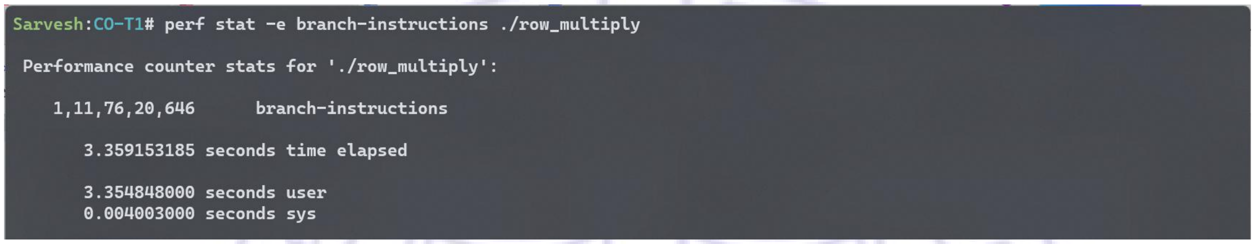
Hardware Commands

cpu-cycles



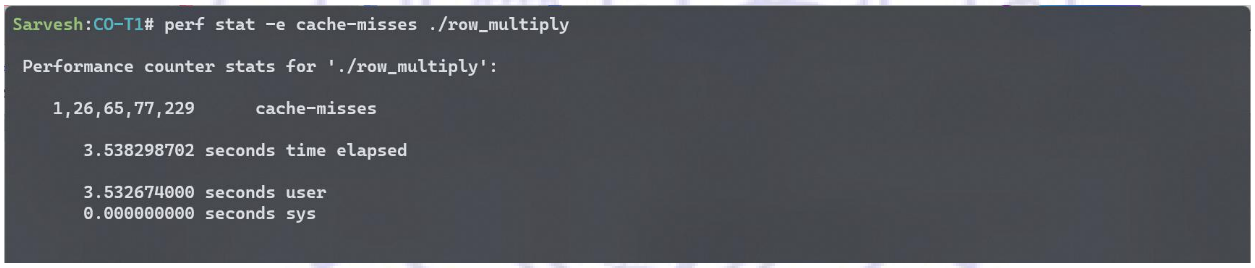
```
Sarvesh:CO-T1# perf stat -e cpu-cycles ./row_multiply

 Performance counter stats for './row_multiply':

   13,62,43,43,092      cpu-cycles

       3.430279800 seconds time elapsed

       3.426113000 seconds user
       0.004002000 seconds sys
```

branch-instrcutions



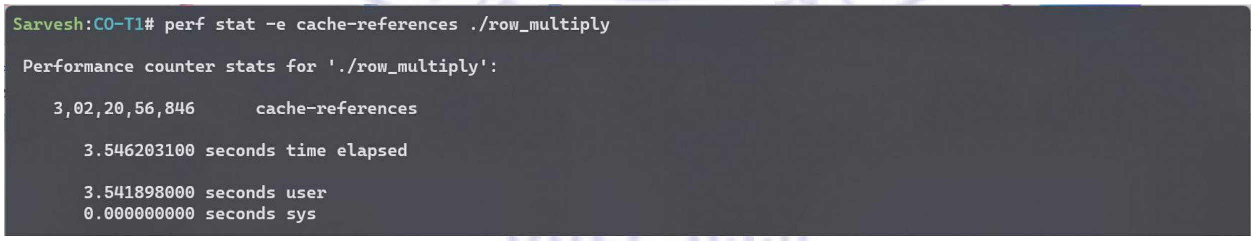
```
Sarvesh:CO-T1# perf stat -e branch-instructions ./row_multiply

 Performance counter stats for './row_multiply':

    1,11,76,20,646      branch-instructions

       3.359153185 seconds time elapsed

       3.354848000 seconds user
       0.004003000 seconds sys
```

cache-misses



```
Sarvesh:CO-T1# perf stat -e cache-misses ./row_multiply

 Performance counter stats for './row_multiply':

    1,26,65,77,229      cache-misses

       3.538298702 seconds time elapsed

       3.532674000 seconds user
       0.000000000 seconds sys
```

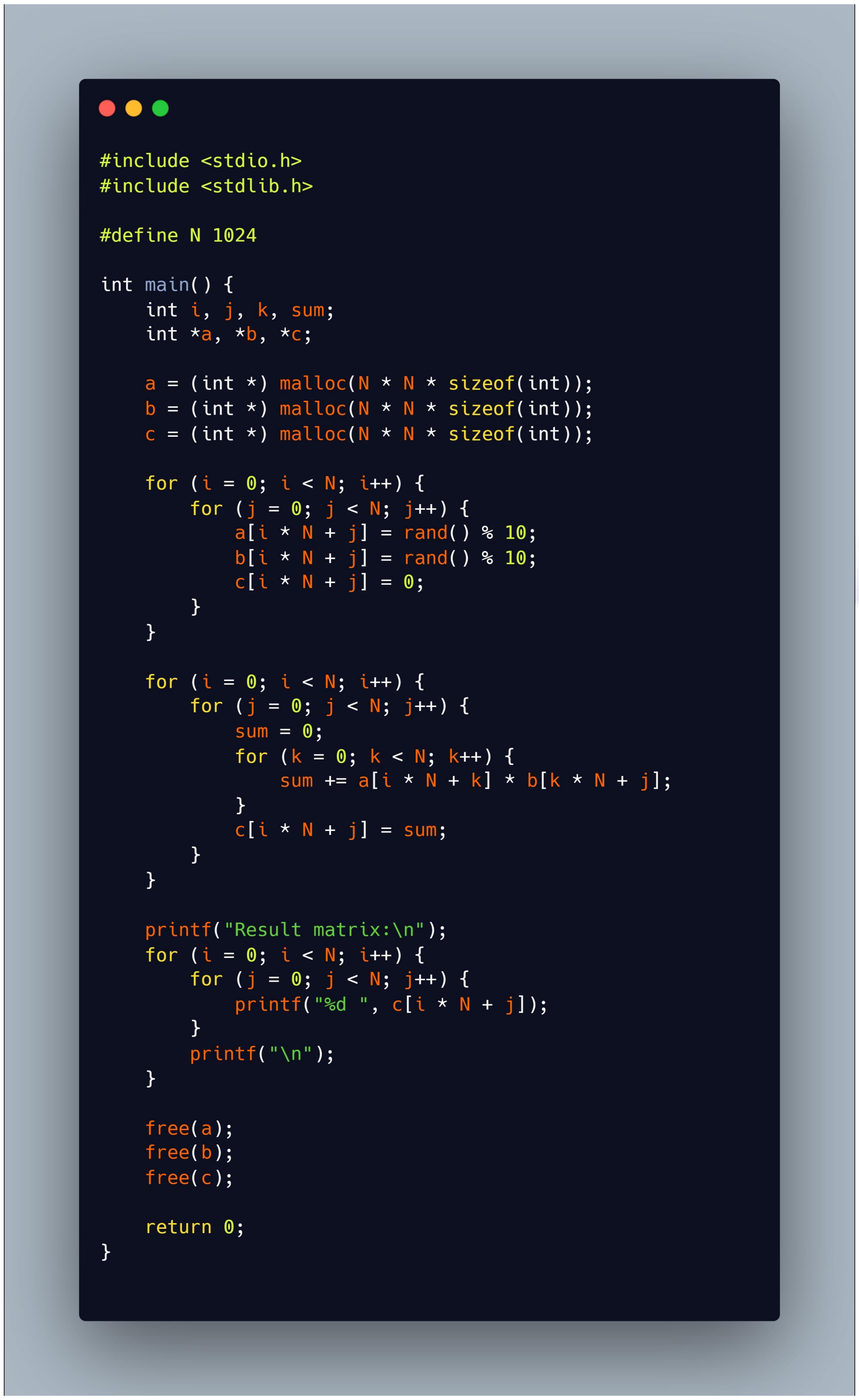
cache-Reference



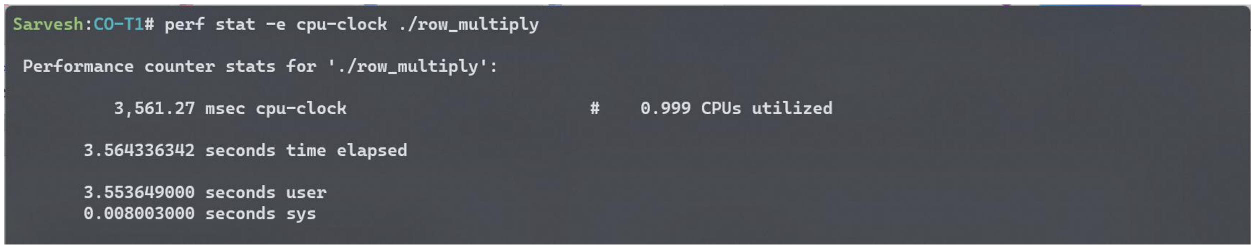
```
Sarvesh:CO-T1# perf stat -e cache-references ./row_multiply

 Performance counter stats for './row_multiply':

    3,02,20,56,846      cache-references

       3.546203100 seconds time elapsed

       3.541898000 seconds user
       0.000000000 seconds sys
```

Title: Write a program to multiply two different matrices of size 1024 x 1024

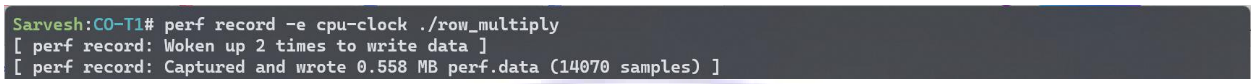
Row-wise Multiply



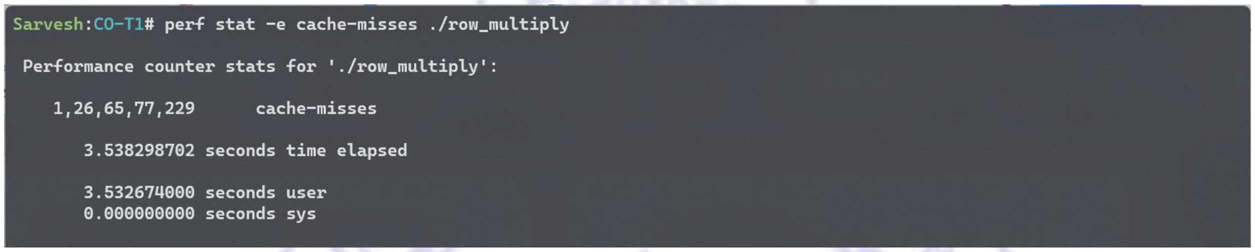
```c
#include <stdio.h>
#include <stdlib.h>

#define N 1024

int main() {
    int i, j, k, sum;
    int *a, *b, *c;

    a = (int *) malloc(N * N * sizeof(int));
    b = (int *) malloc(N * N * sizeof(int));
    c = (int *) malloc(N * N * sizeof(int));

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i * N + j] = rand() % 10;
            b[i * N + j] = rand() % 10;
            c[i * N + j] = 0;
        }
    }

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += a[i * N + k] * b[k * N + j];
            }
            c[i * N + j] = sum;
        }
    }

    printf("Result matrix:\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("%d ", c[i * N + j]);
        }
        printf("\n");
    }

    free(a);
    free(b);
    free(c);

    return 0;
}
```



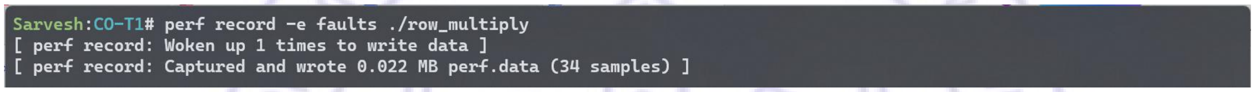
```
Sarvesh:CO-T1# perf stat -e cpu-clock ./row_multiply

 Performance counter stats for './row_multiply':

          3,561.27 msec cpu-clock                        #    0.999 CPUs utilized

       3.564336342 seconds time elapsed

       3.553649000 seconds user
       0.008003000 seconds sys
```



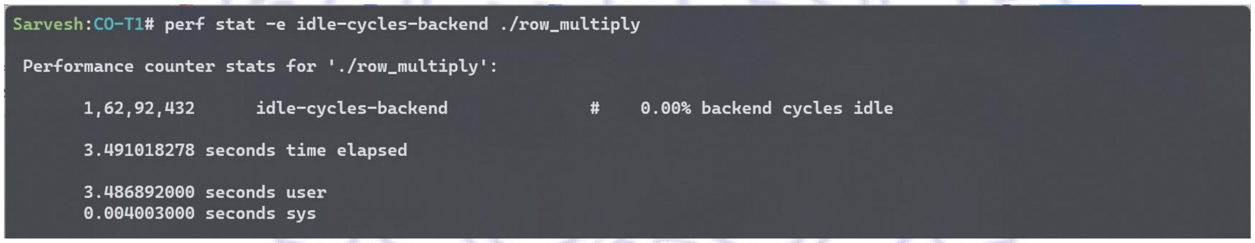
```
Sarvesh:CO-T1# perf record -e cpu-clock ./row_multiply
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 0.558 MB perf.data (14070 samples) ]
```



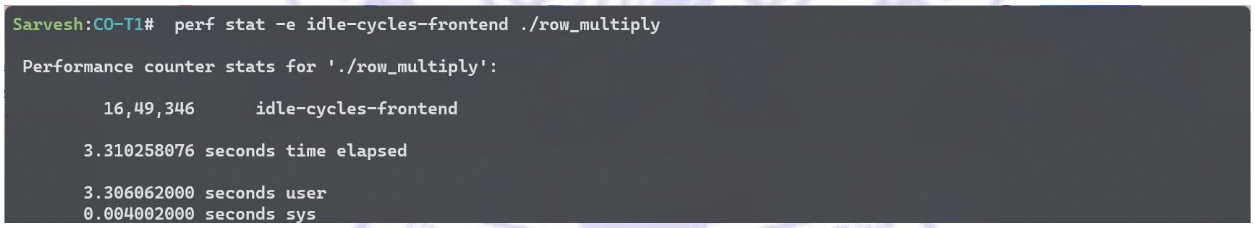
```
Sarvesh:CO-T1# perf stat -e cache-misses ./row_multiply

 Performance counter stats for './row_multiply':

    1,26,65,77,229      cache-misses

       3.538298702 seconds time elapsed

       3.532674000 seconds user
       0.000000000 seconds sys
```



```
Sarvesh:CO-T1# perf record -e faults ./row_multiply
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.022 MB perf.data (34 samples) ]
```

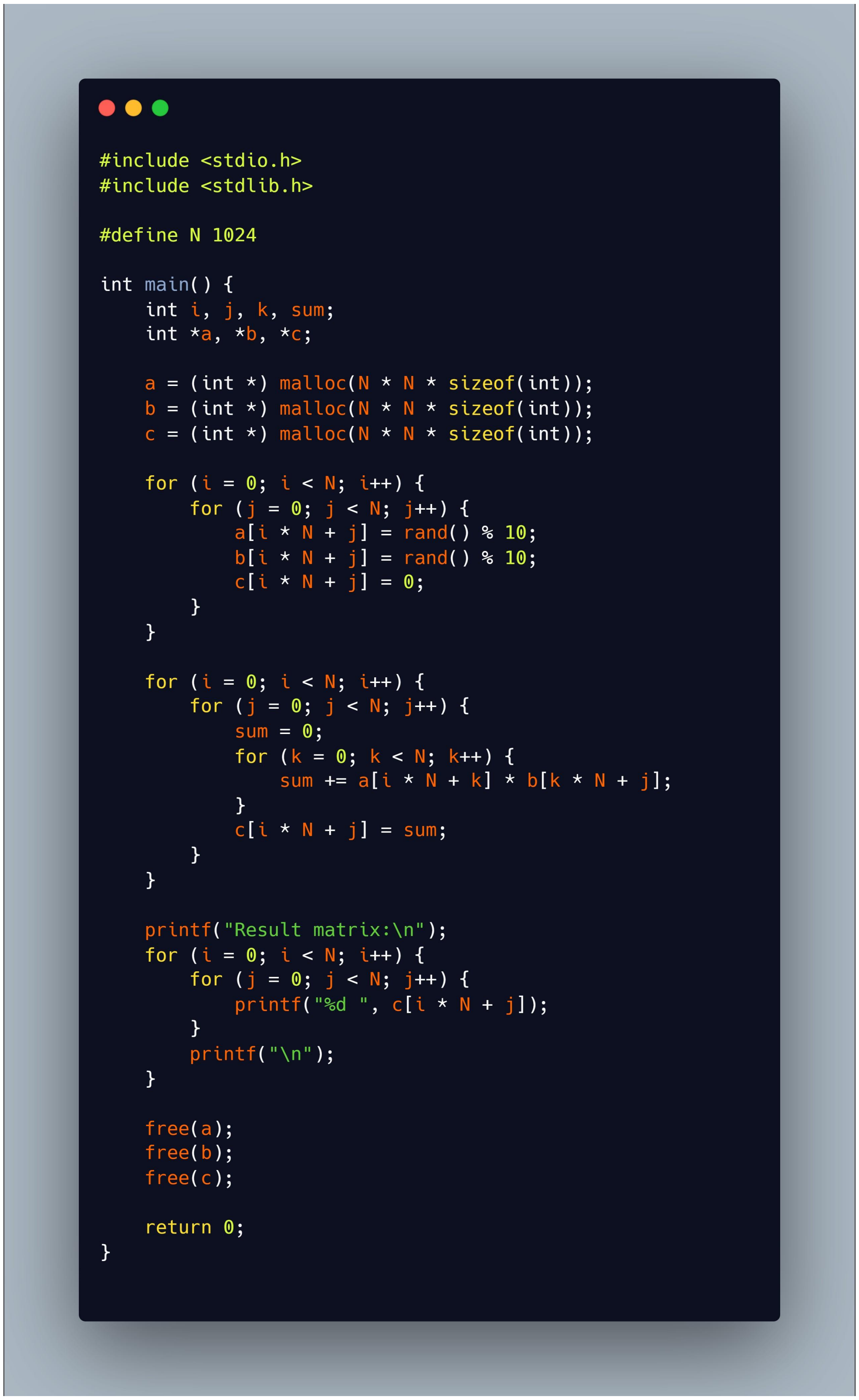


```
Sarvesh:CO-T1# perf stat -e idle-cycles-backend ./row_multiply

 Performance counter stats for './row_multiply':

       1,62,92,432      idle-cycles-backend              #    0.00% backend cycles idle

       3.491018278 seconds time elapsed

       3.486892000 seconds user
       0.004003000 seconds sys
```

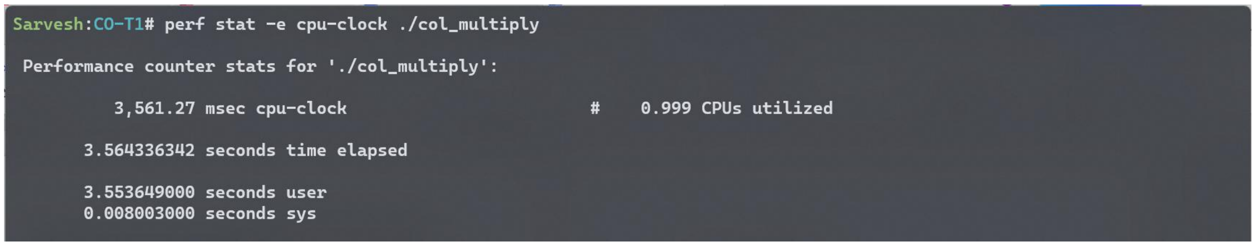


```
Sarvesh:CO-T1#  perf stat -e idle-cycles-frontend ./row_multiply

 Performance counter stats for './row_multiply':

         16,49,346      idle-cycles-frontend

       3.310258076 seconds time elapsed

       3.306062000 seconds user
       0.004002000 seconds sys
```

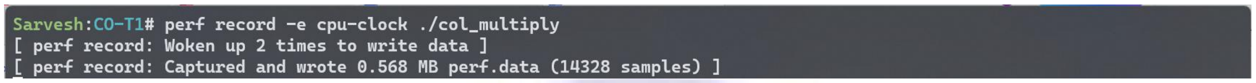
Col-wise Multiply



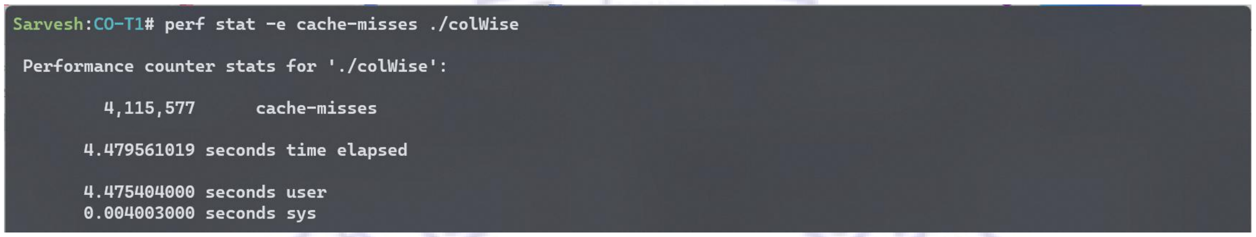
```c
#include <stdio.h>
#include <stdlib.h>

#define N 1024

int main() {
    int i, j, k, sum;
    int *a, *b, *c;

    a = (int *) malloc(N * N * sizeof(int));
    b = (int *) malloc(N * N * sizeof(int));
    c = (int *) malloc(N * N * sizeof(int));

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i * N + j] = rand() % 10;
            b[i * N + j] = rand() % 10;
            c[i * N + j] = 0;
        }
    }

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k = 0; k < N; k++) {
                sum += a[i * N + k] * b[k * N + j];
            }
            c[i * N + j] = sum;
        }
    }

    printf("Result matrix:\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("%d ", c[i * N + j]);
        }
        printf("\n");
    }

    free(a);
    free(b);
    free(c);

    return 0;
}
```



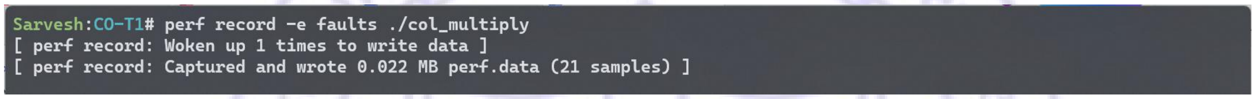
```
Sarvesh:CO-T1# perf stat -e cpu-clock ./col_multiply

 Performance counter stats for './col_multiply':

          3,561.27 msec cpu-clock                        #    0.999 CPUs utilized

       3.564336342 seconds time elapsed

       3.553649000 seconds user
       0.008003000 seconds sys
```



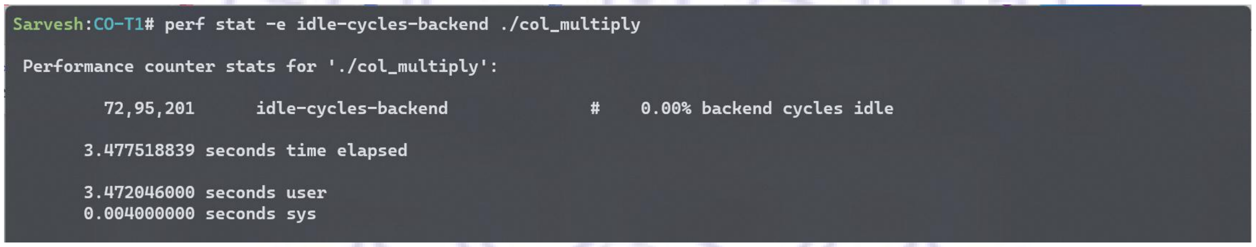
```
Sarvesh:CO-T1# perf record -e cpu-clock ./col_multiply
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 0.568 MB perf.data (14328 samples) ]
```



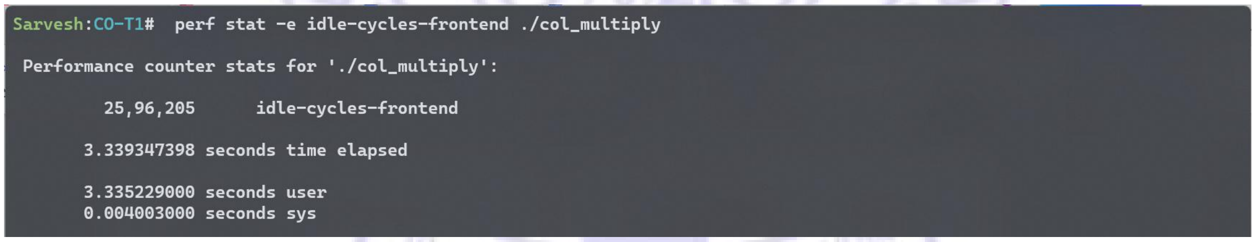
```
Sarvesh:CO-T1# perf stat -e cache-misses ./colWise

 Performance counter stats for './colWise':

         4,115,577      cache-misses

       4.479561019 seconds time elapsed

       4.475404000 seconds user
       0.004003000 seconds sys
```



```
Sarvesh:CO-T1# perf record -e faults ./col_multiply
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.022 MB perf.data (21 samples) ]
```

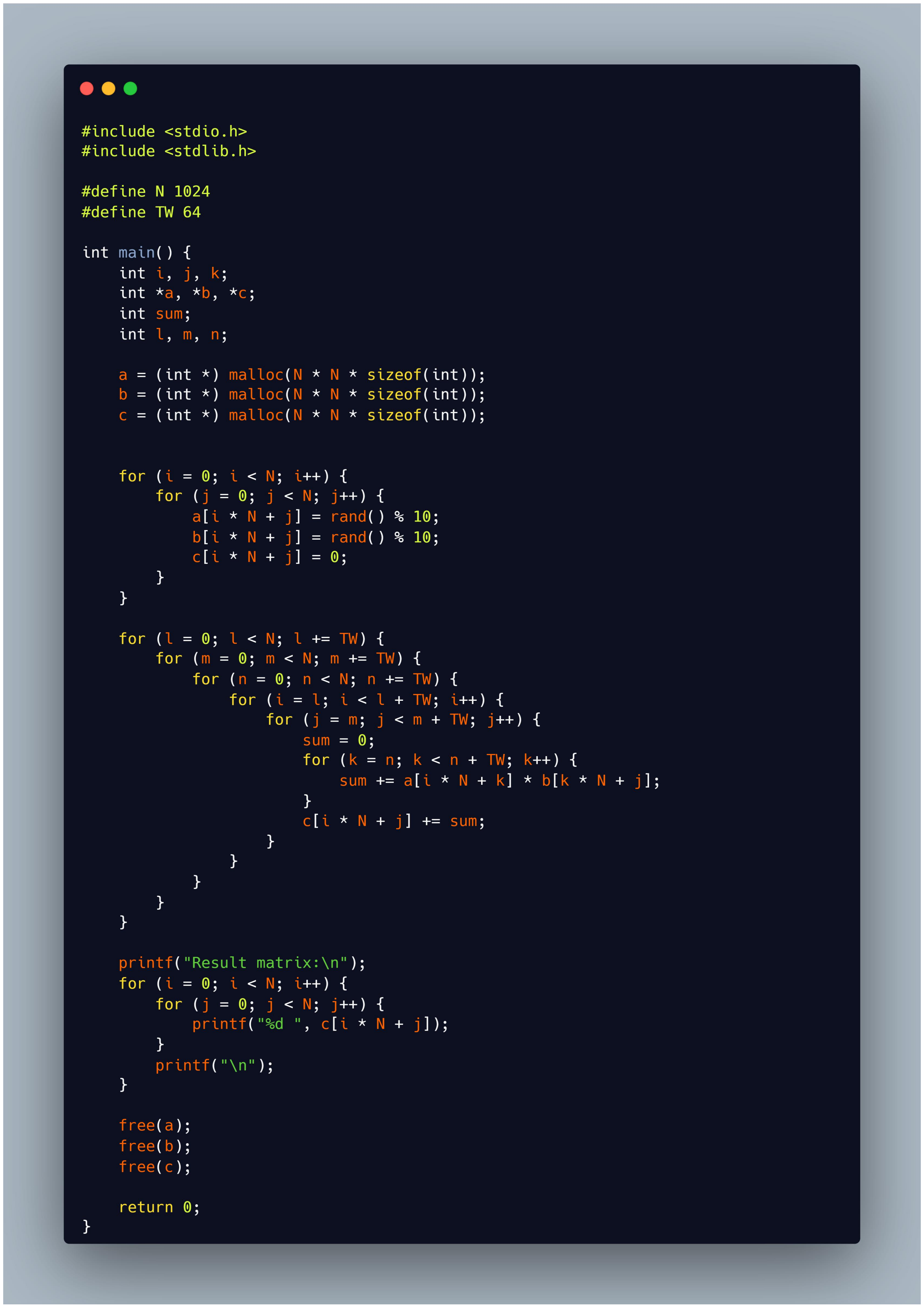


```
Sarvesh:CO-T1# perf stat -e idle-cycles-backend ./col_multiply

 Performance counter stats for './col_multiply':

         72,95,201      idle-cycles-backend              #    0.00% backend cycles idle

       3.477518839 seconds time elapsed

       3.472046000 seconds user
       0.004000000 seconds sys
```

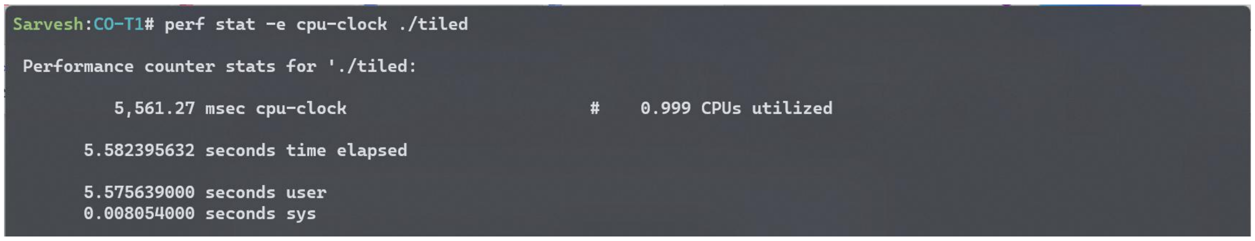


```
Sarvesh:CO-T1#  perf stat -e idle-cycles-frontend ./col_multiply

 Performance counter stats for './col_multiply':

         25,96,205      idle-cycles-frontend

       3.339347398 seconds time elapsed

       3.335229000 seconds user
       0.004003000 seconds sys
```

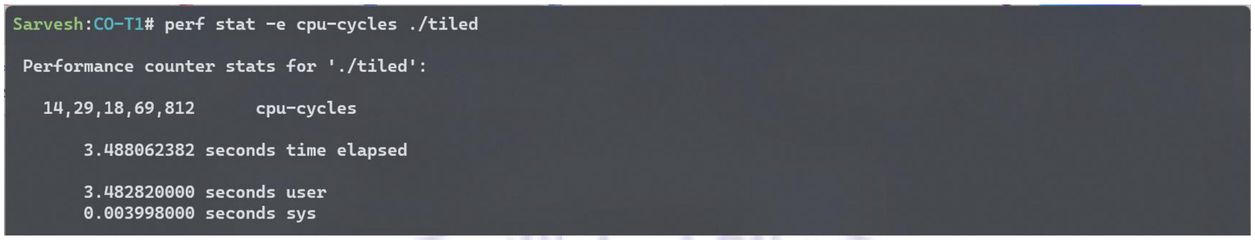
TIled-wise Multiply



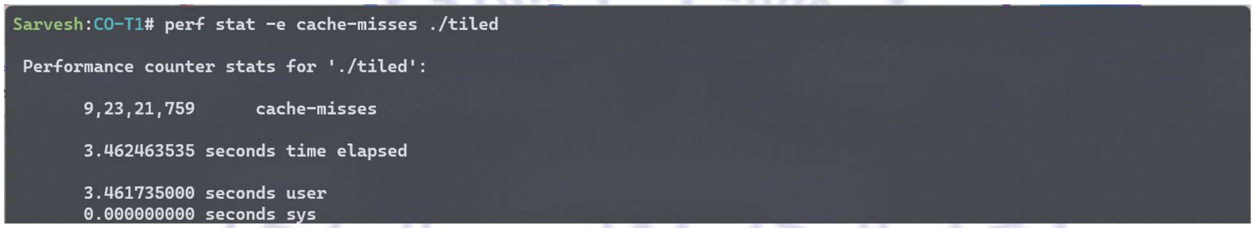
```c
#include <stdio.h>
#include <stdlib.h>

#define N 1024
#define TW 64

int main() {
    int i, j, k;
    int *a, *b, *c;
    int sum;
    int l, m, n;

    a = (int *) malloc(N * N * sizeof(int));
    b = (int *) malloc(N * N * sizeof(int));
    c = (int *) malloc(N * N * sizeof(int));


    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i * N + j] = rand() % 10;
            b[i * N + j] = rand() % 10;
            c[i * N + j] = 0;
        }
    }

    for (l = 0; l < N; l += TW) {
        for (m = 0; m < N; m += TW) {
            for (n = 0; n < N; n += TW) {
                for (i = l; i < l + TW; i++) {
                    for (j = m; j < m + TW; j++) {
                        sum = 0;
                        for (k = n; k < n + TW; k++) {
                            sum += a[i * N + k] * b[k * N + j];
                        }
                        c[i * N + j] += sum;
                    }
                }
            }
        }
    }

    printf("Result matrix:\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("%d ", c[i * N + j]);
        }
        printf("\n");
    }

    free(a);
    free(b);
    free(c);

    return 0;
}
```



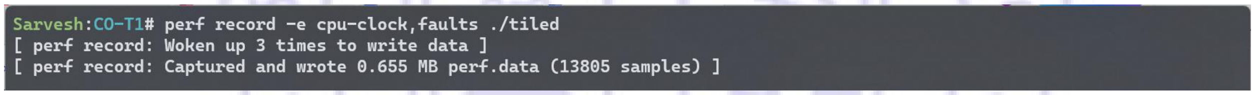
```
Sarvesh:CO-T1# perf stat -e cpu-clock ./tiled

 Performance counter stats for './tiled:

          5,561.27 msec cpu-clock                        #    0.999 CPUs utilized

       5.582395632 seconds time elapsed

       5.575639000 seconds user
       0.008054000 seconds sys
```



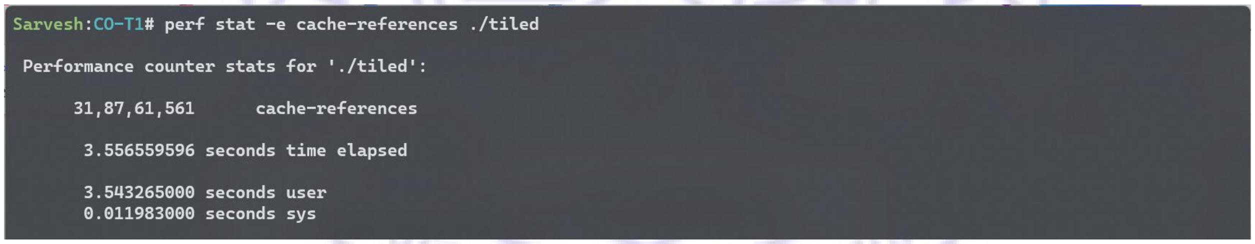
```
Sarvesh:CO-T1# perf stat -e cpu-cycles ./tiled

 Performance counter stats for './tiled':

   14,29,18,69,812      cpu-cycles

       3.488062382 seconds time elapsed

       3.482820000 seconds user
       0.003998000 seconds sys
```



```
Sarvesh:CO-T1# perf stat -e cache-misses ./tiled

 Performance counter stats for './tiled':

       9,23,21,759      cache-misses

       3.462463535 seconds time elapsed

       3.461735000 seconds user
       0.000000000 seconds sys
```



```
Sarvesh:CO-T1# perf record -e cpu-clock,faults ./tiled
[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0.655 MB perf.data (13805 samples) ]
```



```
Sarvesh:CO-T1# perf stat -e cache-references ./tiled

 Performance counter stats for './tiled':

      31,87,61,561      cache-references

       3.556559596 seconds time elapsed

       3.543265000 seconds user
       0.011983000 seconds sys
```