|  |  |  |  |
| --- | --- | --- | --- |
| # threads/time | User | System | Wall |
| 1 | 0.003999 | 0.0 | 0.003305 |
| 5 | 0.003999 | 0.0 | 0.003739 |
| 10 | 0.002999 | 0.000999 | 0.003577 |
| 50 | 0.003999 | 0.000999 | 0.005934 |
| 100 | 0.007998 | 0.008998 | 0.024830 |

Max

Sum

|  |  |  |  |
| --- | --- | --- | --- |
| # threads/time | User | System | Wall |
| 1 | 0.002999 | 0.000999 | 0.003547 |
| 5 | 0.003999 | 0.0 | 0.006059 |
| 10 | 0.000999 | 0.008998 | 0.021875 |
| 50 | 0.008998 | 0.004999 | 0.019108 |
| 100 | 0.003999 | 0.012998 | 0.018697 |

In order to test the efficiency of adding more threads to our program we analyzed the results of running each program with the same seed and number of elements. While we did test this over varying array sizes, the example above is using an array of 100,000 elements. Process time refers to the amount of time the program is running and is made up of user time (the time the OS was running the code) and system time (the time the OS is running system code). We can think of threads like the amount of people in our group and the sum or max function like our exercise. Adding more people to our group will continue to improve efficiency but as that number grows and grows, the rate of change in efficiency will continue to decline (reaching a limit of 0) because a point will be reached where the extra people/threads aren’t changing the number of concurrent tasks being performed.