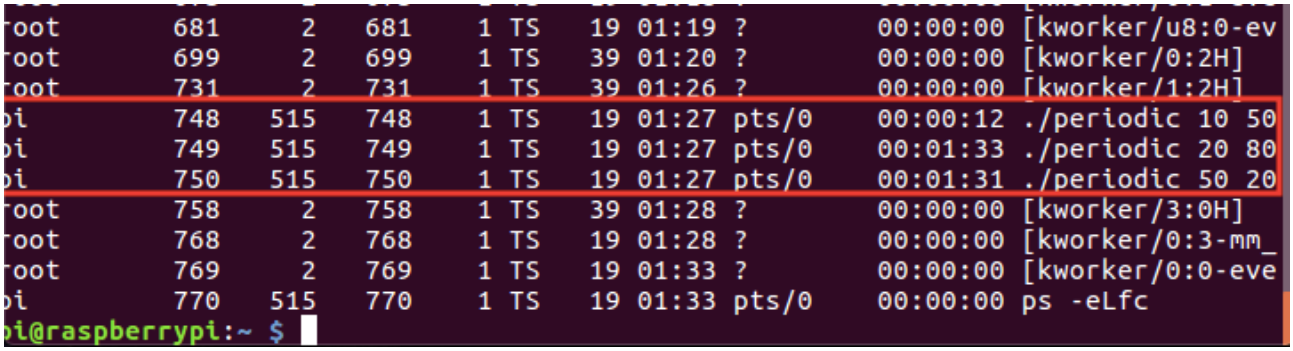
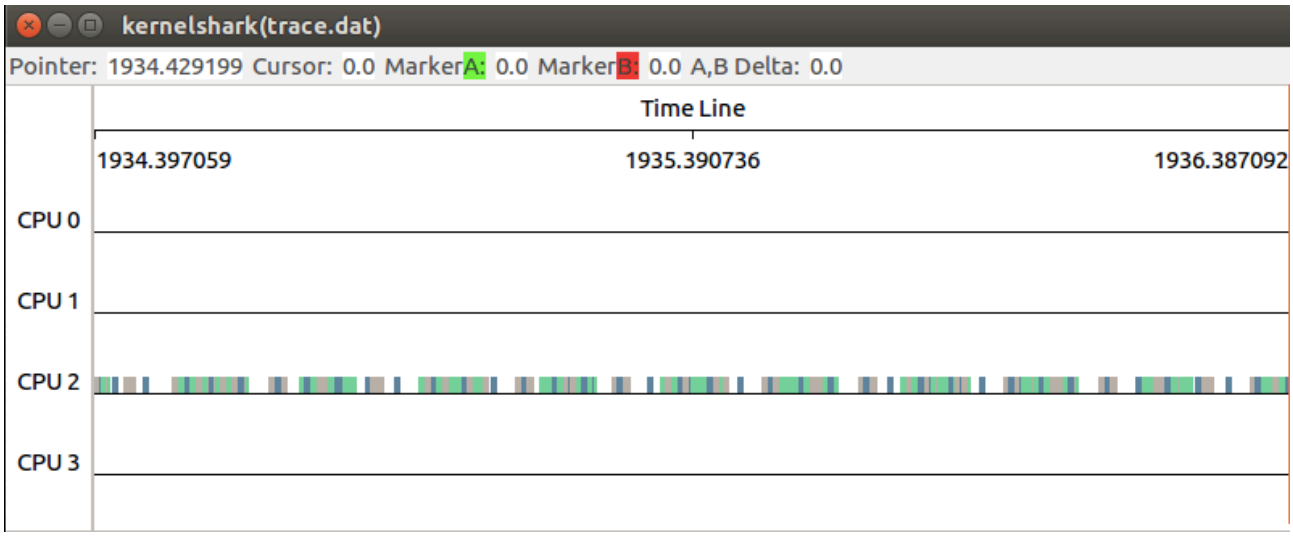
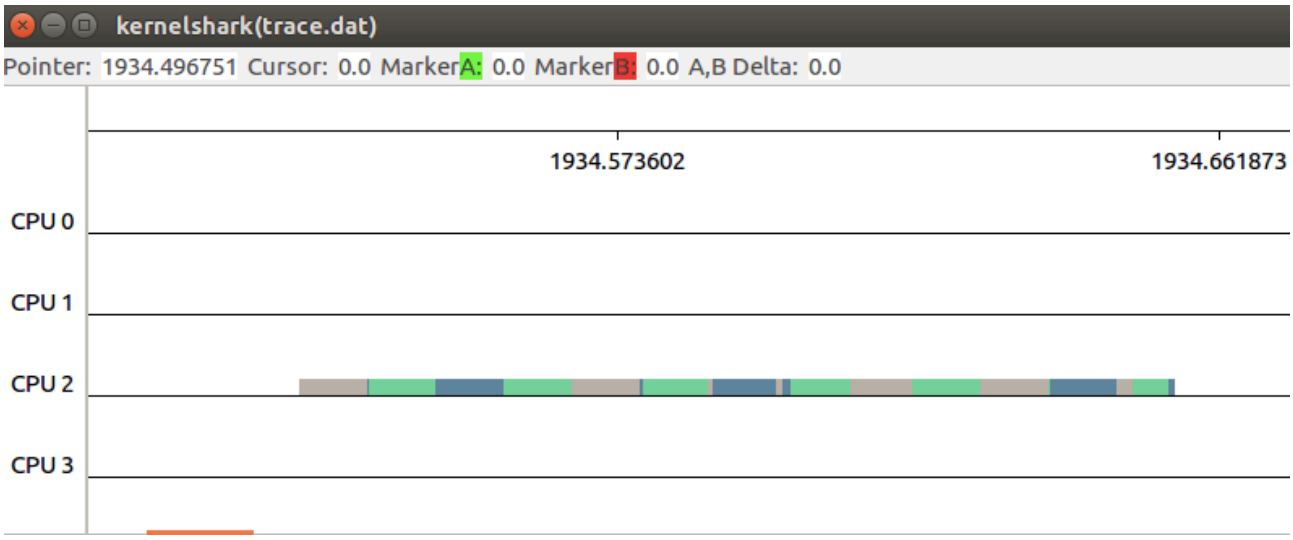
Final Report

1. Periodic program



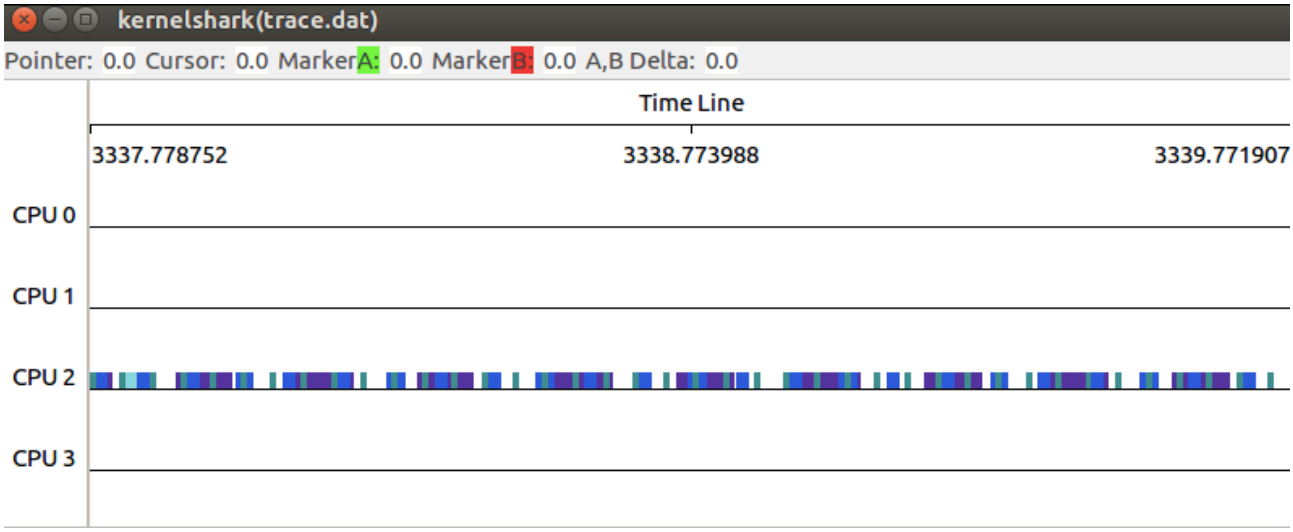


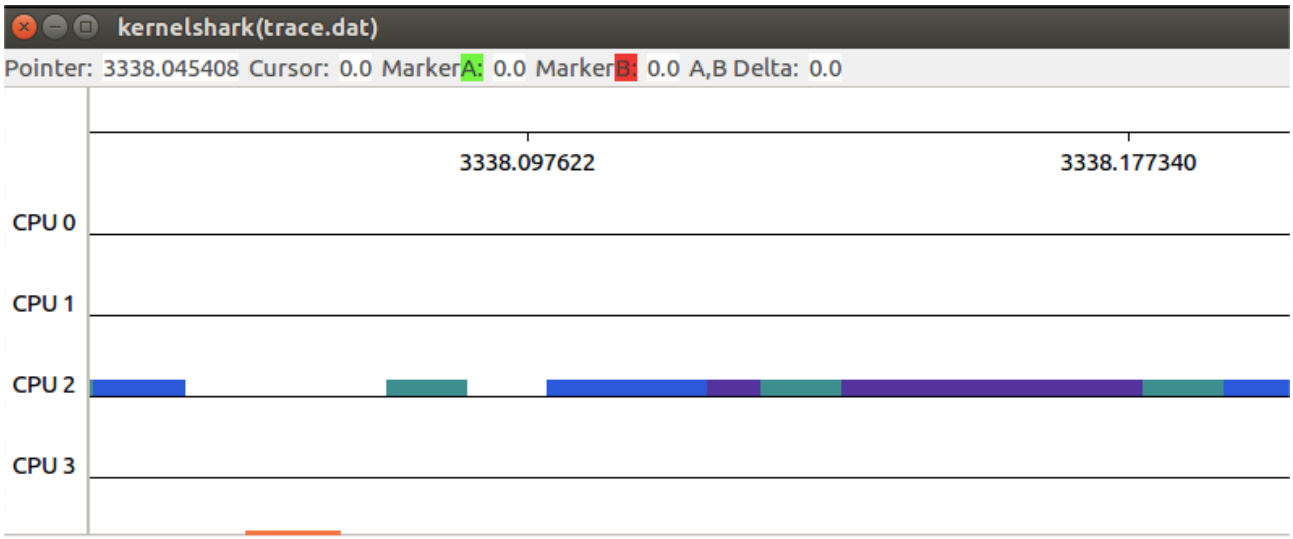


Task 1 (10, 50) in blue. Task 2 (20, 80) in grey, Task 3 (50, 200) in green.
We do not set any type of schedular in periodic.c Hence the tasks are run by the default linux scheduler that gives a fair chance to all the tasks. In the zoomed in screenshot, it can be easily seen that the tasks are given cpu time proportional to their utilisation. Hence Task 2 and Task 3 have almost same cpu time while task 1 has comparatively less cpu time.

2. SCHED_FIFO

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ chrt -p 963
pid 963's current scheduling policy: SCHED_FIFO
pid 963's current scheduling priority: 80
pi@raspberrypi:~ $ chrt -p 964
pid 964's current scheduling policy: SCHED_FIFO
pid 964's current scheduling priority: 79
pi@raspberrypi:~ $ chrt -p 965
pid 965's current scheduling policy: SCHED_FIFO
pid 965's current scheduling priority: 78
pi@raspberrypi:~ $
```





Task 1 (10, 50) in green Task 2 (20, 80) in blue, Task 3 (50, 200) in purple.
We can observe that the tasks are getting preempted by tasks with higher priority. It can be readily observed in the zoomed screenshot that Task 3 in purple is getting preempted by Task 1 in green even though it arrived after Task 3. Also Task 3 was not scheduled till Task 2 in blue was completed. Hence preemption according to the set priorities is observed. The scheduling policy is FIFO hence there is no preemption after a fixed amount of time. The task will run unless another high priority task is scheduled or the task itself completes.

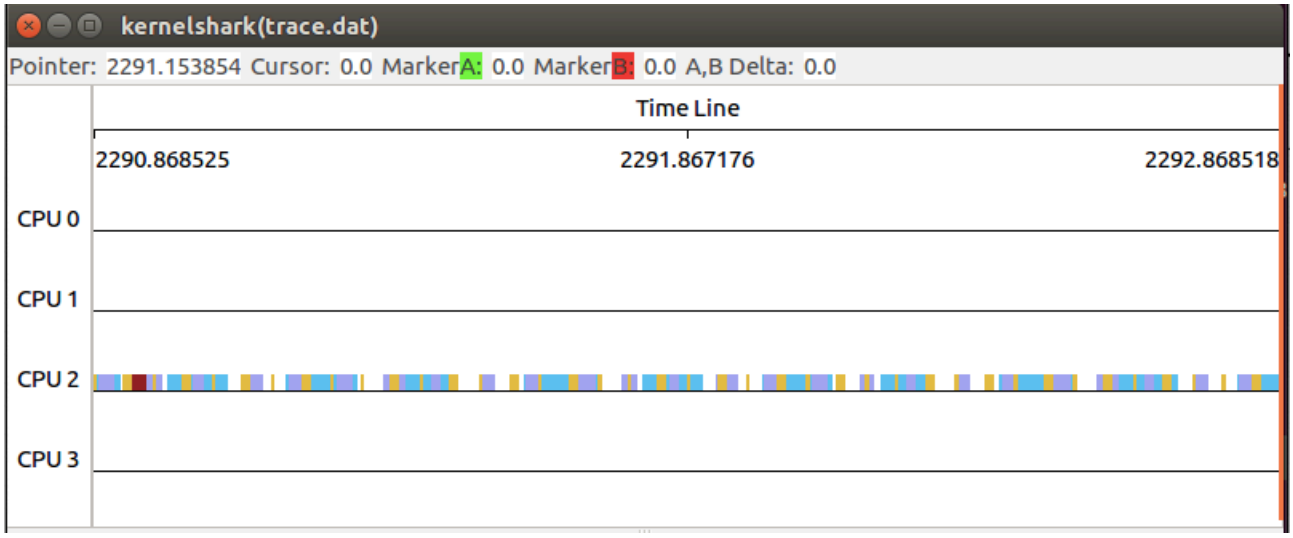
3. SCHED_DEADLINE and schedutil governor

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ ./periodic 10 50 2 &
[2] 753
pi@raspberrypi:~ $ PID: 753, C: 10, T: 50, CPUID: 2

pi@raspberrypi:~ $ ./periodic 20 80 2 &
[3] 754
pi@raspberrypi:~ $ PID: 754, C: 20, T: 80, CPUID: 2

pi@raspberrypi:~ $ ./periodic 50 200 2 &
[4] 755
pi@raspberrypi:~ $ PID: 755, C: 50, T: 200, CPUID: 2

pi@raspberrypi:~ $ sudo ./test_edf 753 10 50 2
PID: 753, C: 10, T: 50
pi@raspberrypi:~ $ sudo ./test_edf 754 20 80 2
PID: 754, C: 20, T: 80
pi@raspberrypi:~ $ sudo ./test_edf 755 50 200 2
PID: 755, C: 50, T: 200
pi@raspberrypi:~ $
```

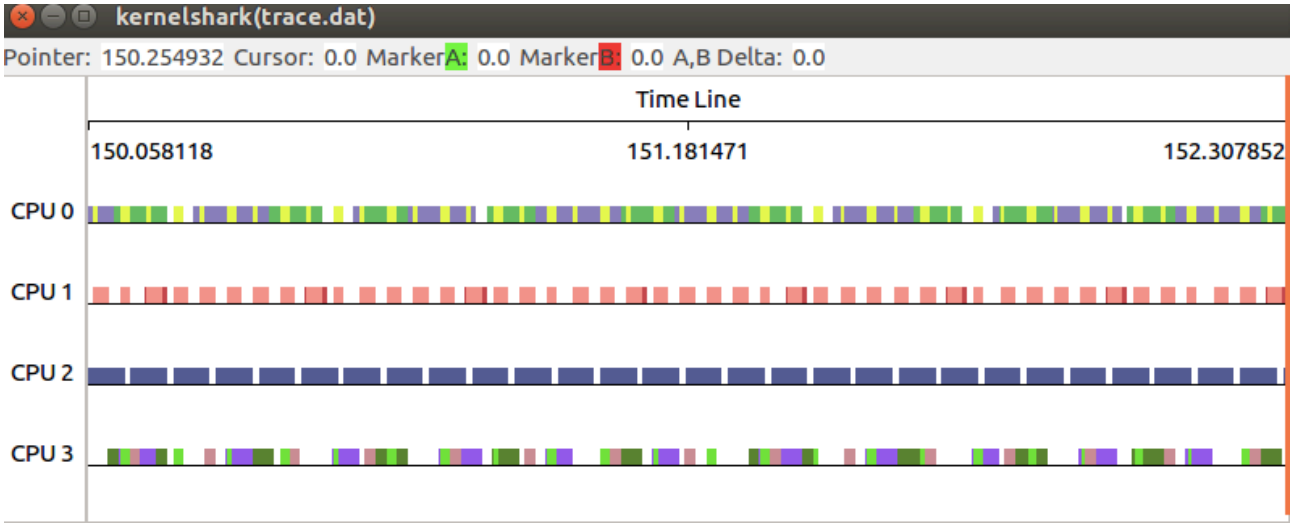


Task 1 (10, 50) in orange Task 2 (20, 80) in purple, Task 3 (50, 200) in blue.
As compared to part 1 and part 2 screenshots, this looks to have a greater utilisation. This is due to the governor which is set to schedutil scales the frequency for energy saving. However in first 2 parts, frequency was fixed to be 1.5 GHz. The utilisation is not near 100% as schedutil scales frequency conservatively to avoid system crash.

4. Multicore periodic using userspace governor

```
pi@raspberrypi: ~/multicore_periodic
pi@raspberrypi:~/multicore_periodic $ sudo sh -c "echo userspace > /sys/devices/
system/cpu/cpu0/cpufreq/scaling_governor"
pi@raspberrypi:~/multicore_periodic $ sudo sh -c "echo 1500000 > /sys/devices/sy
stem/cpu/cpu0/cpufreq/scaling_setspeed"
pi@raspberrypi:~/multicore_periodic $ sudo ./multicore_periodic taskset2.csv &
[1] 537
pi@raspberrypi:~/multicore_periodic $ PID: 539 C:10 T:50 CPUID:0
PID: 540 C:20 T:50 CPUID:1
PID: 541 C:50 T:80 CPUID:2
PID: 542 C:10 T:100 CPUID:3
Userspace governor detected. Freq set to 1100000
PID: 545 C:50 T:240 CPUID:0
PID: 547 C:10 T:300 CPUID:1
PID: 548 C:80 T:300 CPUID:0
PID: 543 C:15 T:150 CPUID:3
PID: 546 C:30 T:240 CPUID:3
PID: 544 C:30 T:200 CPUID:3

pi@raspberrypi:~/multicore_periodic $
```



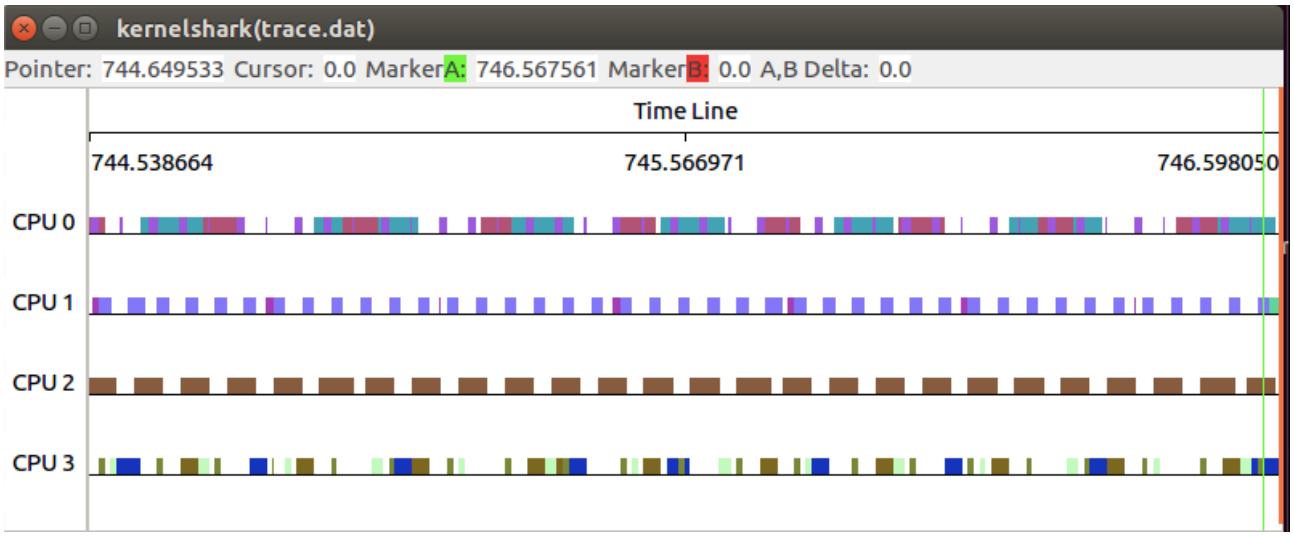
The tasks are scheduled using best fit increasing heuristic so that each core is uniformly utilised. Then the frequency is scaled such that core with maximum utilisation still maintains schedulability.

We can easily observe C values increased proportionally to the reduced frequency from 1.5 GHz to 1.1 GHz. For cpu 2, only one task is assigned with C = 50 and T = 80 with a utilisation of 0.625. However the in trace file, cpu 2 utilisation is as high as 0.85 which goes on to show that the frequency of the cores has indeed reduced.

5. Multicore periodic using schedutil governor

```
pi@raspberrypi: ~/multicore_periodic
pi@raspberrypi:~/multicore_periodic $ sudo sh -c "echo schedutil > /sys/devices/
system/cpu/cpu0/cpufreq/scaling_governor"
pi@raspberrypi:~/multicore_periodic $ sudo ./multicore_periodic taskset2.csv &
[1] 553
pi@raspberrypi:~/multicore_periodic $ PID: 555 C:10 T:50 CPUID:0
PID: 556 C:20 T:50 CPUID:1
PID: 559 C:15 T:150 CPUID:3
PID: 562 C:30 T:240 CPUID:3
PID: 563 C:10 T:300 CPUID:1
PID: 561 C:50 T:240 CPUID:0
PID: 557 C:50 T:80 CPUID:2
PID: 558 C:10 T:100 CPUID:3
PID: 564 C:80 T:300 CPUID:0
PID: 560 C:30 T:200 CPUID:3

pi@raspberrypi:~/multicore_periodic $
```



Here the governor is set to schedutil. Hence the user level application does not set the scaling frequency. It just schedules the tasks using best first increasing and allows the kernel to manipulate the frequency.

As before we can see only one task is assigned to core 2. The utilisation looks a bit higher than 0.625 at 1.5 GHz but not as high as it was for 1.1 GHz. I believe the frequency is scaled but conservatively to avoid a system crash.