Report on Optimization of

N-Body LJMD Refactoring

Nazarova Natalia [*]

February 2020

Contents

[*]E-mail: nnazarov@sissa.it

1 Optimization of the force computation

The most expensive operation in our code (with C and python interfaces) is the calculation of the force between particles. My part of the work in this exercise is to optimize the calculation of forces.

1.1 Slow down by splitting main source file into multiple files

Here we gonna discuss which slow down (negative speed up) we get by break down the single file `ljmd.c` into multiple files.

It is very clear, that after that our main file will calling for functions and spend cycles on this operations. But we actually studied this numbers by `perf`.

So, by splitting of our main source file we get next **slow down**:

- $\sim$ 15% per C interface task for $N = 108$
- $\sim$ 25 % per C interface task for $N = 2916$
- $\sim$ 30% per Python interface task for $N = 108$
- $\sim$ 46 % per Python interface task for $N = 2916$

1.2 Newton's third law optimization

The first optimization we did was optimization using Newton's third law.

The third law states that all forces between two objects exist in equal magnitude and opposite direction: if one object A exerts a force FA on a second object B, then B simultaneously exerts a force FB on A, and the two forces are equal in magnitude and opposite in direction: FA = FB.[29] The third law means that all forces are interactions between different bodies,[30][31] or different regions within one body, and thus that there is no such thing as a force that is not accompanied by an equal and opposite force.

Its speed up calculation of the force in two times per cycle in loop.

By the using the Newton's third law optimization we get next **speed up** per run time:

- $\sim$ 80% per C interface task for $N = 108$
- $\sim$ 69 % per C interface task for $N = 2916$
- $\sim$ 58% per Python interface task for $N = 108$
- $\sim$ 50 % per Python interface task for $N = 2916$

1.3 Math function optimization

Another expensive part of calculation its calculation of $pow()$, $sqrt()$ and divisions.

As we know, addition (1), subtraction(1), comparison(1), abs(2), multiplication(4) are the "green" basic operations.

Division is more than twice as expensive as multiplication. So, we can replace division by multiplication, we do get a speed-up of more than 2 by this.

$Sqrt()$, $pow()$ is very expensive calculation. For example, power is in 100 times more expensive than an addition.

We replaced pow(sigma,6) by 6 multiplication, what get for us speed-up of more than 5 by this.

Also we did optimization operations in cycle of `velocity.c` file: we replace 2 divisions and 1 multiplications by one multiplication, what get for us speed up more then 16 per one cycle in loop.

By the using math function optimization we get next **speed up** per run time:

- $\sim 13\%$ per C interface task for $N = 108$
- $\sim 10\%$ per C interface task for $N = 2916$
- $\sim 20\%$ per Python interface task for $N = 108$
- $\sim 10\%$ per Python interface task for $N = 2916$

2 Main result

Here table with main numbers, which we obtained by using `perf`. Actual `perf report` with all information about perfomance of each task you can find in section 4.

perf parameters				
file	size	time, s	cycles * 10^9	speed up
main	108	1.216	–	1
C, no optimization	108	1.430	4.102	0.850
C, newton law	108	0.733	2.174	1.659
C, math func	108	0.679	1.927	1.791
Python3, no opt	108	1.737	4.222	0.700
Python3, newton law	108	0.947	2.290	1.284
Python3, math func	108	0.820	1.976	1.483
main	2916	37.801	112.029	1
C, no optimization	2916	50.106	148.480	0.754
C, newton law	2916	26.274	77.922	1.439
C, math func	2916	24.555	70.937	1.539
Python3, no opt	2916	70.105	172.011	0.539
Python3, newton law	2916	34.753	84.898	1.088
Python3, math func	2916	32.192	78.719	1.174

3 Summary

As we can see from the plots, run time of task increased, when we split the task into files, because we we spend time for function calls. Then using knowledge of Third Newton's law and avoiding expensive mathematical functions (such as $pow(), sqrt(), division$) reduces the calculation time, but not so much, since the run time was initially very small.

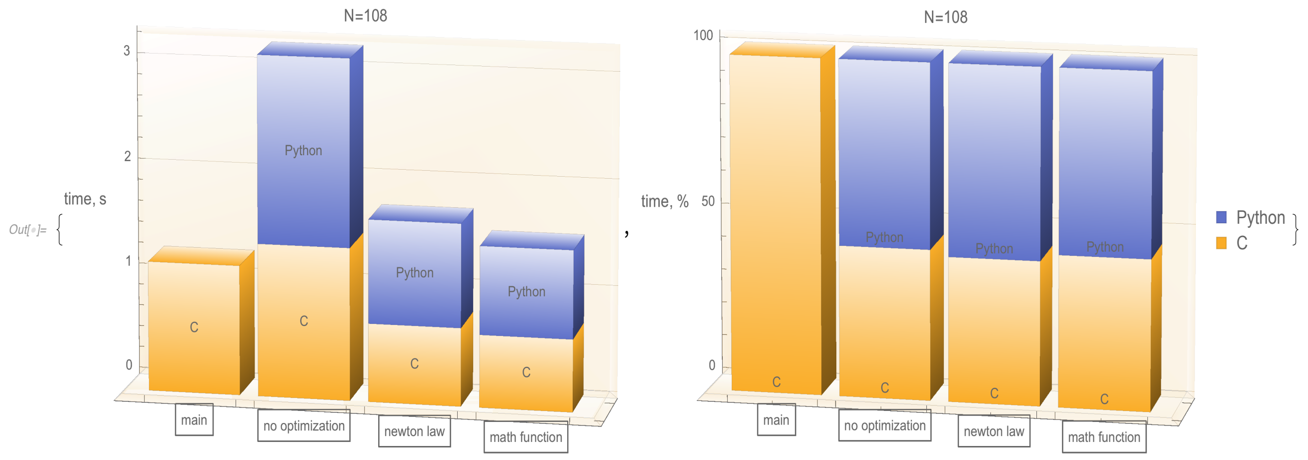


Figure 1: The histogram data shows: (on the left) the absolute and (on the right) relative time of the task with the number of particles $N = 108$. **The first column** of the histogram shows the execution time of the task for the initial file, not split into multiple files. **The second column** of the task, which include multiple files (force compute, verlet time integration, input, output, utilities, header for data structures, prototypes), but without optimization. **The third column** shows the runtime obtained using knowledge of Newton's third law. **The fourth column** shows the runtime obtained by avoiding expensive mathematical functions such as $pow(), sqrt()$, division. Blue column shows run time with python interface, the orange column shows the run time of code, which performed only in C. Data obtained on Ulisse and with the using the `perf`.

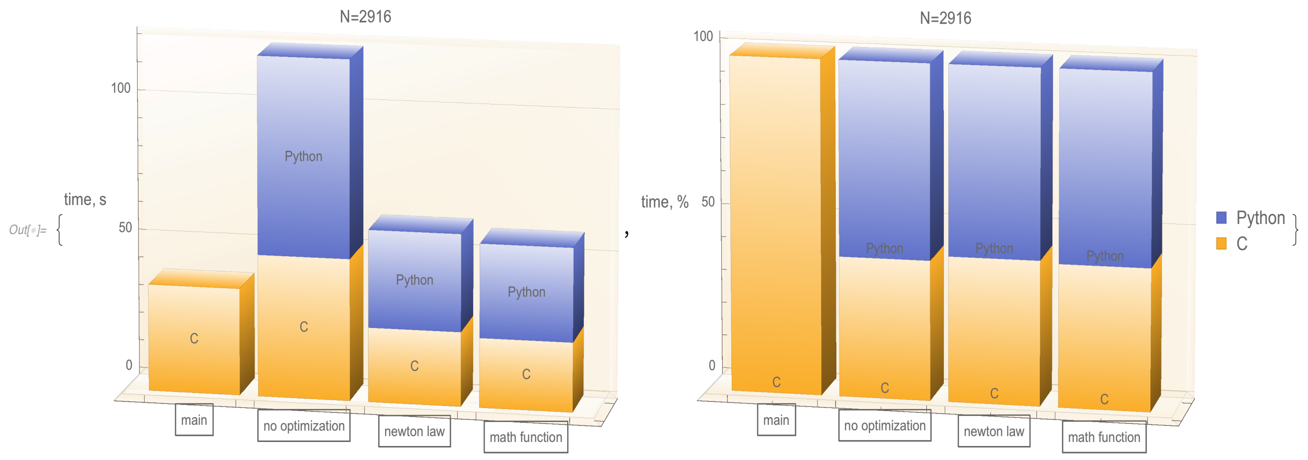
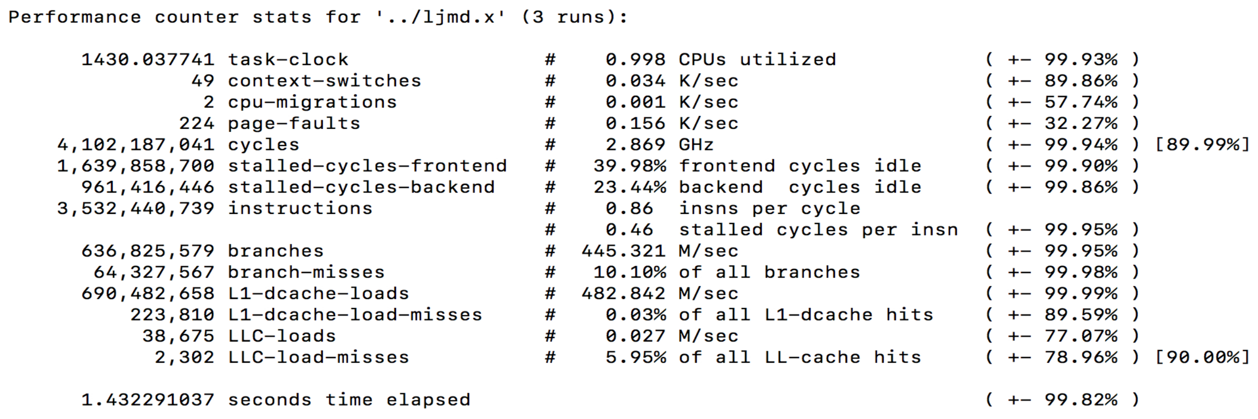


Figure 2: The histogram data shows: (on the left) the absolute and (on the right) relative time of the task with the number of particles $N = 2916$. **The first column** of the histogram shows the execution time of the task for the initial file, not split into multiple files. **The second column** of the task, which include multiple files (force compute, verlet time integration, input, output, utilities, header for data structures, prototypes), but without optimization. **The third column** shows the runtime obtained using knowledge of Newton's third law. **The fourth column** shows the runtime obtained by avoiding expensive mathematical functions such as $pow(), sqrt()$, division. Blue column shows run time with python interface, the orange column shows the run time of code, which performed only in C. Data obtained on Ulisse and with the using the `perf`.

We also see that the runtime using the python interface is longer than using only the C interface, but this excess does not exceed 10%. Another important note is that this excess of relative run time of task with python interface is very stable.

4 Perf data

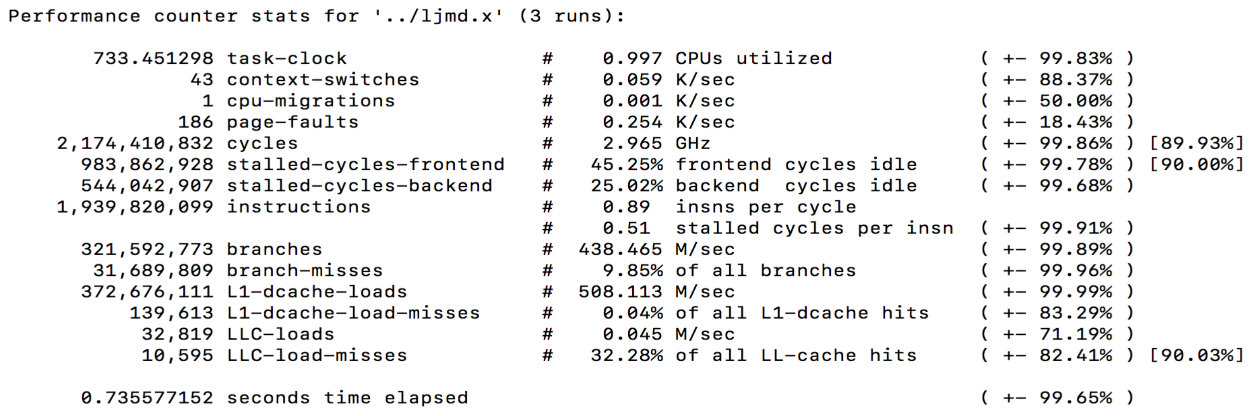


```
 Performance counter stats for '../ljmd.x' (3 runs):

       1430.037741 task-clock                #    0.998 CPUs utilized            ( +- 99.93% )
                49 context-switches          #    0.034 K/sec                    ( +- 89.86% )
                 2 cpu-migrations            #    0.001 K/sec                    ( +- 57.74% )
               224 page-faults               #    0.156 K/sec                    ( +- 32.27% )
     4,102,187,041 cycles                    #    2.869 GHz                      ( +- 99.94% ) [89.99%]
     1,639,858,700 stalled-cycles-frontend   #   39.98% frontend cycles idle     ( +- 99.90% )
       961,416,446 stalled-cycles-backend    #   23.44% backend  cycles idle     ( +- 99.86% )
     3,532,440,739 instructions              #    0.86  insns per cycle
                                             #    0.46  stalled cycles per insn  ( +- 99.95% )
       636,825,579 branches                  #  445.321 M/sec                    ( +- 99.95% )
        64,327,567 branch-misses             #   10.10% of all branches          ( +- 99.98% )
       690,482,658 L1-dcache-loads           #  482.842 M/sec                    ( +- 99.99% )
           223,810 L1-dcache-load-misses     #    0.03% of all L1-dcache hits    ( +- 89.59% )
            38,675 LLC-loads                 #    0.027 M/sec                    ( +- 77.07% )
             2,302 LLC-load-misses           #    5.95% of all LL-cache hits     ( +- 78.96% ) [90.00%]

       1.432291037 seconds time elapsed                                          ( +- 99.82% )
```

Figure 3: Data obtained on Ulisse and with the using the perf for $N = 108$ for task only with C interface without optimization.

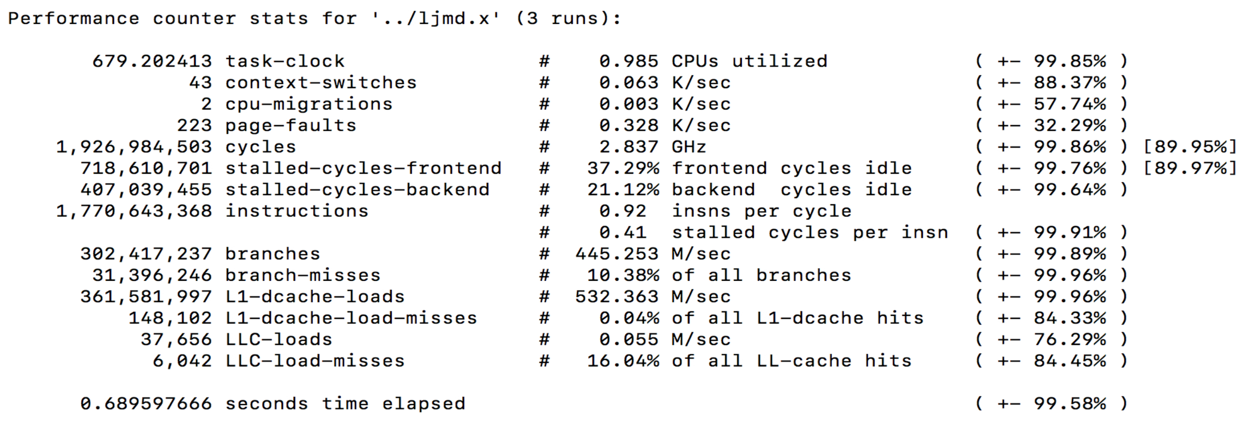


```
 Performance counter stats for '../ljmd.x' (3 runs):

        733.451298 task-clock                #    0.997 CPUs utilized            ( +- 99.83% )
                43 context-switches          #    0.059 K/sec                    ( +- 88.37% )
                 1 cpu-migrations            #    0.001 K/sec                    ( +- 50.00% )
               186 page-faults               #    0.254 K/sec                    ( +- 18.43% )
     2,174,410,832 cycles                    #    2.965 GHz                      ( +- 99.86% ) [89.93%]
       983,862,928 stalled-cycles-frontend   #   45.25% frontend cycles idle     ( +- 99.78% ) [90.00%]
       544,042,907 stalled-cycles-backend    #   25.02% backend  cycles idle     ( +- 99.68% )
     1,939,820,099 instructions              #    0.89  insns per cycle
                                             #    0.51  stalled cycles per insn  ( +- 99.91% )
       321,592,773 branches                  #  438.465 M/sec                    ( +- 99.89% )
        31,689,809 branch-misses             #    9.85% of all branches          ( +- 99.96% )
       372,676,111 L1-dcache-loads           #  508.113 M/sec                    ( +- 99.99% )
           139,613 L1-dcache-load-misses     #    0.04% of all L1-dcache hits    ( +- 83.29% )
            32,819 LLC-loads                 #    0.045 M/sec                    ( +- 71.19% )
            10,595 LLC-load-misses           #   32.28% of all LL-cache hits     ( +- 82.41% ) [90.03%]

       0.735577152 seconds time elapsed                                          ( +- 99.65% )
```

Figure 4: Data obtained on Ulisse and with the using the perf for $N = 108$ for task only with C interface with using knowledge of Third Newton's law.

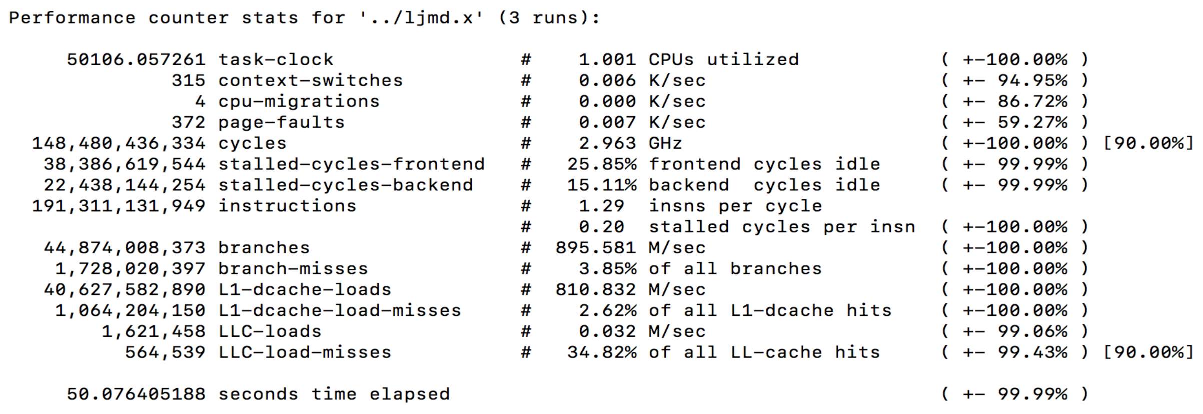


```
 Performance counter stats for '../ljmd.x' (3 runs):

        679.202413 task-clock                #    0.985 CPUs utilized            ( +- 99.85% )
                43 context-switches          #    0.063 K/sec                    ( +- 88.37% )
                 2 cpu-migrations            #    0.003 K/sec                    ( +- 57.74% )
               223 page-faults               #    0.328 K/sec                    ( +- 32.29% )
     1,926,984,503 cycles                    #    2.837 GHz                      ( +- 99.86% ) [89.95%]
       718,610,701 stalled-cycles-frontend   #   37.29% frontend cycles idle     ( +- 99.76% ) [89.97%]
       407,039,455 stalled-cycles-backend    #   21.12% backend  cycles idle     ( +- 99.64% )
     1,770,643,368 instructions              #    0.92  insns per cycle
                                             #    0.41  stalled cycles per insn  ( +- 99.91% )
       302,417,237 branches                  #  445.253 M/sec                    ( +- 99.89% )
        31,396,246 branch-misses             #   10.38% of all branches          ( +- 99.96% )
       361,581,997 L1-dcache-loads           #  532.363 M/sec                    ( +- 99.96% )
           148,102 L1-dcache-load-misses     #    0.04% of all L1-dcache hits    ( +- 84.33% )
            37,656 LLC-loads                 #    0.055 M/sec                    ( +- 76.29% )
             6,042 LLC-load-misses           #   16.04% of all LL-cache hits     ( +- 84.45% )

       0.689597666 seconds time elapsed                                          ( +- 99.58% )
```

Figure 5: Data obtained on Ulisse and with the using the perf for $N = 108$ for task only with C interface with avoiding expensive mathematical functions.

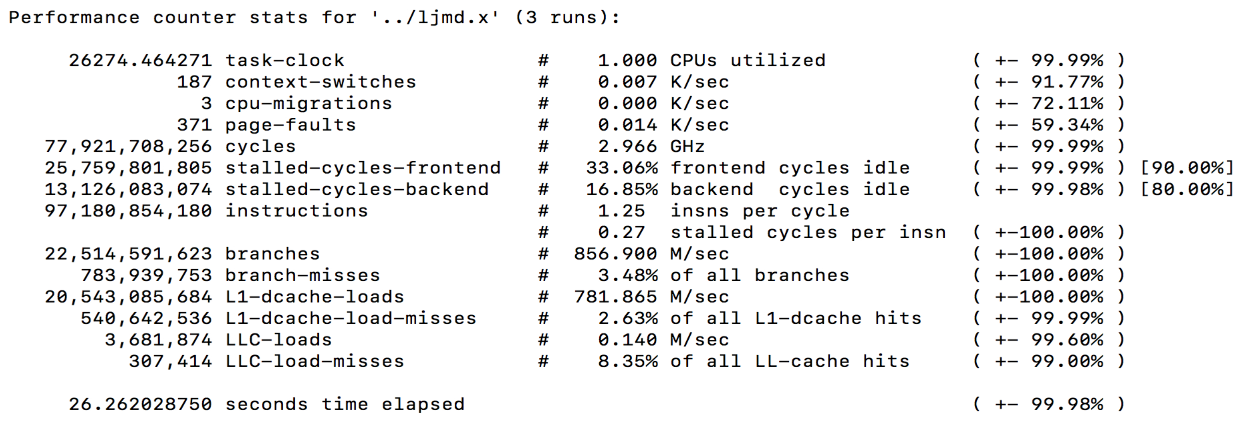


```
 Performance counter stats for '../ljmd.x' (3 runs):

      50106.057261 task-clock                #    1.001 CPUs utilized            ( +-100.00% )
               315 context-switches          #    0.006 K/sec                    ( +- 94.95% )
                 4 cpu-migrations            #    0.000 K/sec                    ( +- 86.72% )
               372 page-faults               #    0.007 K/sec                    ( +- 59.27% )
   148,480,436,334 cycles                    #    2.963 GHz                      ( +-100.00% ) [90.00%]
    38,386,619,544 stalled-cycles-frontend   #   25.85% frontend cycles idle     ( +- 99.99% )
    22,438,144,254 stalled-cycles-backend    #   15.11% backend  cycles idle     ( +- 99.99% )
   191,311,131,949 instructions              #    1.29  insns per cycle
                                             #    0.20  stalled cycles per insn  ( +-100.00% )
    44,874,008,373 branches                  #  895.581 M/sec                    ( +-100.00% )
     1,728,020,397 branch-misses             #    3.85% of all branches          ( +-100.00% )
    40,627,582,890 L1-dcache-loads           #  810.832 M/sec                    ( +-100.00% )
     1,064,204,150 L1-dcache-load-misses     #    2.62% of all L1-dcache hits    ( +-100.00% )
         1,621,458 LLC-loads                 #    0.032 M/sec                    ( +- 99.06% )
           564,539 LLC-load-misses           #   34.82% of all LL-cache hits     ( +- 99.43% ) [90.00%]

      50.076405188 seconds time elapsed                                          ( +- 99.99% )
```

Figure 6: Data obtained on Ulisse and with the using the perf for $N = 2916$ for task only with C interface without optimization.

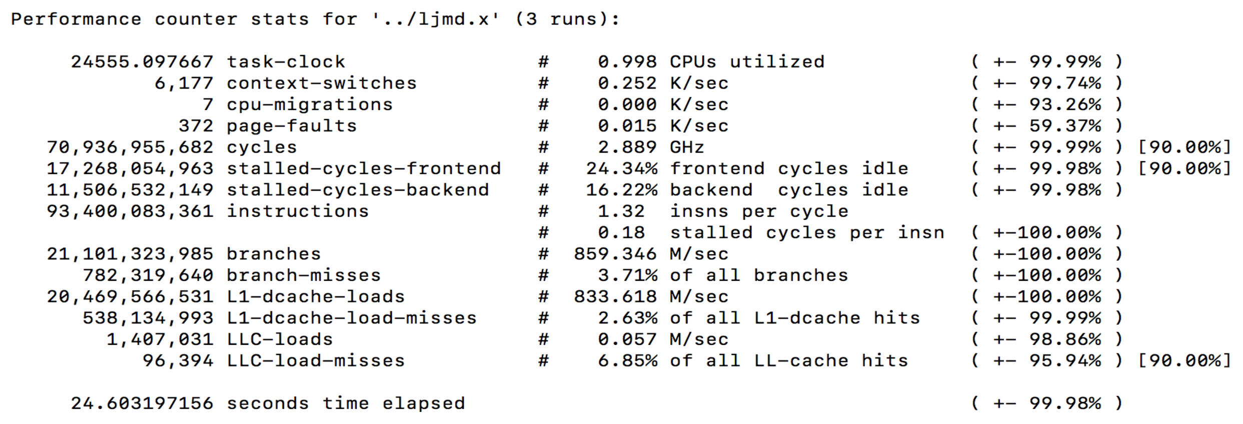


```
 Performance counter stats for '../ljmd.x' (3 runs):

      26274.464271 task-clock                #    1.000 CPUs utilized            ( +- 99.99% )
               187 context-switches          #    0.007 K/sec                    ( +- 91.77% )
                 3 cpu-migrations            #    0.000 K/sec                    ( +- 72.11% )
               371 page-faults               #    0.014 K/sec                    ( +- 59.34% )
    77,921,708,256 cycles                    #    2.966 GHz                      ( +- 99.99% )
    25,759,801,805 stalled-cycles-frontend   #   33.06% frontend cycles idle     ( +- 99.99% ) [90.00%]
    13,126,083,074 stalled-cycles-backend    #   16.85% backend  cycles idle     ( +- 99.98% ) [80.00%]
    97,180,854,180 instructions              #    1.25  insns per cycle
                                             #    0.27  stalled cycles per insn  ( +-100.00% )
    22,514,591,623 branches                  #  856.900 M/sec                    ( +-100.00% )
       783,939,753 branch-misses             #    3.48% of all branches          ( +-100.00% )
    20,543,085,684 L1-dcache-loads           #  781.865 M/sec                    ( +-100.00% )
       540,642,536 L1-dcache-load-misses     #    2.63% of all L1-dcache hits    ( +- 99.99% )
         3,681,874 LLC-loads                 #    0.140 M/sec                    ( +- 99.60% )
           307,414 LLC-load-misses           #    8.35% of all LL-cache hits     ( +- 99.00% )

      26.262028750 seconds time elapsed                                          ( +- 99.98% )
```

Figure 7: Data obtained on Ulisse and with the using the perf for $N = 2916$ for task only with C interface with using knowledge of Third Newton's law.

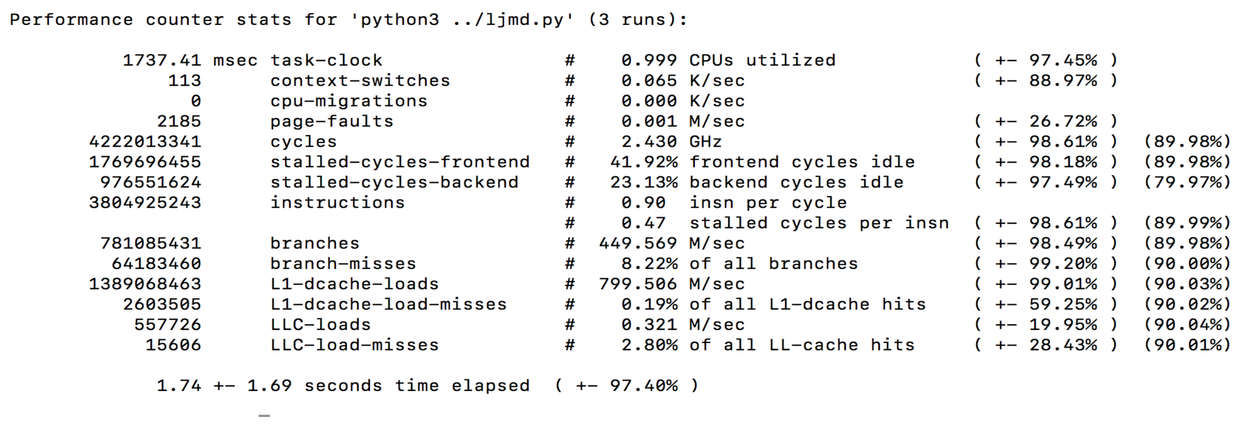


```
 Performance counter stats for '../ljmd.x' (3 runs):

      24555.097667 task-clock                #    0.998 CPUs utilized          ( +- 99.99% )
             6,177 context-switches          #    0.252 K/sec                  ( +- 99.74% )
                 7 cpu-migrations            #    0.000 K/sec                  ( +- 93.26% )
               372 page-faults               #    0.015 K/sec                  ( +- 59.37% )
    70,936,955,682 cycles                    #    2.889 GHz                    ( +- 99.99% ) [90.00%]
    17,268,054,963 stalled-cycles-frontend   #   24.34% frontend cycles idle   ( +- 99.98% ) [90.00%]
    11,506,532,149 stalled-cycles-backend    #   16.22% backend  cycles idle   ( +- 99.98% )
    93,400,083,361 instructions              #    1.32  insns per cycle
                                             #    0.18  stalled cycles per insn  ( +-100.00% )
    21,101,323,985 branches                  #  859.346 M/sec                  ( +-100.00% )
       782,319,640 branch-misses             #    3.71% of all branches        ( +-100.00% )
    20,469,566,531 L1-dcache-loads           #  833.618 M/sec                  ( +-100.00% )
       538,134,993 L1-dcache-load-misses     #    2.63% of all L1-dcache hits  ( +- 99.99% )
         1,407,031 LLC-loads                 #    0.057 M/sec                  ( +- 98.86% )
            96,394 LLC-load-misses           #    6.85% of all LL-cache hits   ( +- 95.94% ) [90.00%]

      24.603197156 seconds time elapsed                                        ( +- 99.98% )
```

Figure 8: Data obtained on Ulisse and with the using the **perf** for $N = 2916$ for task only with C interface with avoiding expensive mathematical functions.

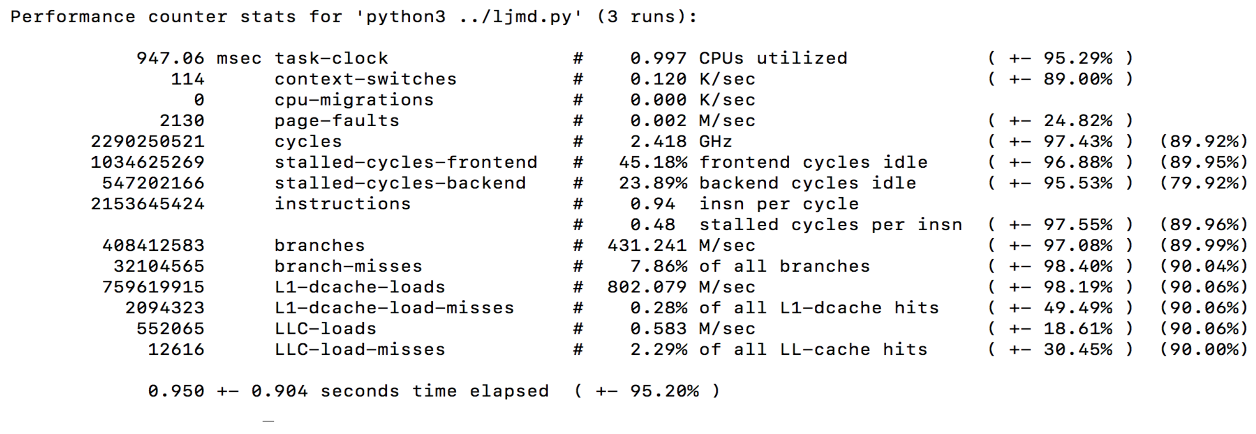


```
 Performance counter stats for 'python3 ../ljmd.py' (3 runs):

          1737.41 msec task-clock                #    0.999 CPUs utilized          ( +- 97.45% )
              113      context-switches          #    0.065 K/sec                  ( +- 88.97% )
                0      cpu-migrations            #    0.000 K/sec
             2185      page-faults               #    0.001 M/sec                  ( +- 26.72% )
       4222013341      cycles                    #    2.430 GHz                    ( +- 98.61% )  (89.98%)
       1769696455      stalled-cycles-frontend   #   41.92% frontend cycles idle   ( +- 98.18% )  (89.98%)
        976551624      stalled-cycles-backend    #   23.13% backend cycles idle    ( +- 97.49% )  (79.97%)
       3804925243      instructions              #    0.90  insn per cycle
                                                 #    0.47  stalled cycles per insn  ( +- 98.61% )  (89.99%)
        781085431      branches                  #  449.569 M/sec                  ( +- 98.49% )  (89.98%)
         64183460      branch-misses             #    8.22% of all branches        ( +- 99.20% )  (90.00%)
       1389068463      L1-dcache-loads           #  799.506 M/sec                  ( +- 99.01% )  (90.03%)
          2603505      L1-dcache-load-misses     #    0.19% of all L1-dcache hits  ( +- 59.25% )  (90.02%)
           557726      LLC-loads                 #    0.321 M/sec                  ( +- 19.95% )  (90.04%)
            15606      LLC-load-misses           #    2.80% of all LL-cache hits   ( +- 28.43% )  (90.01%)

             1.74 +- 1.69 seconds time elapsed  ( +- 97.40% )
```

Figure 9: Data obtained on Ulisse and with the using the **perf** for $N = 108$ for task with python interface without optimization.

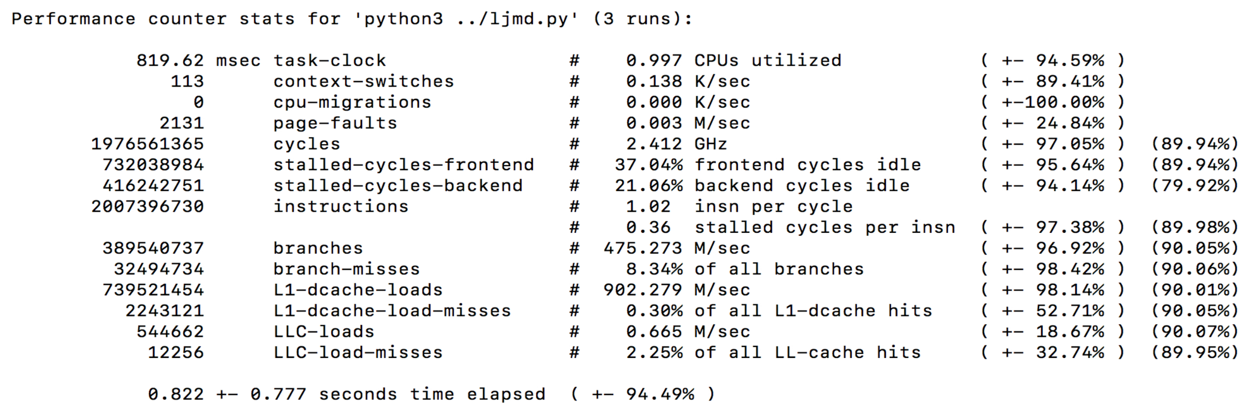


```
 Performance counter stats for 'python3 ../ljmd.py' (3 runs):

           947.06 msec task-clock                #    0.997 CPUs utilized          ( +- 95.29% )
              114      context-switches          #    0.120 K/sec                  ( +- 89.00% )
                0      cpu-migrations            #    0.000 K/sec
             2130      page-faults               #    0.002 M/sec                  ( +- 24.82% )
       2290250521      cycles                    #    2.418 GHz                    ( +- 97.43% )  (89.92%)
       1034625269      stalled-cycles-frontend   #   45.18% frontend cycles idle   ( +- 96.88% )  (89.95%)
        547202166      stalled-cycles-backend    #   23.89% backend cycles idle    ( +- 95.53% )  (79.92%)
       2153645424      instructions              #    0.94  insn per cycle
                                                 #    0.48  stalled cycles per insn  ( +- 97.55% )  (89.96%)
        408412583      branches                  #  431.241 M/sec                  ( +- 97.08% )  (89.99%)
         32104565      branch-misses             #    7.86% of all branches        ( +- 98.40% )  (90.04%)
        759619915      L1-dcache-loads           #  802.079 M/sec                  ( +- 98.19% )  (90.06%)
          2094323      L1-dcache-load-misses     #    0.28% of all L1-dcache hits  ( +- 49.49% )  (90.06%)
           552065      LLC-loads                 #    0.583 M/sec                  ( +- 18.61% )  (90.06%)
            12616      LLC-load-misses           #    2.29% of all LL-cache hits   ( +- 30.45% )  (90.00%)

            0.950 +- 0.904 seconds time elapsed  ( +- 95.20% )
```

Figure 10: Data obtained on Ulisse and with the using the **perf** for $N = 108$ for task with python interface with using knowledge of Third Newton's law.

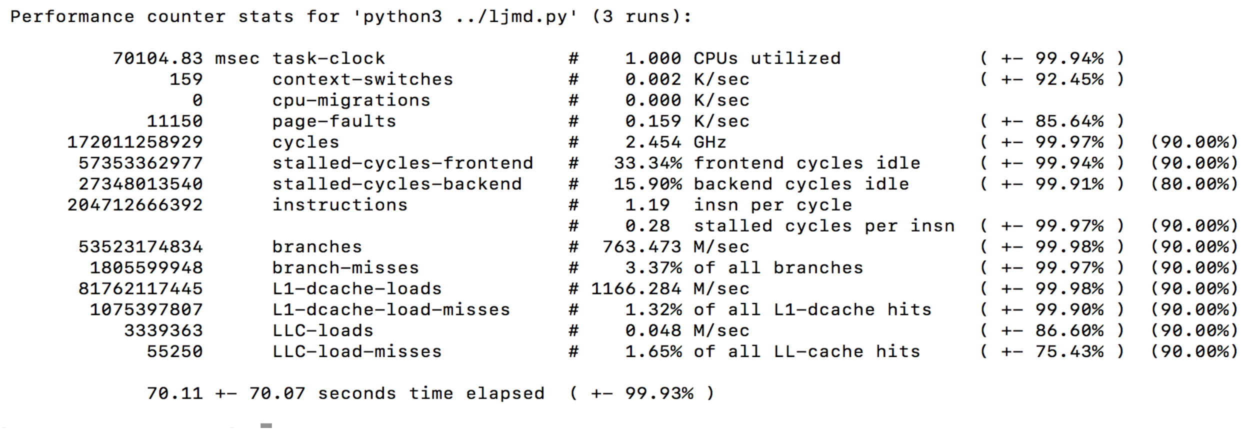


```
 Performance counter stats for 'python3 ../ljmd.py' (3 runs):

            819.62 msec task-clock                #    0.997 CPUs utilized            ( +- 94.59% )
               113      context-switches          #    0.138 K/sec                    ( +- 89.41% )
                 0      cpu-migrations            #    0.000 K/sec                    ( +-100.00% )
              2131      page-faults               #    0.003 M/sec                    ( +- 24.84% )
        1976561365      cycles                    #    2.412 GHz                      ( +- 97.05% )  (89.94%)
         732038984      stalled-cycles-frontend   #   37.04% frontend cycles idle     ( +- 95.64% )  (89.94%)
         416242751      stalled-cycles-backend    #   21.06% backend cycles idle      ( +- 94.14% )  (79.92%)
        2007396730      instructions              #    1.02  insn per cycle
                                                  #    0.36  stalled cycles per insn  ( +- 97.38% )  (89.98%)
         389540737      branches                  #  475.273 M/sec                    ( +- 96.92% )  (90.05%)
          32494734      branch-misses             #    8.34% of all branches          ( +- 98.42% )  (90.06%)
         739521454      L1-dcache-loads           #  902.279 M/sec                    ( +- 98.14% )  (90.01%)
           2243121      L1-dcache-load-misses     #    0.30% of all L1-dcache hits    ( +- 52.71% )  (90.05%)
            544662      LLC-loads                 #    0.665 M/sec                    ( +- 18.67% )  (90.07%)
             12256      LLC-load-misses           #    2.25% of all LL-cache hits     ( +- 32.74% )  (89.95%)

             0.822 +- 0.777 seconds time elapsed  ( +- 94.49% )
```

Figure 11: Data obtained on Ulisse and with the using the `perf` for $N = 108$ for task with python interface with avoiding expensive mathematical functions.

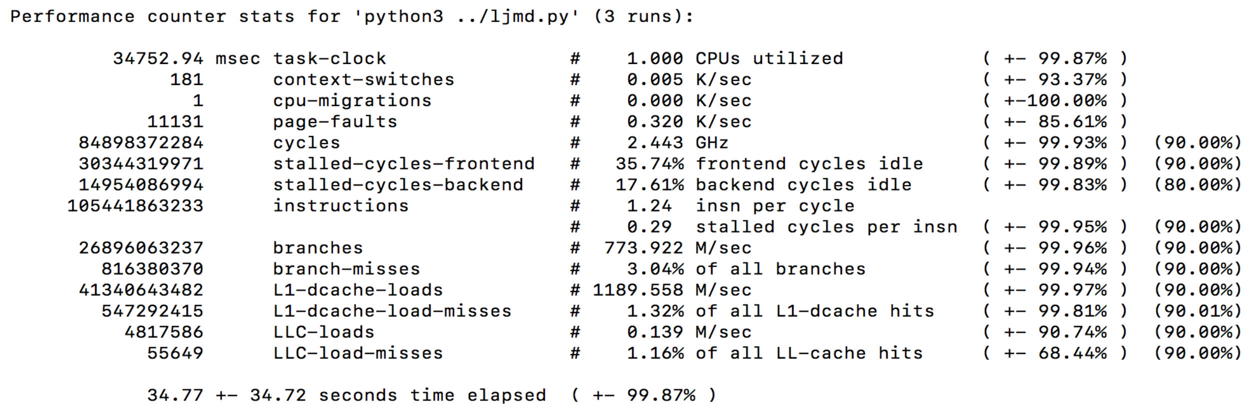


```
 Performance counter stats for 'python3 ../ljmd.py' (3 runs):

          70104.83 msec task-clock                #    1.000 CPUs utilized            ( +- 99.94% )
               159      context-switches          #    0.002 K/sec                    ( +- 92.45% )
                 0      cpu-migrations            #    0.000 K/sec
             11150      page-faults               #    0.159 K/sec                    ( +- 85.64% )
      172011258929      cycles                    #    2.454 GHz                      ( +- 99.97% )  (90.00%)
       57353362977      stalled-cycles-frontend   #   33.34% frontend cycles idle     ( +- 99.94% )  (90.00%)
       27348013540      stalled-cycles-backend    #   15.90% backend cycles idle      ( +- 99.91% )  (80.00%)
      204712666392      instructions              #    1.19  insn per cycle
                                                  #    0.28  stalled cycles per insn  ( +- 99.97% )  (90.00%)
       53523174834      branches                  #  763.473 M/sec                    ( +- 99.98% )  (90.00%)
        1805599948      branch-misses             #    3.37% of all branches          ( +- 99.97% )  (90.00%)
       81762117445      L1-dcache-loads           # 1166.284 M/sec                    ( +- 99.98% )  (90.00%)
        1075397807      L1-dcache-load-misses     #    1.32% of all L1-dcache hits    ( +- 99.90% )  (90.00%)
           3339363      LLC-loads                 #    0.048 M/sec                    ( +- 86.60% )  (90.00%)
             55250      LLC-load-misses           #    1.65% of all LL-cache hits     ( +- 75.43% )  (90.00%)

             70.11 +- 70.07 seconds time elapsed  ( +- 99.93% )
```

Figure 12: Data obtained on Ulisse and with the using the `perf` for $N = 2916$ for task with python interface without optimization.



```
 Performance counter stats for 'python3 ../ljmd.py' (3 runs):

          34752.94 msec task-clock                #    1.000 CPUs utilized            ( +- 99.87% )
               181      context-switches          #    0.005 K/sec                    ( +- 93.37% )
                 1      cpu-migrations            #    0.000 K/sec                    ( +-100.00% )
             11131      page-faults               #    0.320 K/sec                    ( +- 85.61% )
       84898372284      cycles                    #    2.443 GHz                      ( +- 99.93% )  (90.00%)
       30344319971      stalled-cycles-frontend   #   35.74% frontend cycles idle     ( +- 99.89% )  (90.00%)
       14954086994      stalled-cycles-backend    #   17.61% backend cycles idle      ( +- 99.83% )  (80.00%)
      105441863233      instructions              #    1.24  insn per cycle
                                                  #    0.29  stalled cycles per insn  ( +- 99.95% )  (90.00%)
       26896063237      branches                  #  773.922 M/sec                    ( +- 99.96% )  (90.00%)
         816380370      branch-misses             #    3.04% of all branches          ( +- 99.94% )  (90.00%)
       41340643482      L1-dcache-loads           # 1189.558 M/sec                    ( +- 99.97% )  (90.00%)
         547292415      L1-dcache-load-misses     #    1.32% of all L1-dcache hits    ( +- 99.81% )  (90.01%)
           4817586      LLC-loads                 #    0.139 M/sec                    ( +- 90.74% )  (90.00%)
             55649      LLC-load-misses           #    1.16% of all LL-cache hits     ( +- 68.44% )  (90.00%)

             34.77 +- 34.72 seconds time elapsed  ( +- 99.87% )
```

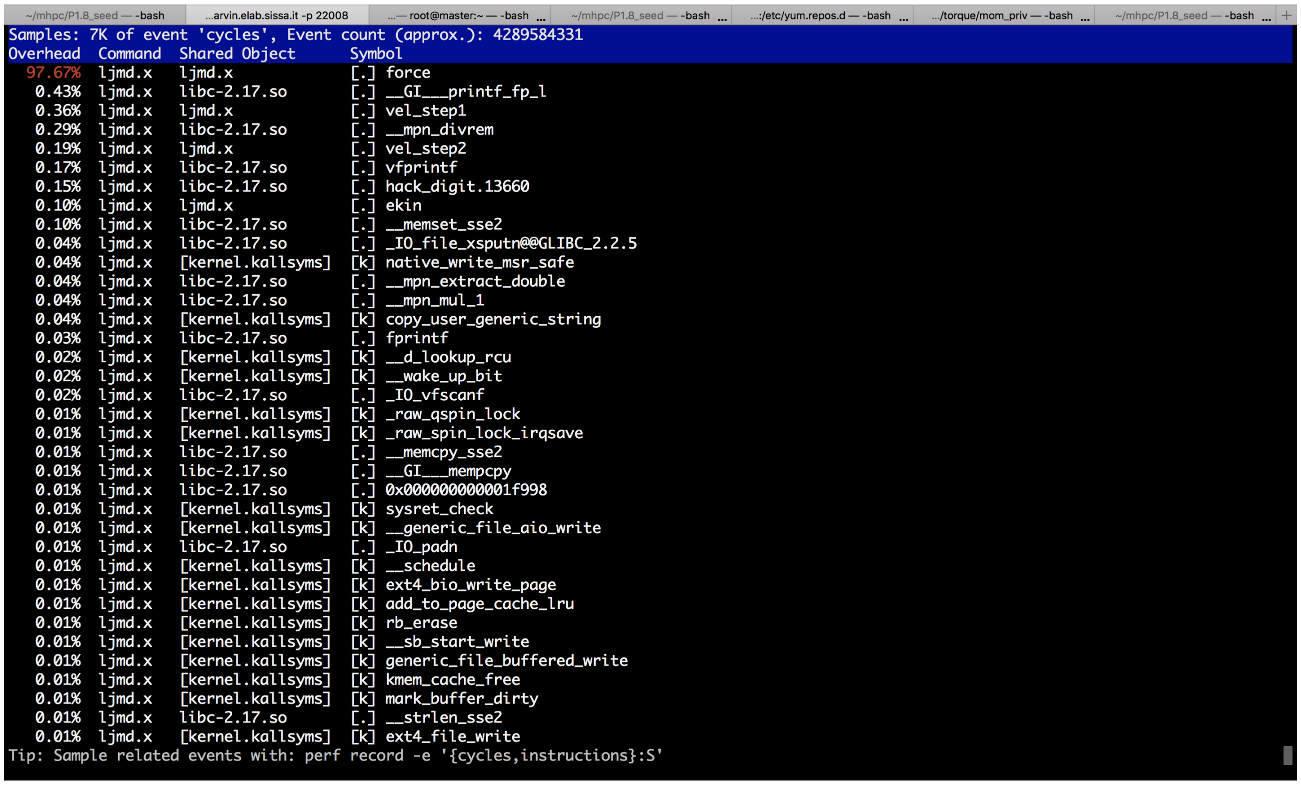
Figure 13: Data obtained on Ulisse and with the using the `perf` for $N = 2916$ for task with python interface with using knowledge of Third Newton's law.

```
Performance counter stats for 'python3 ../ljmd.py' (3 runs):

       32191.80 msec task-clock                #    1.000 CPUs utilized            ( +- 99.86% )
            173      context-switches          #    0.005 K/sec                    ( +- 92.77% )
              1      cpu-migrations            #    0.000 K/sec                    ( +-100.00% )
          10154      page-faults               #    0.315 K/sec                    ( +- 84.23% )
    78719556794      cycles                    #    2.445 GHz                      ( +- 99.92% )  (90.00%)
    22050842846      stalled-cycles-frontend   #   28.01% frontend cycles idle     ( +- 99.85% )  (90.00%)
    12717566544      stalled-cycles-backend    #   16.16% backend cycles idle      ( +- 99.80% )  (80.00%)
   101773405431      instructions              #    1.29  insn per cycle
                                               #    0.22  stalled cycles per insn  ( +- 99.95% )  (90.00%)
    25475179705      branches                  #  791.356 M/sec                    ( +- 99.95% )  (90.00%)
      818649178      branch-misses             #    3.21% of all branches          ( +- 99.94% )  (90.00%)
    41188544129      L1-dcache-loads           # 1279.473 M/sec                    ( +- 99.97% )  (90.00%)
      550701183      L1-dcache-load-misses     #    1.34% of all L1-dcache hits    ( +- 99.81% )  (90.00%)
        4337324      LLC-loads                 #    0.135 M/sec                    ( +- 89.73% )  (90.00%)
          56496      LLC-load-misses           #    1.30% of all LL-cache hits     ( +- 63.63% )  (90.00%)

          32.20 +- 32.16 seconds time elapsed  ( +- 99.86% )
```

Figure 14: Data obtained on Ulisse and with the using the `perf` for $N = 2916$ for task with python interface with avoiding expensive mathematical functions.


~/mhpc/P1.8_seed — -bash
...arvin.elab.sissa.it -p 22008
...— root@master:~ — -bash
~/mhpc/P1.8_seed — -bash
...:/etc/yum.repos.d — -bash
.../torque/mom_priv — -bash
~/mhpc/P1.8_seed — -bash

```
Samples: 7K of event 'cycles', Event count (approx.): 4289584331
Overhead  Command  Shared Object      Symbol
  97.67%  ljmd.x   ljmd.x             [.] force
   0.43%  ljmd.x   libc-2.17.so       [.] __GI___printf_fp_l
   0.36%  ljmd.x   ljmd.x             [.] vel_step1
   0.29%  ljmd.x   libc-2.17.so       [.] __mpn_divrem
   0.19%  ljmd.x   ljmd.x             [.] vel_step2
   0.17%  ljmd.x   libc-2.17.so       [.] vfprintf
   0.15%  ljmd.x   libc-2.17.so       [.] hack_digit.13660
   0.10%  ljmd.x   ljmd.x             [.] ekin
   0.10%  ljmd.x   libc-2.17.so       [.] __memset_sse2
   0.04%  ljmd.x   libc-2.17.so       [.] _IO_file_xsputn@@GLIBC_2.2.5
   0.04%  ljmd.x   [kernel.kallsyms]  [k] native_write_msr_safe
   0.04%  ljmd.x   libc-2.17.so       [.] __mpn_extract_double
   0.04%  ljmd.x   libc-2.17.so       [.] __mpn_mul_1
   0.04%  ljmd.x   [kernel.kallsyms]  [k] copy_user_generic_string
   0.03%  ljmd.x   libc-2.17.so       [.] fprintf
   0.02%  ljmd.x   [kernel.kallsyms]  [k] __d_lookup_rcu
   0.02%  ljmd.x   [kernel.kallsyms]  [k] __wake_up_bit
   0.02%  ljmd.x   libc-2.17.so       [.] _IO_vfscanf
   0.01%  ljmd.x   [kernel.kallsyms]  [k] _raw_qspin_lock
   0.01%  ljmd.x   [kernel.kallsyms]  [k] _raw_spin_lock_irqsave
   0.01%  ljmd.x   libc-2.17.so       [.] __memcpy_sse2
   0.01%  ljmd.x   libc-2.17.so       [.] __GI___mempcpy
   0.01%  ljmd.x   libc-2.17.so       [.] 0x000000000001f998
   0.01%  ljmd.x   [kernel.kallsyms]  [k] sysret_check
   0.01%  ljmd.x   [kernel.kallsyms]  [k] __generic_file_aio_write
   0.01%  ljmd.x   libc-2.17.so       [.] _IO_padn
   0.01%  ljmd.x   [kernel.kallsyms]  [k] __schedule
   0.01%  ljmd.x   [kernel.kallsyms]  [k] ext4_bio_write_page
   0.01%  ljmd.x   [kernel.kallsyms]  [k] add_to_page_cache_lru
   0.01%  ljmd.x   [kernel.kallsyms]  [k] rb_erase
   0.01%  ljmd.x   [kernel.kallsyms]  [k] __sb_start_write
   0.01%  ljmd.x   [kernel.kallsyms]  [k] generic_file_buffered_write
   0.01%  ljmd.x   [kernel.kallsyms]  [k] kmem_cache_free
   0.01%  ljmd.x   [kernel.kallsyms]  [k] mark_buffer_dirty
   0.01%  ljmd.x   libc-2.17.so       [.] __strlen_sse2
   0.01%  ljmd.x   [kernel.kallsyms]  [k] ext4_file_write
Tip: Sample related events with: perf record -e '{cycles,instructions}:S'
```

Figure 15: Data obtained on Ulisse and with the using the `perf record` We can see that main amount of calls (about 98%) it is calls for force function.