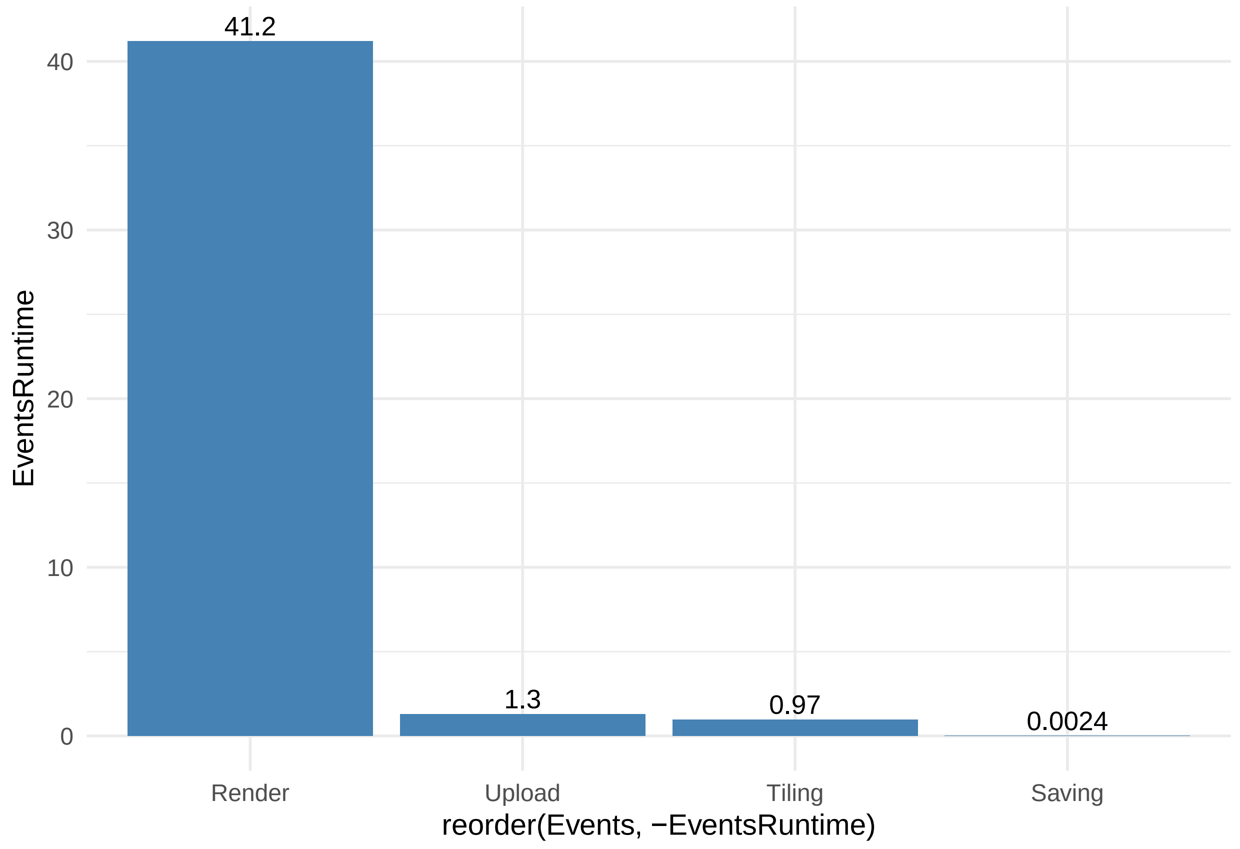
Analysis Results

Angelos Nikolas

Objective 1 Which event types dominate task runtimes?

By categorizing the events and checking run times the rendering event type dominate the run times. Specifically rendering accounts for approximately 96.7 of the whole process which lasts 42.6 seconds in average for each task. The graph below visually represents the run time difference for each type:

```
Barchart1
```



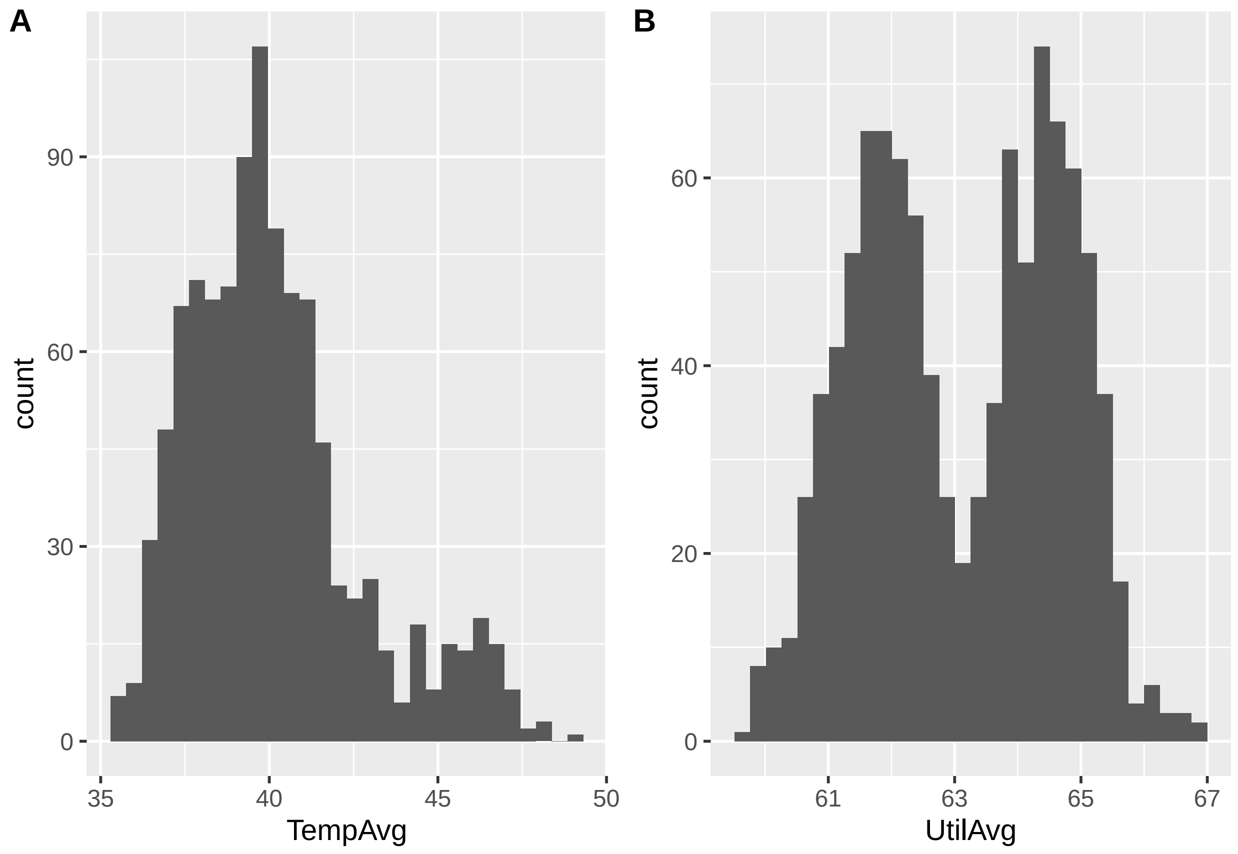
Objective 2 What is the interplay between GPU temperature and performance?

By investigating the GPU's temperature it is noticeable that when the temperature is rising so its the utilization of the GPU's and vice versa.

```
summary(GpuMeans)
```

```
##    gpuSerial           TempAvg         UtilAvg
##  Min.   :3.201e+11   Min.   :35.37   Min.   :59.71
##  1st Qu.:3.236e+11   1st Qu.:38.28   1st Qu.:61.77
##  Median :3.236e+11   Median :39.73   Median :63.14
##  Mean   :3.240e+11   Mean   :40.12   Mean   :63.12
##  3rd Qu.:3.250e+11   3rd Qu.:41.22   3rd Qu.:64.49
##  Max.   :3.252e+11   Max.   :48.93   Max.   :66.95
##                                      NA's   :4
```

```
TempHist
```



Objective 3 What is the interplay between increased power draw and render time?

As it was concluded from the 2nd Objective the rendering is the event that dominates run times meaning that every change in the power should account for the rendering needs of the image. By summarizing the data it is noticeable that the GPU's show similar trends regarding power distribution while undertaking a task, the same its true for all tasks undertaken by a single GPU with the difference being the time it occurs. The average highest power draw was 106 Watts and the lowest 80 which is not a significant difference in term of power distribution the the data frame below contains all calculated average power values for all 1024 GPUs. A numerical summary is included to depict how the power distribution vary.

```
head(HighPowerDraw)
```

```
## # A tibble: 6 x 2
##   hostname                                PowerDrawMeans
##   <chr>                                            <dbl>
## 1 04dc4e9647154250beeee51b866b0715000000           95.9
## 2 04dc4e9647154250beeee51b866b0715000001           91.8
## 3 04dc4e9647154250beeee51b866b0715000002           82.5
## 4 04dc4e9647154250beeee51b866b0715000003           86.6
## 5 04dc4e9647154250beeee51b866b0715000004           94.3
## 6 04dc4e9647154250beeee51b866b0715000005           86.2
```

```
summary(HighPowerDraw)
```

```
##    hostname         PowerDrawMeans
##  Length:1024        Min.   : 80.51
##  Class :character   1st Qu.: 86.55
##  Mode  :character   Median : 88.93
##                     Mean   : 89.20
##                     3rd Qu.: 91.63
##                     Max.   :106.24
##                     NA's   :25
```

By filtering out the idle time that the GPU's doesn't seem to be rendering we can conclude that for 74.9% of the process of rendering each GPU showed an ascending trend reaching a maximum point before the rendering was concluded. This indicates that for optimal rendering time high power distribution is significant.

Objective 4 Can we quantify the variation in computation requirements for particular tiles?

By tracing the task ID and matching time stamps assigned to particular tiles it is possible to extract the computational requirements for this particular tile. Also all tasks made for the rendering of the tile. An example is given below.

```
summary(GpuComp)
```

```
##    timestamp                     hostname           gpuSerial
##  Min.   :2018-11-08 08:06:40   Length:15          Min.   :3.251e+11
##  1st Qu.:2018-11-08 08:06:47   Class :character   1st Qu.:3.251e+11
##  Median :2018-11-08 08:06:55   Mode  :character   Median :3.251e+11
##  Mean   :2018-11-08 08:06:54                      Mean   :3.251e+11
##  3rd Qu.:2018-11-08 08:07:02                      3rd Qu.:3.251e+11
##  Max.   :2018-11-08 08:07:09                      Max.   :3.251e+11
##    gpuUUID          powerDrawWatt       gpuTempC     gpuUtilPerc
##  Length:15          Min.   : 32.60   Min.   :45.0   Min.   : 0.00
##  Class :character   1st Qu.: 67.16   1st Qu.:46.0   1st Qu.:30.50
##  Mode  :character   Median : 98.96   Median :48.0   Median :87.00
##                     Mean   : 85.76   Mean   :47.2   Mean   :62.27
##                     3rd Qu.:102.48   3rd Qu.:48.0   3rd Qu.:88.00
##                     Max.   :119.03   Max.   :48.0   Max.   :89.00
##  gpuMemUtilPerc
##  Min.   : 0.0
##  1st Qu.:16.0
##  Median :39.0
##  Mean   :29.2
##  3rd Qu.:41.0
##  Max.   :47.0
```

```
GpuComp
```

```
## # A tibble: 15 x 8
##    timestamp           hostname     gpuSerial gpuUUID     powerDrawWatt gpuTempC
##    <dttm>              <chr>            <dbl> <chr>               <dbl>    <int>
##  1 2018-11-08 08:06:46 0745914f4de~   3.25e11 GPU-f3d43c~          79.8       46
##  2 2018-11-08 08:06:52 0745914f4de~   3.25e11 GPU-f3d43c~         105.        48
##  3 2018-11-08 08:06:57 0745914f4de~   3.25e11 GPU-f3d43c~         118.        48
##  4 2018-11-08 08:07:01 0745914f4de~   3.25e11 GPU-f3d43c~          99.9       48
##  5 2018-11-08 08:07:09 0745914f4de~   3.25e11 GPU-f3d43c~          54.5       46
##  6 2018-11-08 08:06:48 0745914f4de~   3.25e11 GPU-f3d43c~         103.        48
##  7 2018-11-08 08:06:55 0745914f4de~   3.25e11 GPU-f3d43c~         101.        48
##  8 2018-11-08 08:06:59 0745914f4de~   3.25e11 GPU-f3d43c~          93.4       48
##  9 2018-11-08 08:06:42 0745914f4de~   3.25e11 GPU-f3d43c~          32.6       45
## 10 2018-11-08 08:07:07 0745914f4de~   3.25e11 GPU-f3d43c~          98.6       48
## 11 2018-11-08 08:06:40 0745914f4de~   3.25e11 GPU-f3d43c~          32.8       46
## 12 2018-11-08 08:06:44 0745914f4de~   3.25e11 GPU-f3d43c~          47.4       45
## 13 2018-11-08 08:06:50 0745914f4de~   3.25e11 GPU-f3d43c~          99.0       48
## 14 2018-11-08 08:07:03 0745914f4de~   3.25e11 GPU-f3d43c~         102.        48
## 15 2018-11-08 08:07:05 0745914f4de~   3.25e11 GPU-f3d43c~         119.        48
## # ... with 2 more variables: gpuUtilPerc <int>, gpuMemUtilPerc <int>
```

Using the task ID it is possible to repeat this process for every tile used in the image and make comparisons if it's needed.

Objective 5 Can we identify GPU cards (based on their serial numbers) whose performance differs to other cards? (i.e., perpetually slow cards).

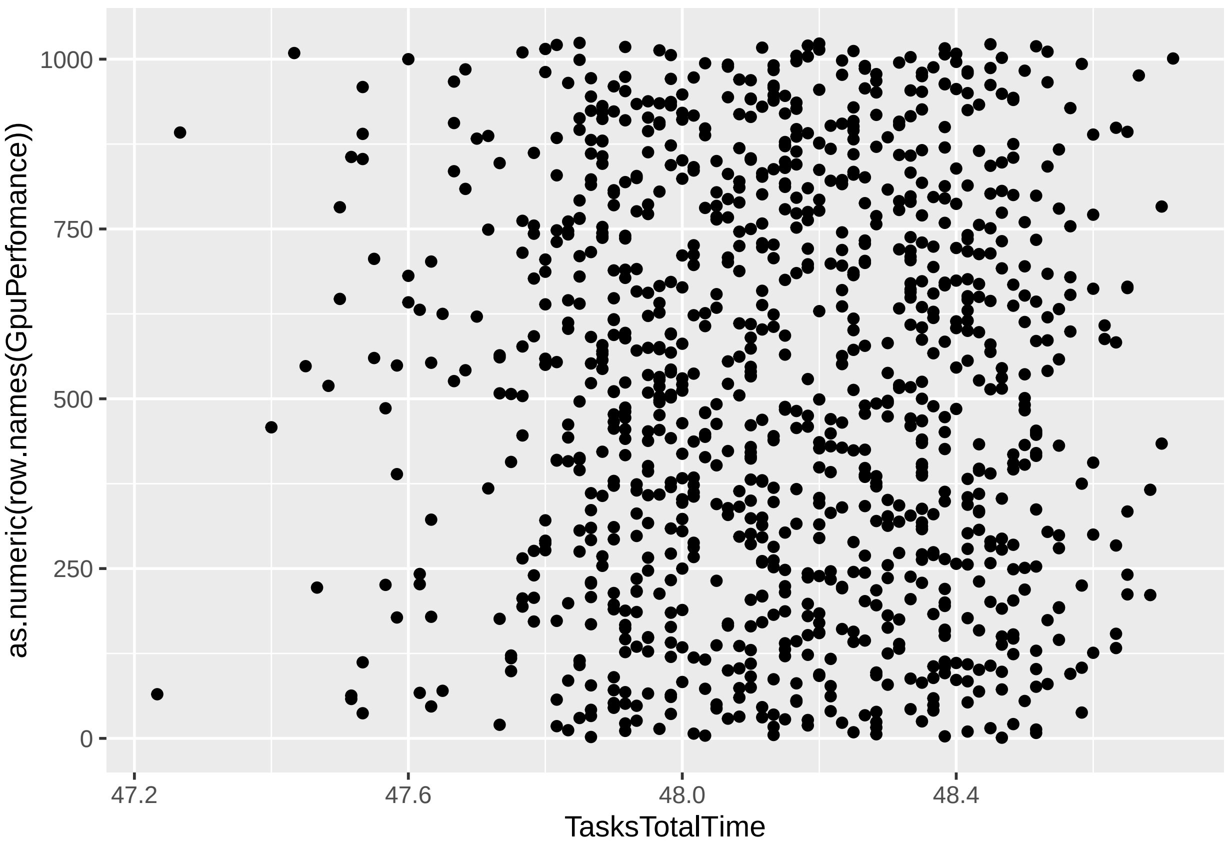
By conducting the process detailed in the main report it is possible to store all GPUs by serial number with the total time it took each one to finish the task assigned to them in minutes. Below is the first 6 entries of a new data frame holding all 1024 GPUs total task run time.

```
head(GpuPerfomance)
```

```
## # A tibble: 6 x 2
##      gpuSerial TasksTotalTime
##          <dbl>          <dbl>
## 1 323217055910           48.5
## 2 323617020295           47.9
## 3 323217056562           48.4
## 4 325217085931           48.0
## 5 323217056464           48.1
## 6 323617020227           48.3
```

A scatter plot is created to depict the time allocation for each card:

```
Scatter1
```



It is important to mention that the task allocation in each card is not equally made four specific cards had almost double the tasks allocated to them. Despite that the time it took to finish their tasks is similar to the other cards indicating that these four cards are much faster or configured in way to produce similar results. This can be seen in the task allocation data set that was made to count how many tasks each card was responsible to execute.

The average task time can be seen below for all GPUs combined:

```
summary(GpuPerfomance)
```

```
##    gpuSerial         TasksTotalTime
##  Min.   :3.201e+11   Min.   :47.23
##  1st Qu.:3.236e+11   1st Qu.:47.95
##  Median :3.236e+11   Median :48.15
##  Mean   :3.240e+11   Mean   :48.14
##  3rd Qu.:3.250e+11   3rd Qu.:48.35
##  Max.   :3.252e+11   Max.   :48.72
```

Objective 6 What can we learn about the efficiency of the task scheduling process?

For this Objective a general understanding of the whole process is required, the scheduling process has been derived through the EDA process. 1024 graphics cards have been given the task to render a terrapixel visualization of the city of Newcastle, as mentioned above the tasks allocated are no equal across all cards. Although, the performance is very similar to one another each task contain 5 events that need to be completed in order for the tile to be rendered. Each task represents a tile on the image totaling of 65793 tiles. This can be derived quite easily from the EDA process although an interesting find was that each GPU seems to render specific areas on the visualization this became obvious by examining the x and y columns on the tasks data set that refers to each tile coordinates. An assumption made was that it could be much easier to notice a render issue in specific areas of the image and the whole rendering could occur smoothly if the work is allocated in hierarchical manner.

```
head(TileTask)
```

```
## # A tibble: 6 x 4
##   taskId                                    x     y level
##   <chr>                                 <int> <int> <int>
## 1 43a4aee3-a4af-43e5-8a0b-69f73ace4fa1   197     8    12
## 2 43a4aee3-a4af-43e5-8a0b-69f73ace4fa1   197     8    12
## 3 8cdaa058-3c28-45ca-bf2b-558955801c6d     9     3     8
## 4 8cdaa058-3c28-45ca-bf2b-558955801c6d     9     3     8
## 5 06ff2ffa-c05d-4b37-bbd2-4a8a0a578ce0    95   228    12
## 6 06ff2ffa-c05d-4b37-bbd2-4a8a0a578ce0    95   228    12
```

In this data frame every event for every tile rendered is stored the coordinates indicate an ordered trend suggesting that this specific GPU render tiles connected with one another.