1. Scatter Plots & Relationships among `/proc/*/sched` Statistics

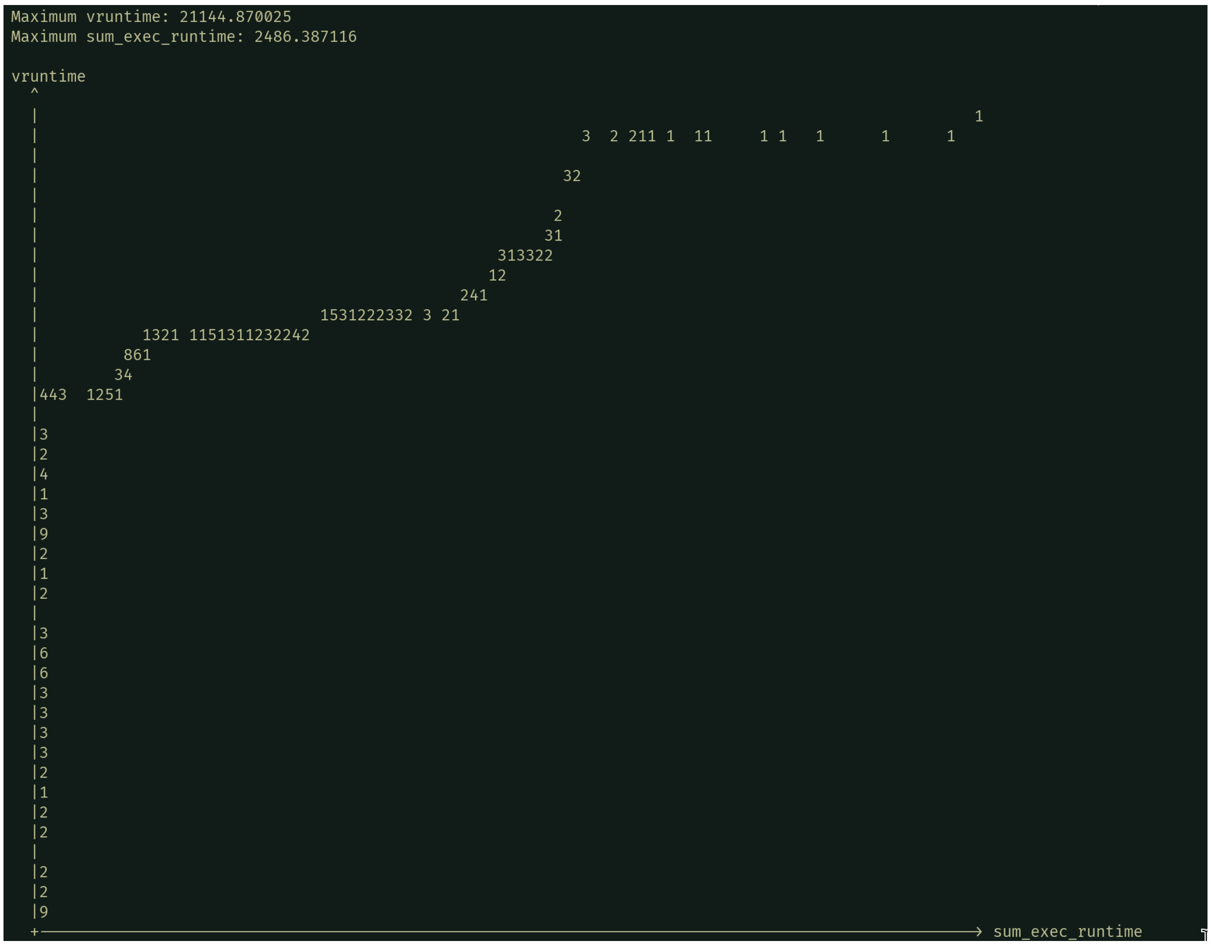
I wrote the following awk script to create a scatter plot from any two variables in the in the `/proc/*/sched` files. `sched_scatter_plot.awk`:

```
BEGIN {
  num_processes = 0;

  if (x_axis=="") { x_axis = "sum_exec_runtime"; }
  if (y_axis=="") { y_axis = "vruntime"; }

  if (x=="") { x = 100; }
  if (y=="") { y = 30; }
}

FNR == 1 {
  num_processes++;
}

match($1, y_axis) { y_axis_vals[num_processes-1]=$3; next; }
match($1, x_axis) { x_axis_vals[num_processes-1]=$3; next; }

END {
  asort(y_axis_vals);
  asort(x_axis_vals);

  printf("Maximum %s: %f\n", y_axis, y_axis_vals[num_processes]);
  printf("Maximum %s: %f\n\n", x_axis, x_axis_vals[num_processes]);

  if (log_scale_x > 1 || log_scale_y > 1) {
    for (i=1; i<=num_processes; i++) {
      if (log_scale_x) {
        x_axis_vals[i] = log(x_axis_vals[i] + 1) / log(log_scale_x);
      } if (log_scale_y) {
        y_axis_vals[i] = log(y_axis_vals[i] + 1) / log(log_scale_y);
      }
    }
  }

  print y_axis;

  x_max = x_axis_vals[num_processes];
  y_max = y_axis_vals[num_processes];
  x_scale = (x_max > 0) ? x / x_max : 1;
  y_scale = (y_max > 0) ? y / y_max : 1;

  for (i=1; i<=num_processes; i++) {
    x_coord = int(x_scale * x_axis_vals[i]);
    y_coord = int(y_scale * y_axis_vals[i]);

    plot[x_coord, y_coord] += 1;
    if (plot[x_coord, y_coord] > 9) { plot[x_coord, y_coord] = 9; }
  }

  print "  ^";
  for (curr_y=y; curr_y >= 0; curr_y--) {
    printf "  |";
    for (curr_x=0; curr_x<=x; curr_x++) {
      printf (plot[curr_x, curr_y] ? plot[curr_x, curr_y] : " ");
    }
    printf "\n";
  }

  printf "  +";
  for (i=0; i<x; i++) { printf "-"; }
  printf ("> %s\n", x_axis);
}
```

> Note: I prompted ChatGPT with the prompt “Why isn’t the graph produced by this awk graph scaling correctly” in order to debug an error in my code.

This awk script was used to generate log-scaled graphs showing the relationship between the inputted variables. The numbers on the graphs represent how many processes had the corresponding relationship between the variables, i.e. how many dots there would be at any given position in a scatter plot.

Before generating the graphs, I restarted the virtual machine and waited a approximately 10 minutes to make the data consistent with section 2.



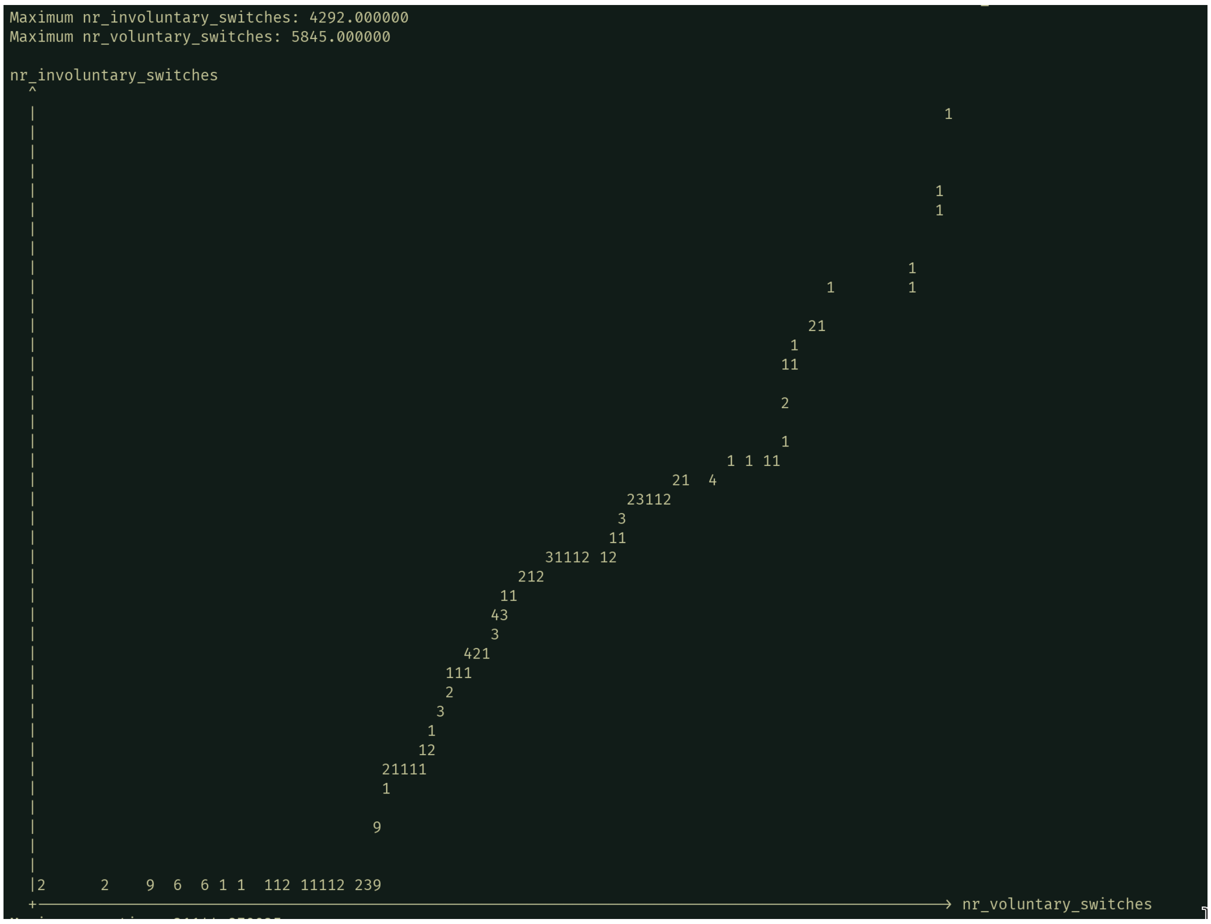
```
Maximum vruntime: 21144.870025
Maximum sum_exec_runtime: 2486.387116

vruntime
  ^
  |                                                                                                    1
  |                                                          3  2 211 1  11     1 1   1      1      1
  |
  |                                                        32
  |
  |                                                       2
  |                                                      31
  |                                                 313322
  |                                                12
  |                                             241
  |                              1531222332 3 21
  |           1321 1151311232242
  |         861
  |        34
  |443  1251
  |
  |3
  |2
  |4
  |1
  |3
  |9
  |2
  |1
  |2
  |
  |3
  |6
  |6
  |3
  |3
  |3
  |3
  |2
  |1
  |2
  |2
  |
  |2
  |2
  |9
  +----------------------------------------------------------------------------------------------------> sum_exec_runtime
```

Figure 1: ‘vruntime’ vs. ‘sum_exec_runtime’

> Note: I now realize that I was supposed to make this graph showing log(vruntime) vs sum_exec_runtime, not both log-scaled. However, it took forever to actually get the data from sections 1 and 2 to be consistent outside of the kernel parameters, so I am just going to leave a correct graph in the Appendix, sorry about that.

Figure 1 shows the relationship of vruntime and total CPU runtime of all the processes on my root machine. There is a relatively strong logarithmic correlation between vruntime and total CPU time. Figure 1 was produced using the following bash command:

```
awk -f sched_scatter_plot.awk -v x=100 -v y=40
  -v x_axis=sum_exec_runtime
  -v y_axis=vruntime
  -v log_scale_x=2
  -v log_scale_y=2
  /proc/*/sched
  > vruntime_vs_sumexectime.plot
```

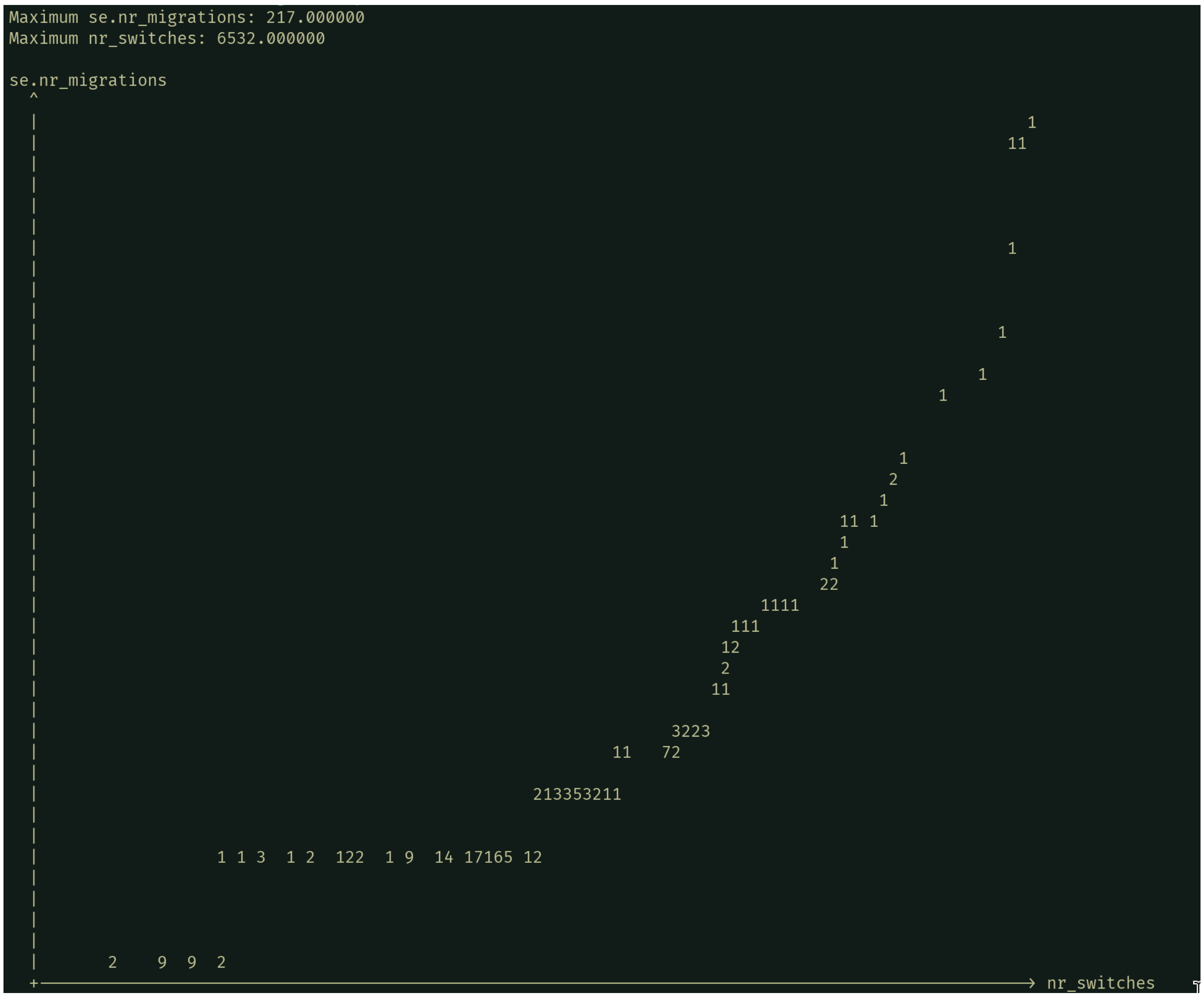


```
Maximum nr_involuntary_switches: 4292.000000
Maximum nr_voluntary_switches: 5845.000000

nr_involuntary_switches
  ^
  |                                                                                                    1
  |
  |
  |
  |                                                                                                   1
  |                                                                                                   1
  |
  |
  |                                                                                                1
  |                                                                                       1        1
  |
  |                                                                                     21
  |                                                                                   1
  |                                                                                  11
  |
  |                                                                                  2
  |
  |                                                                                  1
  |                                                                            1 1 11
  |                                                                      21  4
  |                                                                 23112
  |                                                                3
  |                                                               11
  |                                                        31112 12
  |                                                     212
  |                                                   11
  |                                                  43
  |                                                  3
  |                                               421
  |                                             111
  |                                             2
  |                                            3
  |                                           1
  |                                          12
  |                                      21111
  |                                      1
  |
  |                                     9
  |
  |
  |2      2    9  6  6 1 1  112 11112 239
  +----------------------------------------------------------------------------------------------------> nr_voluntary_switches
```

Figure 2: ‘voluntary_switches’ vs. ‘involuntary_switches’

Figure 2 shows the relationship of voluntary and involuntary context switches of all the processes on my root machine; there is a relatively strong direct correlation. It was produced using the following bash command:

```
awk -f sched_scatter_plot.awk -v x=100 -v y=40
  -v x_axis=nr_voluntary_switches
  -v y_axis=nr_involuntary_switches
  -v log_scale_x=2
  -v log_scale_y=2
  /proc/*/sched
  > voluntary_vs_involuntary_switches.plot
```



```
Maximum se.nr_migrations: 217.000000
Maximum nr_switches: 6532.000000
```

Figure 3: 'switches' vs. 'migrations'

Figure 3 shows the relationship of context switches and CPU migrations of all the processes on my root machine; there is a relatively strong direct relationship. It was produced using the following bash command:

```
awk -f sched_scatter_plot.awk -v x=100 -v y=40
  -v x_axis=nr_switches
  -v y_axis=se.nr_migrations
  -v log_scale_x=2
  -v log_scale_y=2
  /proc/*/sched
  > switches_vs_migrations.plot
```

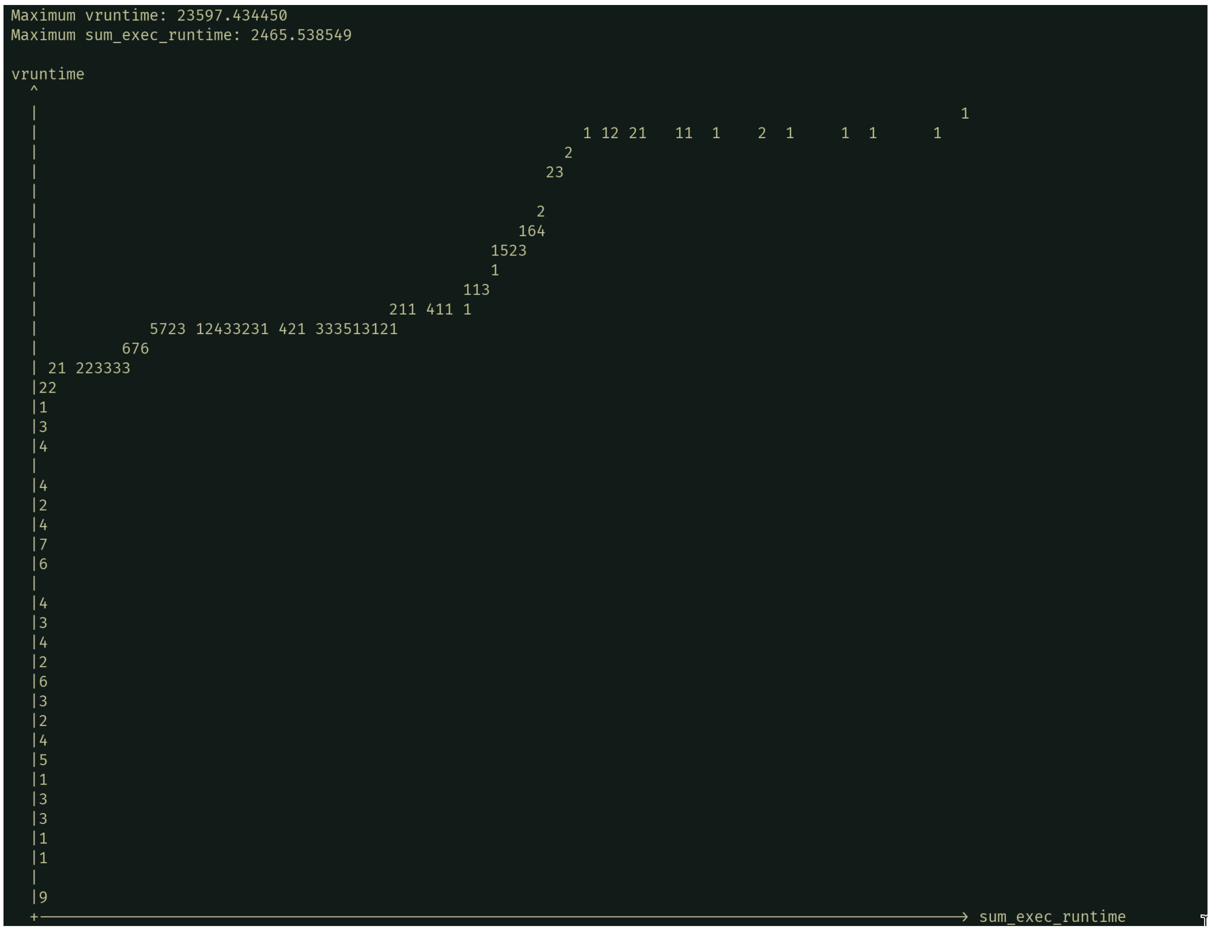
2. `/proc/*/sched` Statistics with Kernel Parameter Manipulation

For this section, I began by creating a bash script that increased all of the specified kernel parameters by a factor of 10:

```bash
sysctl -w kernel.sched_latency_ns=60000000
  -w kernel.sched_migration_cost_ns=5000000
  -w kernel.sched_min_granularity_ns=7500000
  -w kernel.sched_nr_migrate=320
  -w kernel.sched_rr_timeslice_ms=1000
  -w kernel.sched_rt_period_us=10000000
  -w kernel.sched_rt_runtime_us=9500000
  -w kernel.sched_shares_window_ns=100000000
```

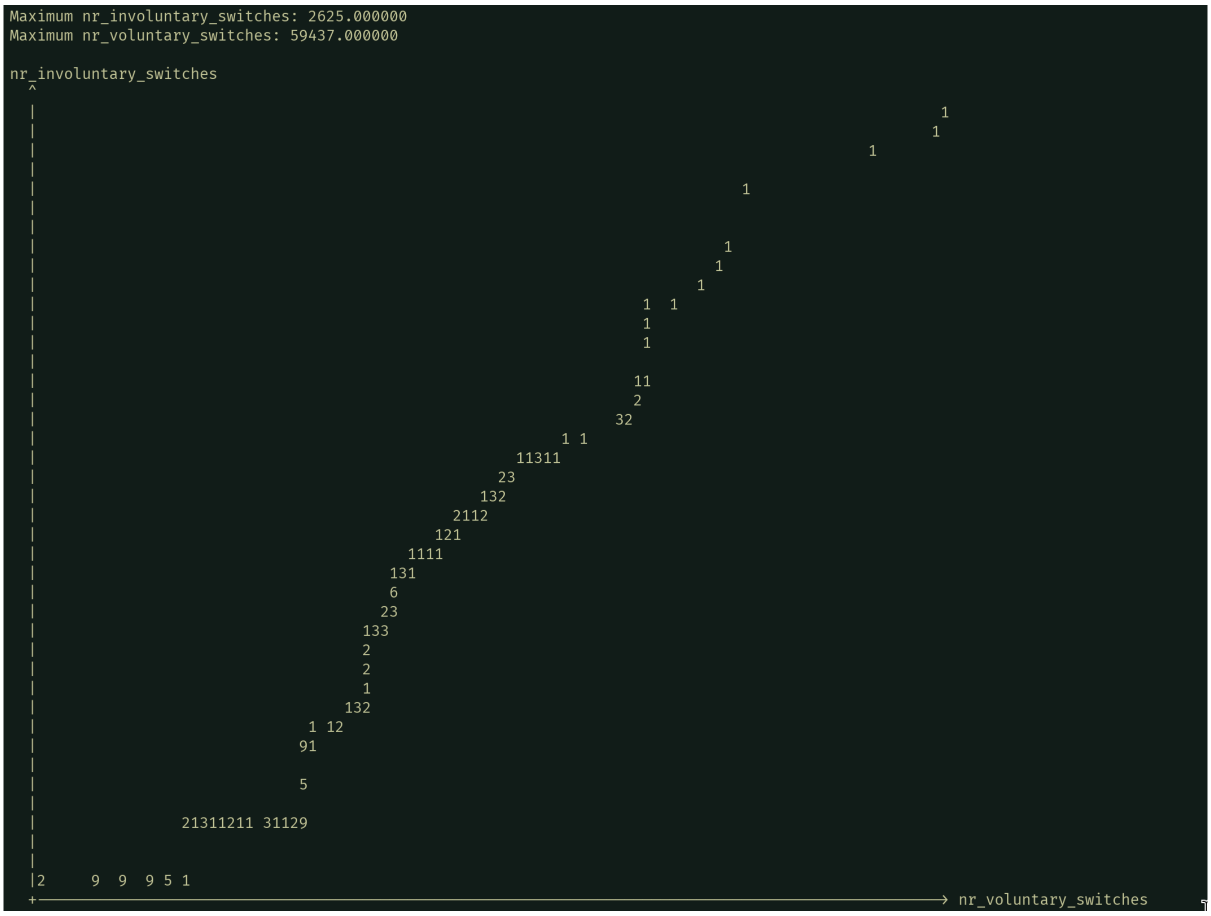
> Note: the last two kernel parameters specified by the assignment (kernel.sched_shares_window_ns and kernel.sched_time_avg_ms) didn't exist on my root machine according to `sysctl`.

Then, I restarted the virtual machine and ran the bash script to change the parameters immediately. I then waited 10 minutes and generated the plots in the using the same script and commands as section 1:



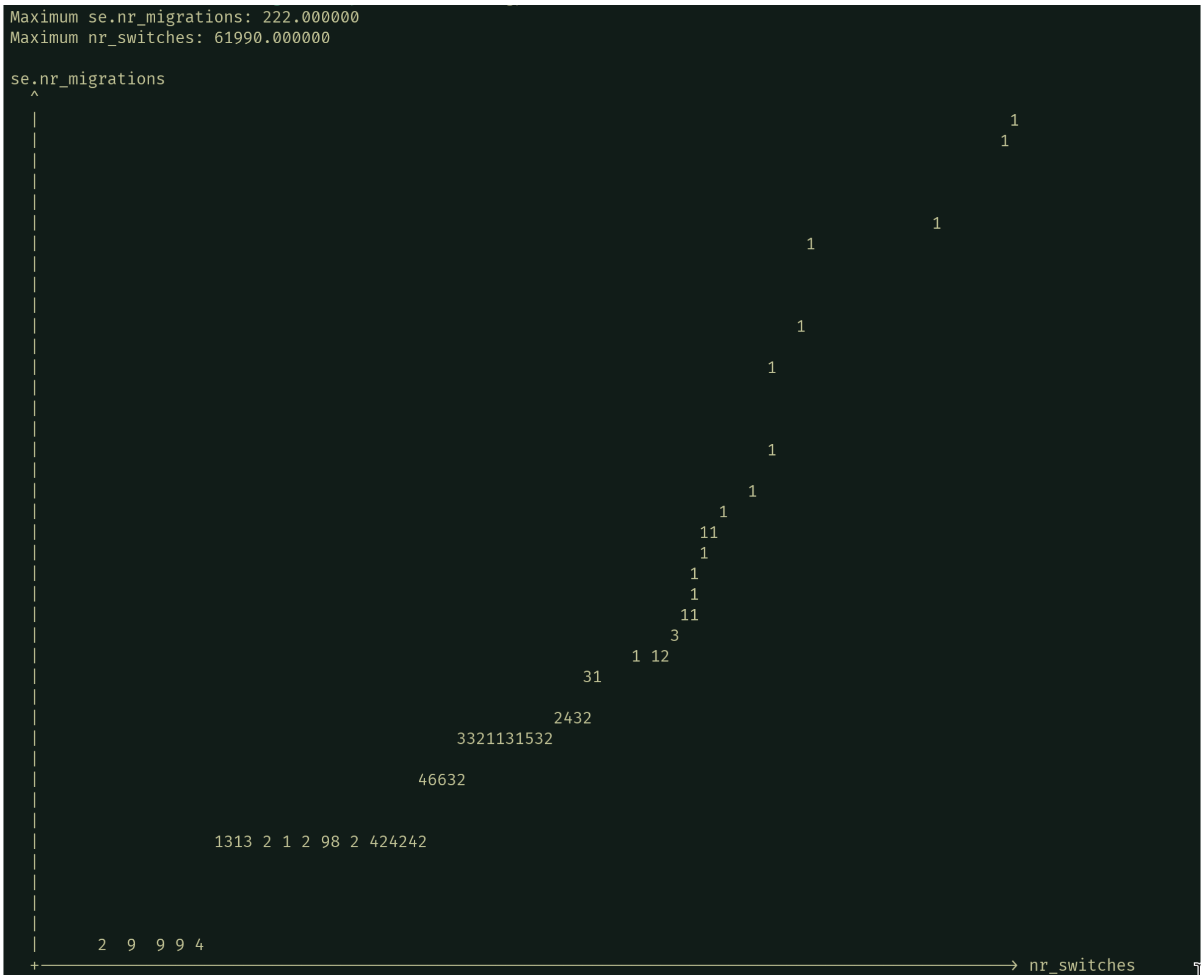
```
Maximum vruntime: 23597.434450
Maximum sum_exec_runtime: 2465.538549

vruntime
  ^
  |                                                                                                    1
  |                                                           1 12 21   11  1    2  1     1  1      1
  |                                                         2
  |                                                       23
  |
  |                                                      2
  |                                                    164
  |                                                 1523
  |                                                 1
  |                                              113
  |                                      211 411 1
  |            5723 12433231 421 333513121
  |         676
  | 21 223333
  |22
  |1
  |3
  |4
  |
  |4
  |2
  |4
  |7
  |6
  |
  |4
  |3
  |4
  |2
  |6
  |3
  |2
  |4
  |5
  |1
  |3
  |3
  |1
  |1
  |
  |9
  +────────────────────────────────────────────────────────────────────────────────────────────────────────→ sum_exec_runtime
```

Figure 4: 'vruntime' vs. 'sum_exec_runtime', multiplied kernel parameters



```
Maximum nr_involuntary_switches: 2625.000000
Maximum nr_voluntary_switches: 59437.000000

nr_involuntary_switches
  ^
  |                                                                                                    1
  |                                                                                                   1
  |                                                                                            1
  |
  |                                                                              1
  |
  |
  |                                                                            1
  |                                                                           1
  |                                                                         1
  |                                                                   1  1
  |                                                                   1
  |                                                                   1
  |
  |                                                                  11
  |                                                                  2
  |                                                                32
  |                                                          1 1
  |                                                     11311
  |                                                   23
  |                                                 132
  |                                              2112
  |                                            121
  |                                         1111
  |                                       131
  |                                       6
  |                                      23
  |                                    133
  |                                    2
  |                                    2
  |                                    1
  |                                  132
  |                              1 12
  |                             91
  |
  |                             5
  |
  |                21311211 31129
  |
  |
  |2     9  9  9 5 1
  +────────────────────────────────────────────────────────────────────────────────────────────────────→ nr_voluntary_switches
```

Figure 5: ‘voluntary_switches’ vs. ‘involuntary_switches’ multiplied kernel parameters



```
Maximum se.nr_migrations: 222.000000
Maximum nr_switches: 61990.000000

se.nr_migrations
  ^
  |                                                                                                    1
  |                                                                                                   1
  |
  |
  |
  |                                                                                            1
  |                                                                               1
  |
  |
  |
  |                                                                              1
  |
  |                                                                           1
  |
  |
  |
  |                                                                           1
  |
  |                                                                         1
  |                                                                      1
  |                                                                    11
  |                                                                    1
  |                                                                   1
  |                                                                   1
  |                                                                  11
  |                                                                 3
  |                                                             1 12
  |                                                        31
  |
  |                                                     2432
  |                                           3321131532
  |
  |                                       46632
  |
  |
  |                  1313 2 1 2 98 2 424242
  |
  |
  |
  |
  |      2  9  9 9 4
  +────────────────────────────────────────────────────────────────────────────────────────────────────→ nr_switches
```

Figure 6: ‘switches’ vs. ‘migrations’ multiplied kernel parameters

After increasing the kernel parameters tenfold, the relationship between voluntary and involuntary switches changed the most out of the three graphs created.

As figures 5 and 6 show, the number of context switches after multiplying the kernel parameters increased by a factor of about 10 from a maximum of 6532 to 61990 switches. Additionally, the number of involuntary_switches actually decreased despite the larger number of switches; this was likely due to the increased timeslice, which allowed processes to finish their work within a timeslice more often and give up their CPU time voluntarily more often.

> Note that the spread of these graphs look very similar despite tenfold differences due to the log-scaling and differences in zooming/linear scaling.

Another notable comparison is that, although the number of switches increased dramatically, the number of migrations actually stayed somewhat constant, meaning there were 10x more switches per migration. This is probably due to increasing the cost of migration.

The relationship between vruntime and total CPU time stayed approximately the same after changing the kernel parameters.

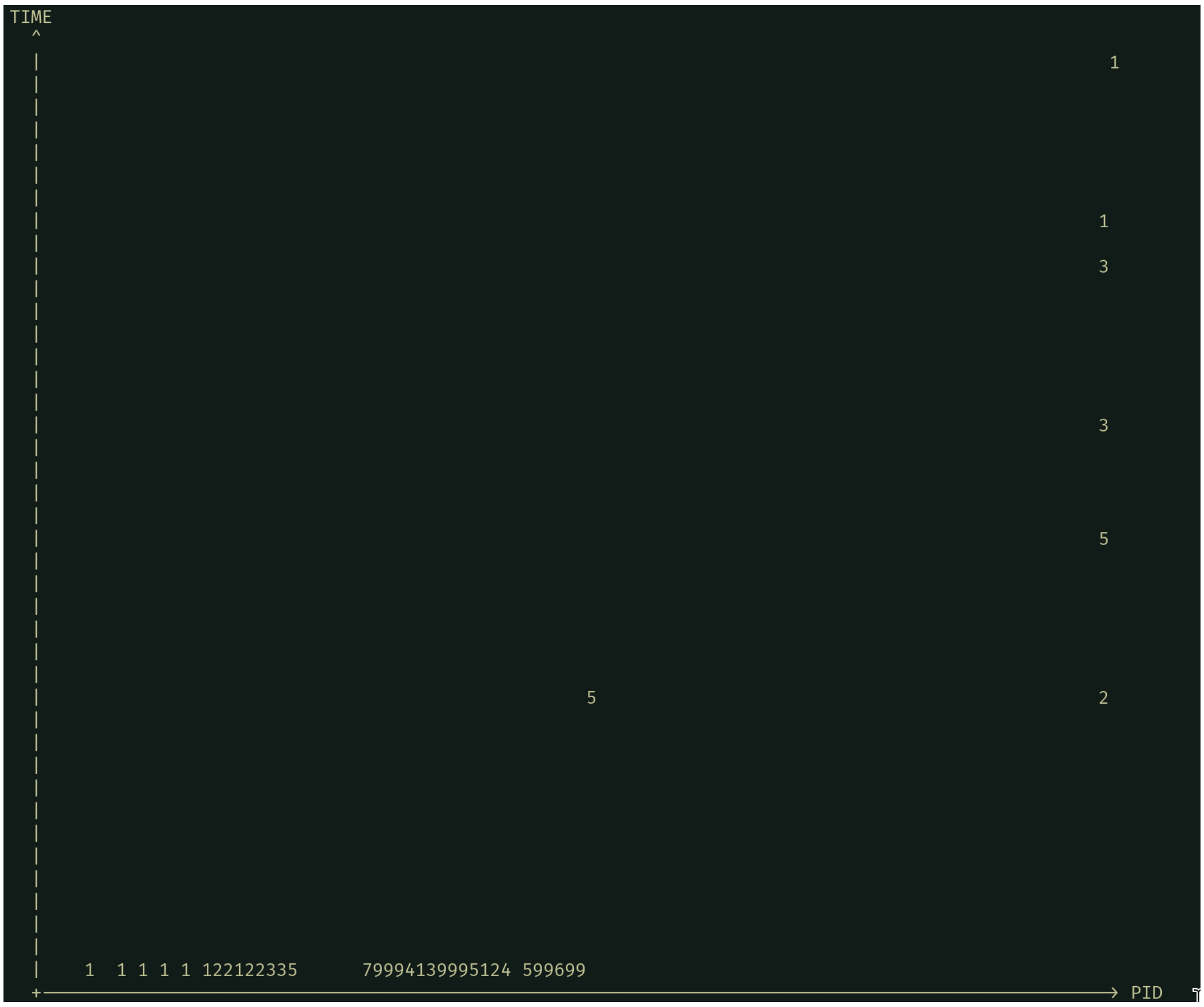
3. Scatter Plots & Relationships for `top` Statistics

For this section, I began by enabling the CODE and DATA' fields in my.toprc' configuration, then I used the following command to output the top data to a file for the awk script to read:

```
top -b -n 1 > top.out
```

Then, I manipulated my scatter_plot awk script from section 1 so that I could read data from the top output and specify the top fields that I want to read:

```
BEGIN {
  num_processes = 0;
  processing_lines_bool = 0;

  if (x_axis=="") { x_axis = "PID"; }
  if (y_axis=="") { y_axis = "TIME"; }

  if (x=="") { x = 100; }
  if (y=="") { y = 30; }
}


match($0, x_axis) && match($0, y_axis) {
  processing_lines_bool = 1;
  print $0;

  for (i=1; i<100; i++) {
    if (match($i, x_axis)) {
      x_col = i;
    }
    if (match($i, y_axis)) {
      y_col = i;
    }
  }

  next;
}

processing_lines_bool { num_processes++;
  x_axis_vals[num_processes]=$x_col;
  y_axis_vals[num_processes]=$y_col;
}

END {
  print num_processes;

  if (!processing_lines_bool) {
    print "Processes not found";
    exit;
  }

  asort(y_axis_vals);
  asort(x_axis_vals);

  print y_axis;
  if (log_scale_x > 1 || log_scale_y > 1) {
    for (i=1; i<=num_processes; i++) {
      if (log_scale_x) {
        x_axis_vals[i] = log(x_axis_vals[i] + 1) / log(log_scale_x);
      } if (log_scale_y) {
        y_axis_vals[i] = log(y_axis_vals[i] + 1) / log(log_scale_y);
      }
    }
  }

  x_max = x_axis_vals[num_processes];
  y_max = y_axis_vals[num_processes];
  x_scale = (x_max > 0) ? x / x_max : 1;
  y_scale = (y_max > 0) ? y / y_max : 1;

  for (i=1; i<=num_processes; i++) {
    x_coord = int(x_scale * x_axis_vals[i]);
    y_coord = int(y_scale * y_axis_vals[i]);

    plot[x_coord, y_coord] += 1;
    if (plot[x_coord, y_coord] > 9) { plot[x_coord, y_coord] = 9; }
  }

  print "  ^";
  for (curr_y=y; curr_y >= 0; curr_y--) {
    printf "  |";
    for (curr_x=0; curr_x<=x; curr_x++) {
      printf (plot[curr_x, curr_y] ? plot[curr_x, curr_y] : " ");
    }
    printf "\n";
  }

  printf "  +";
  for (i=0; i<x; i++) { printf "-"; }
  printf ("> %s\n", x_axis);

}
```



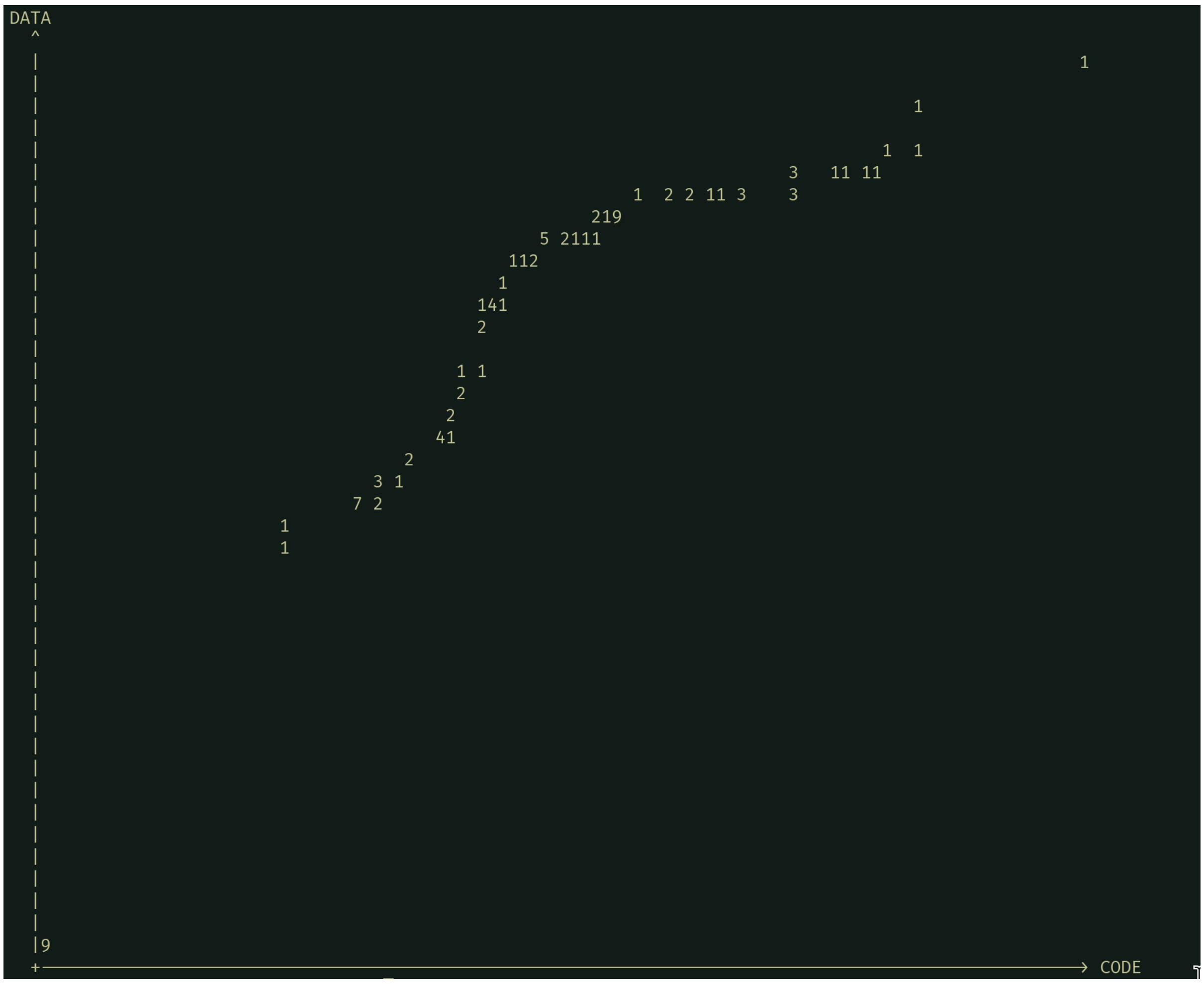
```
TIME
   ^
   |                                                                                                      1
   |
   |
   |
   |
   |
   |
   |                                                                                                     1
   |
   |                                                                                                     3
   |
   |
   |
   |
   |
   |
   |                                                                                                     3
   |
   |
   |
   |
   |                                                                                                     5
   |
   |
   |
   |
   |
   |
   |                                                    5                                                2
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |
   |    1  1 1 1 1 122122335      79994139995124 599699
   +-----------------------------------------------------------------------------------------------------> PID
```

Figure 7: ‘PID’ vs. ‘TIME’

Figure 10 shows the relationship between process ID and total CPU time of all the processes the machine. It was produced using the following bash command:

```
awk -f top_scatter_plot.awk -v x_axis="PID" -v y_axis="TIME" -v y=40
  -v log_scale_x=2 -v log_scale_y=2 top.out
```

The relationship between `TIME` and `PID` shows that (for the most part) only processes with very high process IDs last a very long time.



```
DATA
  ^
  |                                                                                                    1
  |
  |                                                                                    1
  |
  |                                                                                 1  1
  |                                                                        3   11 11
  |                                                         1  2 2 11 3    3
  |                                                     219
  |                                                5 2111
  |                                             112
  |                                            1
  |                                          141
  |                                          2
  |
  |                                        1 1
  |                                        2
  |                                       2
  |                                      41
  |                                   2
  |                                3 1
  |                              7 2
  |                       1
  |                       1
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |9
  +----------------------------------------------------------------------------------------------------> CODE
```

Figure 8: 'CODE' vs. 'DATA'

Figure 11 shows the relationship between the size of executable's machine code and the corresponding amount of memory the process has allocated for all the processes on the machine. It was produced using the following bash command:

```
awk -f top_scatter_plot.awk -v x_axis="CODE" -v y_axis="DATA" -v y=40
  -v log_scale_x=2 -v log_scale_y=2 top.out
```

This relationship shows that the amount of machine code in a program and the amount of data that program uses is directly correlated.

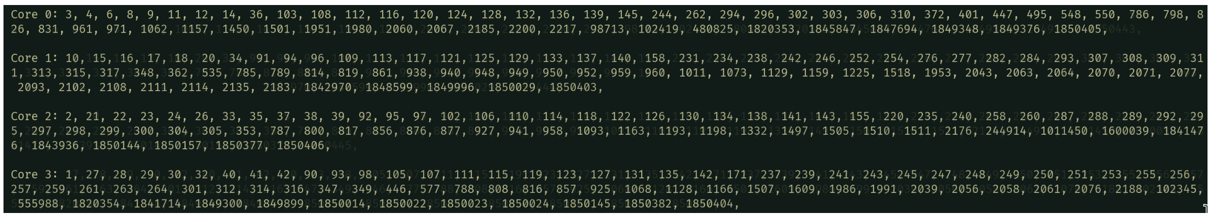
4. CPU Migrations Animation

This was the C program that I wrote (i.e. my modification of `c3.c`) to create an animation showing which processes were on which CPU core.

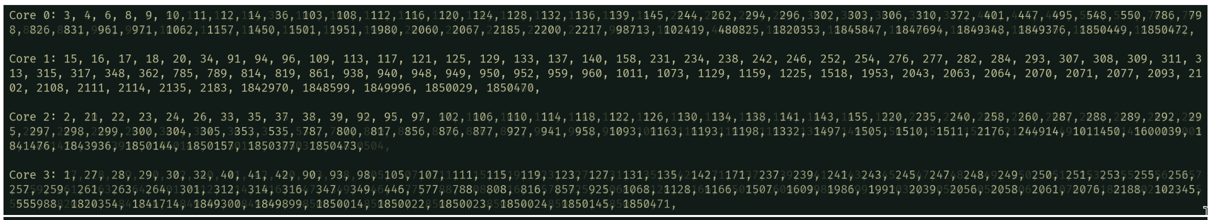
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  FILE *fp;
  char pid[10], user[32], pr[8], ni[8], virt[16], res[16], shr[16], s[4];
  char cpu[8], mem[8], time[32], command[128];
  unsigned p;

  char final_strings[4][2048];

  for (int i = 0; i < 10; i++) {
    memset(final_strings, 0, sizeof(final_strings));

    system("top -b -n 1 | tail -n +8 | sort > out.top");

    fp = fopen("out.top", "r");
    if (fp == NULL) {
      perror("Error opening file");
      return 1;
    }

    char buffer[1024];

    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
      sscanf(buffer, " %s %s %s %s %s %s %s %s %s %s %s %s %d\n", pid, user, pr,
             ni, virt, res, shr, s, cpu, mem, time, command, &p);
      char pid_buf[32];
      sprintf(pid_buf, "%s, ", pid);
      strcat(final_strings[p], pid_buf);
    }

    for (int p = 0; p < 4; p++) {
      printf("Core %d: %s\n\n", p, final_strings[p]);
    }

    fclose(fp);

    system("sleep 1");
    system("clear");
  }

  return 0;
}
```

> The my changes from c3.c to this code are shown in the Appendix.

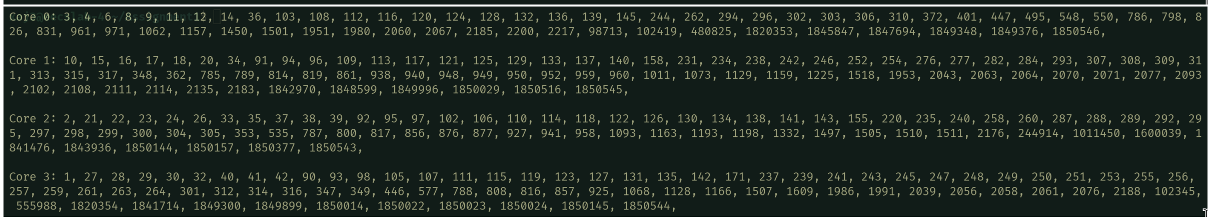
I compiled it and ran it to create the following output snapshots:



```
Core 0: 3, 4, 6, 8, 9, 11, 12, 14, 36, 103, 108, 112, 116, 120, 124, 128, 132, 136, 139, 145, 244, 262, 294, 296, 302, 303, 306, 310, 372, 401, 447, 495, 548, 550, 786, 798, 8
26, 831, 961, 971, 1062, 1157, 1450, 1501, 1951, 1980, 2060, 2067, 2185, 2200, 2217, 98713, 102419, 480825, 1820353, 1845847, 1847694, 1849348, 1849376, 1850405,

Core 1: 10, 15, 16, 17, 18, 20, 34, 91, 94, 96, 109, 113, 117, 121, 125, 129, 133, 137, 140, 158, 231, 234, 238, 242, 246, 252, 254, 276, 277, 282, 284, 293, 307, 308, 309, 31
1, 313, 315, 317, 348, 362, 535, 785, 789, 814, 819, 861, 938, 940, 948, 949, 950, 952, 959, 960, 1011, 1073, 1129, 1159, 1225, 1518, 1953, 2043, 2063, 2064, 2070, 2071, 2077,
 2093, 2102, 2108, 2111, 2114, 2135, 2183, 1842970, 1848599, 1849996, 1850029, 1850403,

Core 2: 2, 21, 22, 23, 24, 26, 33, 35, 37, 38, 39, 92, 95, 97, 102, 106, 110, 114, 118, 122, 126, 130, 134, 138, 141, 143, 155, 220, 235, 240, 258, 260, 287, 288, 289, 292, 29
5, 297, 298, 299, 300, 304, 305, 353, 787, 800, 817, 856, 876, 877, 927, 941, 958, 1093, 1163, 1193, 1198, 1332, 1497, 1505, 1510, 1511, 2176, 244914, 1011450, 1600039, 184147
6, 1843936, 1850144, 1850157, 1850377, 1850406,

Core 3: 1, 27, 28, 29, 30, 32, 40, 41, 42, 90, 93, 98, 105, 107, 111, 115, 119, 123, 127, 131, 135, 142, 171, 237, 239, 241, 243, 245, 247, 248, 249, 250, 251, 253, 255, 256,
257, 259, 261, 263, 264, 301, 312, 314, 316, 347, 349, 446, 577, 788, 808, 816, 857, 925, 1068, 1128, 1166, 1507, 1609, 1986, 1991, 2039, 2056, 2058, 2061, 2076, 2188, 102345,
 555988, 1820354, 1841714, 1849300, 1849899, 1850014, 1850022, 1850023, 1850024, 1850145, 1850382, 1850404,
```



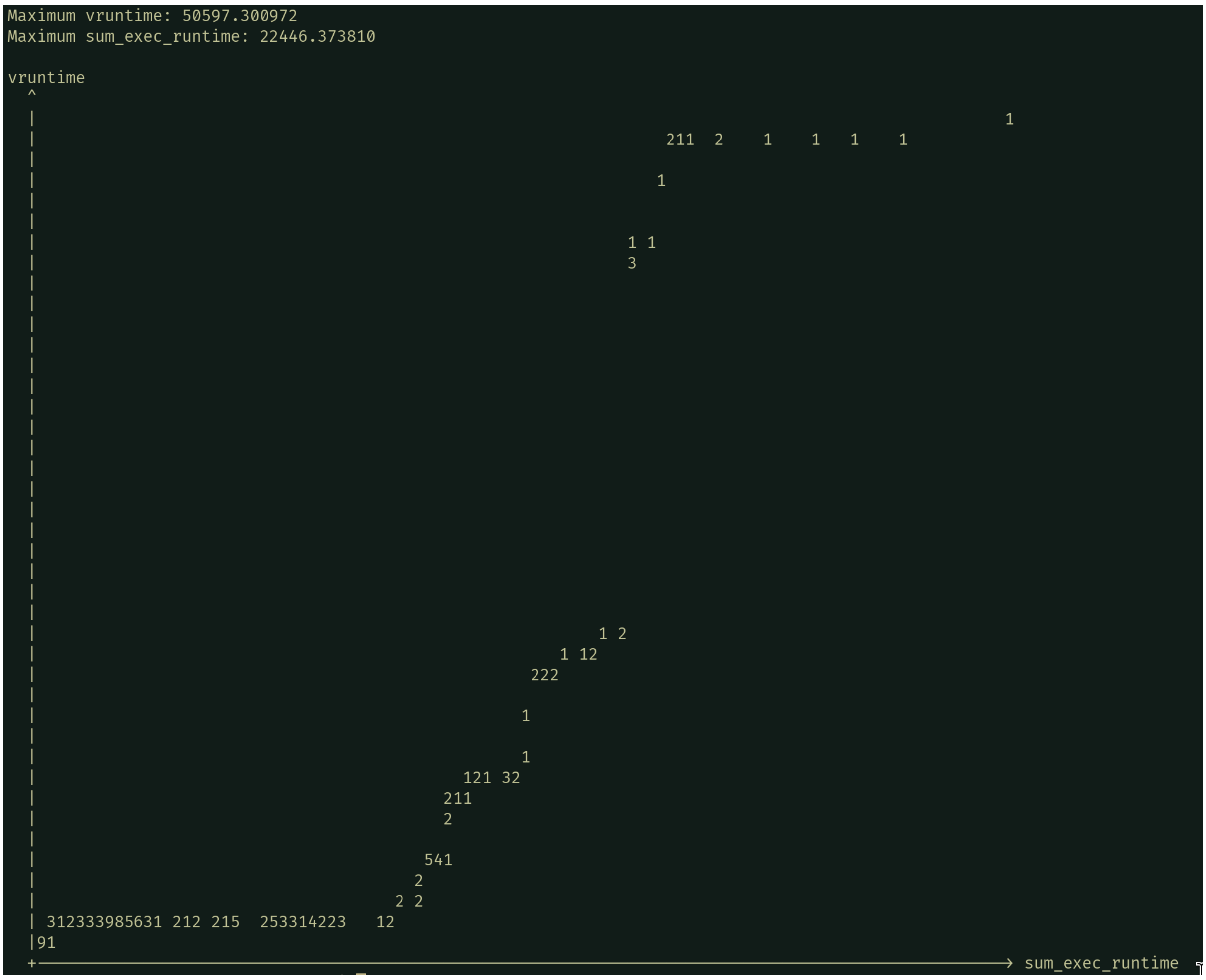
```
Core 0: 3, 4, 6, 8, 9, 10, 11, 12, 14, 36, 103, 108, 112, 116, 120, 124, 128, 132, 136, 139, 145, 244, 262, 294, 296, 302, 303, 306, 310, 372, 401, 447, 495, 548, 550, 786, 79
8, 826, 831, 961, 971, 1062, 1157, 1450, 1501, 1951, 1980, 2060, 2067, 2185, 2200, 2217, 98713, 102419, 480825, 1820353, 1845847, 1847694, 1849348, 1849376, 1850449, 1850472,

Core 1: 15, 16, 17, 18, 20, 34, 91, 94, 96, 109, 113, 117, 121, 125, 129, 133, 137, 140, 158, 231, 234, 238, 242, 246, 252, 254, 276, 277, 282, 284, 293, 307, 308, 309, 311, 3
13, 315, 317, 348, 362, 785, 789, 814, 819, 861, 938, 940, 948, 949, 950, 952, 959, 960, 1011, 1073, 1129, 1159, 1225, 1518, 1953, 2043, 2063, 2064, 2070, 2071, 2077, 2093, 21
02, 2108, 2111, 2114, 2135, 2183, 1842970, 1848599, 1849996, 1850029, 1850470,

Core 2: 2, 21, 22, 23, 24, 26, 33, 35, 37, 38, 39, 92, 95, 97, 102, 106, 110, 114, 118, 122, 126, 130, 134, 138, 141, 143, 155, 220, 235, 240, 258, 260, 287, 288, 289, 292, 29
5, 297, 298, 299, 300, 304, 305, 353, 535, 787, 800, 817, 856, 876, 877, 927, 941, 958, 1093, 1163, 1193, 1198, 1332, 1497, 1505, 1510, 1511, 2176, 244914, 1011450, 1600039, 1
841476, 1843936, 1850144, 1850157, 1850377, 1850473,

Core 3: 1, 27, 28, 29, 30, 32, 40, 41, 42, 90, 93, 98, 105, 107, 111, 115, 119, 123, 127, 131, 135, 142, 171, 237, 239, 241, 243, 245, 247, 248, 249, 250, 251, 253, 255, 256,
257, 259, 261, 263, 264, 301, 312, 314, 316, 347, 349, 446, 577, 788, 808, 816, 857, 925, 1068, 1128, 1166, 1507, 1609, 1986, 1991, 2039, 2056, 2058, 2061, 2076, 2188, 102345,
 555988, 1820354, 1841714, 1849300, 1849899, 1850014, 1850022, 1850023, 1850024, 1850145, 1850471,
```



```
Core 0: 3, 4, 6, 8, 9, 11, 12, 14, 36, 103, 108, 112, 116, 120, 124, 128, 132, 136, 139, 145, 244, 262, 294, 296, 302, 303, 306, 310, 372, 401, 447, 495, 548, 550, 786, 798, 8
26, 831, 961, 971, 1062, 1157, 1450, 1501, 1951, 1980, 2060, 2067, 2185, 2200, 2217, 98713, 102419, 480825, 1820353, 1845847, 1847694, 1849348, 1849376, 1850546,

Core 1: 10, 15, 16, 17, 18, 20, 34, 91, 94, 96, 109, 113, 117, 121, 125, 129, 133, 137, 140, 158, 231, 234, 238, 242, 246, 252, 254, 276, 277, 282, 284, 293, 307, 308, 309, 31
1, 313, 315, 317, 348, 362, 785, 789, 814, 819, 861, 938, 940, 948, 949, 950, 952, 959, 960, 1011, 1073, 1129, 1159, 1225, 1518, 1953, 2043, 2063, 2064, 2070, 2071, 2077, 2093
, 2102, 2108, 2111, 2114, 2135, 2183, 1842970, 1848599, 1849996, 1850029, 1850516, 1850545,

Core 2: 2, 21, 22, 23, 24, 26, 33, 35, 37, 38, 39, 92, 95, 97, 102, 106, 110, 114, 118, 122, 126, 130, 134, 138, 141, 143, 155, 220, 235, 240, 258, 260, 287, 288, 289, 292, 29
5, 297, 298, 299, 300, 304, 305, 353, 535, 787, 800, 817, 856, 876, 877, 927, 941, 958, 1093, 1163, 1193, 1198, 1332, 1497, 1505, 1510, 1511, 2176, 244914, 1011450, 1600039, 1
841476, 1843936, 1850144, 1850157, 1850377, 1850543,

Core 3: 1, 27, 28, 29, 30, 32, 40, 41, 42, 90, 93, 98, 105, 107, 111, 115, 119, 123, 127, 131, 135, 142, 171, 237, 239, 241, 243, 245, 247, 248, 249, 250, 251, 253, 255, 256,
257, 259, 261, 263, 264, 301, 312, 314, 316, 347, 349, 446, 577, 788, 808, 816, 857, 925, 1068, 1128, 1166, 1507, 1609, 1986, 1991, 2039, 2056, 2058, 2061, 2076, 2188, 102345,
 555988, 1820354, 1841714, 1849300, 1849899, 1850014, 1850022, 1850023, 1850024, 1850145, 1850544,
```

Appendix

Here is a scatter plot showing `log(vruntime)` vs. `sum_exec_runtime`; there is a weak linear correlation.



```
Maximum vruntime: 50597.300972
Maximum sum_exec_runtime: 22446.373810

vruntime
  ^
  |                                                                                                    1
  |                                                                 211  2    1    1   1    1
  |
  |                                                                1
  |
  |
  |                                                             1 1
  |                                                             3
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |                                                          1 2
  |                                                      1 12
  |                                                   222
  |
  |                                                  1
  |
  |                                                  1
  |                                            121 32
  |                                          211
  |                                          2
  |
  |                                        541
  |                                       2
  |                                     2 2
  | 312333985631 212 215  253314223   12
  |91
  +────────────────────────────────────────────────────────────────────────────────────────────────────→ sum_exec_runtime
```

Figure 9: log vruntime vs. sum_exec_runtime

Here is a representation of the changes from `c3.c` to my animation code powered by AI and the prompt “Show the differences between these two code segments using the formatting of a more reader friendly git diff”

```
Line 3:
+ #include <string.h>

Lines 8-12:
+ unsigned p;
+ char final_strings[4][2048];
+ for (int i = 0; i < 10; i++) {
+   memset(final_strings, 0, sizeof(final_strings));
+   system("top -b -n 1 | tail -n +8 | sort > out.top");

Lines 14-15:
- for (int i = 0; i < 7; i++) fgets(buffer, sizeof(buffer), fp);

Lines 17-19:
- sscanf(buffer, " %s %s %s %s %s %s %s %s %s %s %s %s\n",
-             pid, user, pr, ni, virt, res, shr, s, cpu, mem, time, command);
- printf("PID: %s\tTIME: %s\n", pid, time);

Lines 20-27:
+ sscanf(buffer, " %s %s %s %s %s %s %s %s %s %s %s %s %d\n", pid, user, pr,
+        ni, virt, res, shr, s, cpu, mem, time, command, &p);
+ char pid_buf[32];
+ sprintf(pid_buf, "%s, ", pid);
+ strcat(final_strings[p], pid_buf);
+ }
+ for (int p = 0; p < 4; p++) {
+   printf("Core %d: %s\n\n", p, final_strings[p]);

Lines 30-32:
+ system("sleep 1");
+ system("clear");
+ }
```