Performance Evaluation of Terapixel Rendering in Cloud (Super) Computing

Terapixel images are the images composed of 1 trillion pixels and they can be used to present information to stakeholders intuitively, however, it requires a lot of compute power. So, it is advantageous to analyze the data collected from cloud super computing infrastructure to optimize the process and reduce the cost incurred.

Business Understanding

- Which event types are taking most of the execution time?

  > By analyzing dominant events we can manage resources efficiently and hence reduce the cost of operation.

- How GPU temperature and Performance are related?

  > This will allow us to switch resource efficiently, "GPU throttling" mainly happens because of inefficient heat management. So, if we observe some GPUs starting to throttle because of increase in temperature we can switch them with GPUs of same throughput but at operating at lower temperatures.

- Is there any impact of increased power draw on render time?

  > With this we can quantify whether the increase in power draw has any impact on render time. If yes, then how much improvement are we getting? if no, what can be done to reduce the power consumption.

- Is there any performance difference in GPU cards?

  > If different GPUs are performing at different throughputs we may well be able to allocate high end GPUs for more computation intensive tasks.

- Is there a specific pattern in which tiles/pixels are consuming more or less power?

  > If we can figure out some specific regions on the terapixel image are taking more compute power, we can allocate flagship GPUs for those specific pixels and mid to low end GPUs for rest of the pixels. This will improve the rendering of the image and also better resource management.

Data Description

- **application-checkpoint.csv:** This file contains details of what type of events are happening at what timestamp and their hostname, taskId, jobId.

- **timestamp:** The exact time at which the event started or stopped.
- **hostname:** There are 1024 unique hostnames, each GPU can be considered as a host.
- **eventName:** There are 5 events, Tiling, Render, Saving Config, Uploading, and TotalRender.
    - **TotalRender:** It is the complete task.
    - **Render:** It is when tiles of the image is being rendered.
    - **Tiling:** It the when the post-processing of rendered tile is done.
    - **Saving Config:** Each task has some specific configurations, which is being saved.
    - **Uploading:** Output of post-processed tiles are uploaded to Azure Blob Storage.
- **eventType:** START or STOP, this shows whether the event is getting started or being terminated.
- **jobId:** There are 3 jobIds for Azure batch job. For level 12, level 8, and level 4.
- **taskId:** Id of the Azure batch task. Theres unique ID for each set of all 5 eventNames along with their start and stop timestamps. For example, if there are n rows in the application-checkpoint.csv there will be n/10 unique taskIds. 5 events and their start and stop rows = 5*2 = 10

- **gpu.csv:** This file consists of gpu conditions at a particular instance of time.
    - **gpuSerial:** There are 1024 GPUs and each has a unique serial number.
    - **gpuUUID:** The unique system id assigned by the Azure system to the GPU unit.
    - **powerDrawWatt:** Power drawn by the GPU in Watt.
    - **gpuTempC:** Temperature of the GPU in Celsius.
    - **gpuUtilPerc:** Percentage of cores of GPU utilized.
    - **gpuMemUtilPerc:** Percentage of memory of GPU utilized.

- **task-x-y.csv**: This file has x and y co-ordinates of the image tiles being rendered.
    - **x:** x co-ordinates of the image tiles.
    - **y:** y co-ordinates of the image tiles.
    - **level:** The visualization created is a zoomable "google maps style" map. In total we create 12 levels. Level 1 is zoomed right out and level 12 is zoomed right in. You will only see levels 4, 8 and 12 in the data as the intermediate level are derived in the tiling process.

Data Processing

- Merging 3 raw data files into one.
    - First left join was performed on task-x-y.csv with application-checkpoint.csv on 'jobId' and 'taskId'.
    - Then inner join on merged data from previous step and gpu.csv using 'timestamp'.

- Calculation of time duration required by each event.

    - Extracted only time component from the timestamp column, as whole data was collected on the same date i.e., 2018-11-08.
    - Now grouped final merged data (from previous step) on eventName and eventType.
    - Made different dataframes for each event with their start and stop times separately. Hence, now we have 10 separate dataframes. 5x2=10; 5: events(*TotalRender, Render, Tiling, Saving Config, Uploading*), 2: eventType(*Start and Stop*)
    - Merged start and stop dataframes of respective events on taskId. Now we have 5 dataframes, 1 for each event.
    - Subtracted start time from stop time and stored it in separate column named "duration".
    - Finally, stacked all the 5 event's dataframes into 1.

Important Assumption(s) and Consideration(s)

- In the market when we look for which GPU or processor is better than the other, to measure their performance we look into their peak power draw. More the power draw more will be the performance. This fact is taken into consideration while comparing the performance of the GPU with its other conditions like GPU temperature, GPU memory utilization, GPU utilization.
- We have merged the Application checkpoint and GPU data based on timestamp which can not be done in all the cases however, for the analysis we are carrying out, we are taking the average of most of the features so that we can smoothen the data and don't let the errors affect our results.

Data Analysis

In [ ]:

```
# !pip install sweetviz # installed sweetviz for data visualization...
```

In [2]:

```
# Importing relevant packages...

import IPython
import numpy as np
import pandas as pd
import seaborn as sns
from dateutil import parser
import dask.dataframe as dd
import matplotlib.pyplot as plt
from scipy.sparse import coo_matrix
from datetime import datetime as dt, timedelta, date
# import sweetviz as sv
```

In [19]:

```
# Loading raw as well as processed data...

app_check = pd.read_csv("../data/application-checkpoints.csv")
gpu = pd.read_csv("../data/gpu.csv")
task_xy = pd.read_csv("../data/task-x-y.csv")
merged_data = pd.read_csv("../data/merged_data.csv")
event_total_duration_df = pd.read_csv("../data/event-total-duration.csv")
```

In [ ]:

```
# gpu_report = sv.analyze(gpu, target_feat = \"powerDrawWatt\") # created html page for dat
# gpu_report.show_html(filepath = \"../reports/gpu_report.html\") # saved it into reports d
```

In [4]:

```
print(f"Application Checkpoint has {app_check.shape[0]} rows and {app_check.shape[1]} colum
```

```
Application Checkpoint has 660400 rows and 6 columns
```

In [63]:

```
app_check.head() # checking head of application checkpoint dataset...
```

Out[63]:

	timestamp	hostname	eventName	eventType	
0	2018-11-08T07:41:55.921Z	0d56a730076643d585f77e00d2d8521a00000N	Tiling	STOP	1024-7e026 5fd0-4 abd61f74
1	2018-11-08T07:42:29.842Z	0d56a730076643d585f77e00d2d8521a00000N	Saving Config	START	1024-7e026 5fd0-4 abd61f74
2	2018-11-08T07:42:29.845Z	0d56a730076643d585f77e00d2d8521a00000N	Saving Config	STOP	1024-7e026 5fd0-4 abd61f74
3	2018-11-08T07:42:29.845Z	0d56a730076643d585f77e00d2d8521a00000N	Render	START	1024-7e026 5fd0-4 abd61f74
4	2018-11-08T07:43:13.957Z	0d56a730076643d585f77e00d2d8521a00000N	TotalRender	STOP	1024-7e026 5fd0-4 abd61f74

In [5]:

```
print(f"GPU has {gpu.shape[0]} rows and {gpu.shape[1]} columns")
```

```
GPU has 1543681 rows and 8 columns
```

In [ ]:

```
gpu.head() # checking head of gpu dataset...
```

Out[5]:

	timestamp	hostname	gpuSerial	gpuUUID	pc
0	2018-11-08T08:27:10.314Z	8b6a0eebc87b4cb2b0539e81075191b900001C	323217055910	GPU-1d1602dc-f615-a7c7-ab53-fb4a7a479534	
1	2018-11-08T08:27:10.192Z	d8241877cd994572b46c861e5d144c85000000	323617020295	GPU-04a2dea7-f4f1-12d0-b94d-996446746e6f	
2	2018-11-08T08:27:10.842Z	db871cd77a544e13bc791a64a0c8ed50000006	323217056562	GPU-f4597939-a0b4-e78a-2436-12dbab9a350f	
3	2018-11-08T08:27:10.424Z	b9a1fa7ae2f74eb68f25f607980f97d7000010	325217085931	GPU-ad773c69-c386-a4be-b214-1ea4fc6045df	
4	2018-11-08T08:27:10.937Z	db871cd77a544e13bc791a64a0c8ed50000003	323217056464	GPU-2d4eed64-4ca8-f12c-24bc-28f036493ea2	

In [17]:

```
gpu.describe()
```

Out[17]:

	powerDrawWatt	gpuTempC	gpuUtilPerc	gpuMemUtilPerc
count	1.543681e+06	1.543681e+06	1.543681e+06	1.543681e+06
mean	8.919838e+01	4.007560e+01	6.305820e+01	3.341359e+01
std	3.975742e+01	3.800243e+00	4.144816e+01	2.300107e+01
min	2.255000e+01	2.600000e+01	0.000000e+00	0.000000e+00
25%	4.499000e+01	3.800000e+01	0.000000e+00	0.000000e+00
50%	9.659000e+01	4.000000e+01	8.900000e+01	4.300000e+01
75%	1.213400e+02	4.200000e+01	9.200000e+01	5.100000e+01
max	1.970100e+02	5.500000e+01	1.000000e+02	8.300000e+01

In [6]:

```
print(f"task-x-y has {task_xy.shape[0]} rows and {task_xy.shape[1]} columns")
```

```
task-x-y has 65793 rows and 5 columns
```

In [ ]:

```
task_xy.head() # checking head of task-x-y dataset...
```

Out[7]:

	taskId	jobId	x	y	level
0	00004e77-304c-4fbd-88a1-1346ef947567	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	116	178	12
1	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12
2	0003c380-4db9-49fb-8e1c-6f8ae466ad85	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	86	12
3	000993b6-fc88-489d-a4ca-0a44fd800bd3	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	235	11	12
4	000b158b-0ba3-4dca-bf5b-1b3bd5c28207	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	171	53	12

In [7]:

```
del merged_data['Unnamed: 0'] # deleting column "Unnamed: 0"...
print(f"Merged Data has {merged_data.shape[0]} rows and {merged_data.shape[1]} columns")
```

```
Merged Data has 86633 rows and 16 columns
```

In [67]:

```
merged_data.head() # checking head of merged dataset...
```

Out[67]:

	taskId	jobId	eventName	hostname	
0	00107991-1ad1-42c8-80b7-1c2dea75a1d5	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	Render	2ecb9d8d51bc457aac88073f6da05461000001	08:09:01
1	0010cbd2-8f82-4eb7-b374-f349c6616d27	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	Render	4a79b6d2616049edbf06c6aa58ab426a00000P	07:50:28
2	0010cbd2-8f82-4eb7-b374-f349c6616d27	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	Render	4a79b6d2616049edbf06c6aa58ab426a00000P	07:50:28
3	65d00e61-521b-4a3b-94e6-86a4c3965092	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	Render	8b6a0eebc87b4cb2b0539e81075191b900000O	08:26:28
4	65d00e61-521b-4a3b-94e6-86a4c3965092	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	Render	a77ef58b13ad4c01b769dac8409af3f8000013	08:26:28

In [3]:

```
# Calculating mean of gpu temperature, gpu utilization percentage, power draw, and duration
# by grouping on gpu serial and eventName, excluding "TotalRender"...

avg_data = merged_data[merged_data['eventName'] != 'TotalRender'][['gpuSerial', 'eventName'
avg_data.index.names = ['gpuSerial', 'eventName']
avg_data.reset_index(inplace=True)
```

In [58]:

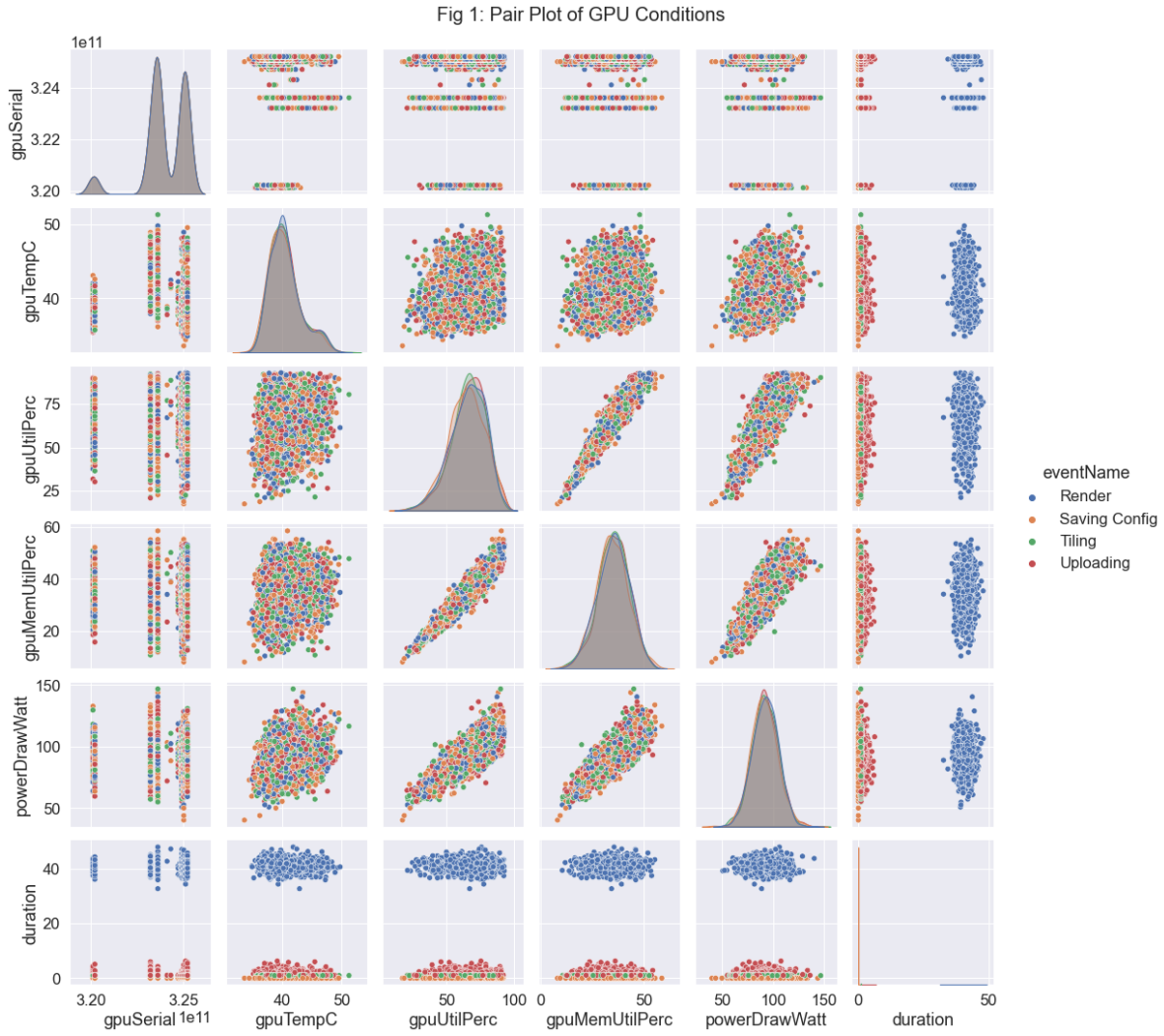
```
avg_data.head() # checking head of averaged dataset...
```

Out[58]:

	gpuSerial	eventName	gpuTempC	gpuUtilPerc	gpuMemUtilPerc	powerDrawWatt	durati
0	320118118607	Render	36.916667	88.583333	48.750000	105.404167	39.3665
1	320118118607	Saving Config	36.518519	70.185185	38.333333	84.607037	0.0027
2	320118118607	Tiling	36.440000	83.040000	44.880000	103.791600	0.9370
3	320118118607	Uploading	35.428571	31.785714	18.571429	62.997857	1.0501
4	320118118641	Render	39.375000	80.500000	44.312500	100.221250	42.5243

In [32]:

```
# Pairplot of averaged data to see relationship between various GPU conditions.
f = sns.pairplot(avg_data, hue="eventName")
sns.set(font_scale = 2) # settng text font scale...
f.fig.suptitle("Fig 1: Pair Plot of GPU Conditions", fontsize = 20, y=1.01)
plt.show()
```



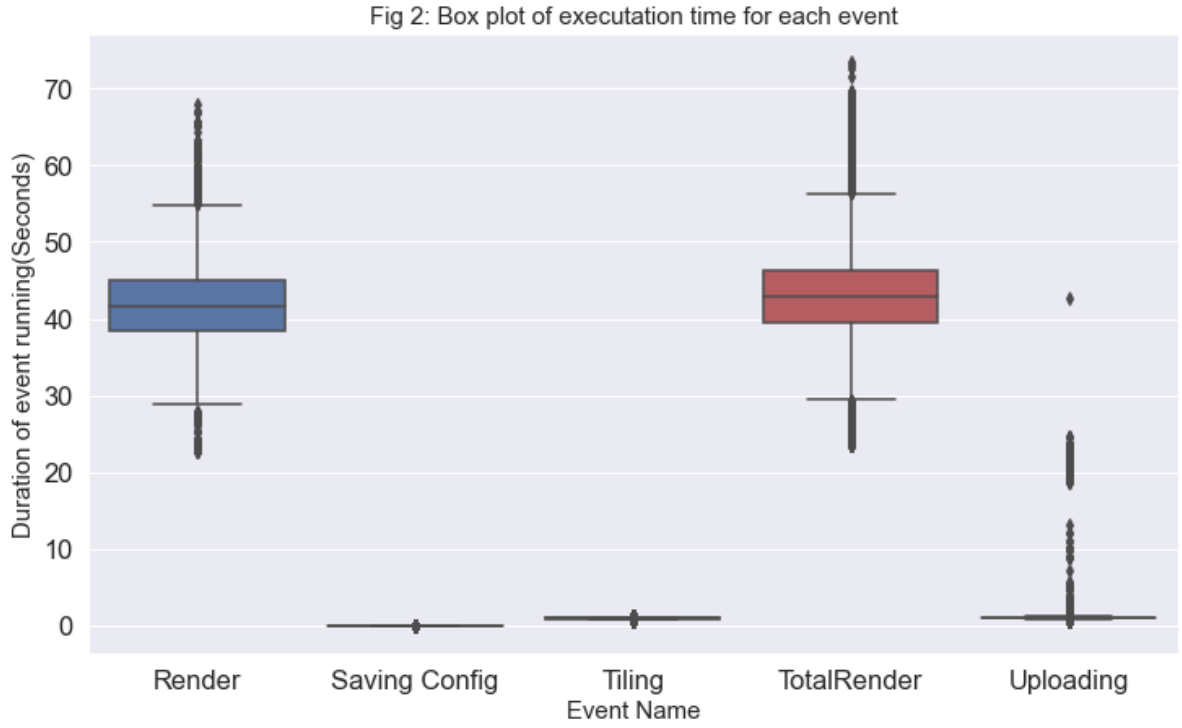
From Fig 1, we can get following insights:

- Duration of Render event is significantly more than other events.
- GPU temperature does not have any specific relation with Power draw, GPU memory utilization, GPU Utilization percentage.
- GPU Utilization percentage is very highly positively co-related to power draw and GPU memory utilization.
- GPU memory utilization has positive high co-relation with power drawn by the GPU.

Events that are dominating the runtime.

In [46]:

```
# Box plot for Events to analyse which event is taking most of the computation time...
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
sns.boxplot(x="eventName", y="duration", data=merged_data)
sns.set(font_scale = 1.5) # Text font scale...
plt.xlabel('Event Name', fontsize = 15)
plt.ylabel('Duration of event running(Seconds)', fontsize = 15)
plt.title("Fig 2: Box plot of executation time for each event", fontsize = 15)
plt.show()
```



In [62]:

```
del event_total_duration_df['Unnamed: 0'] # Deleting column with name "Unnamed: 0"
event_total_duration_df.head()
```

Out[62]:

	eventName	totalDuration
0	Tiling	17.986
1	Saving Config	0.046
2	Render	761.960
3	TotalRender	787.769
4	Uploading	25.755

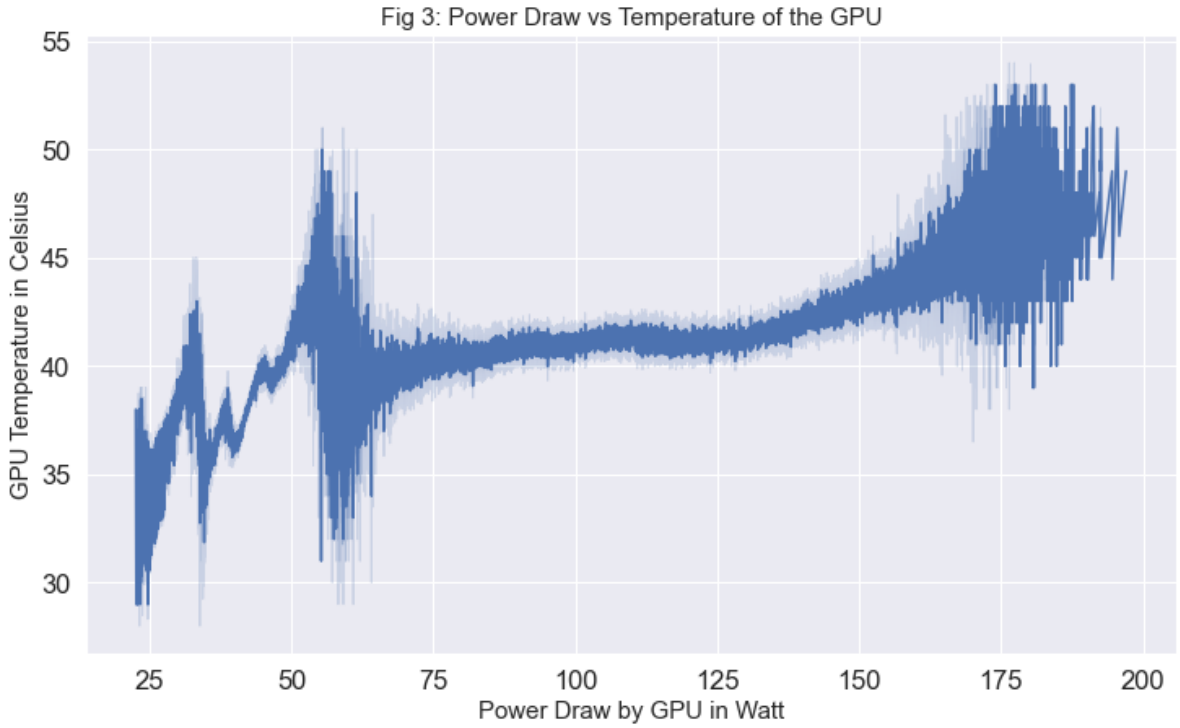
From Fig 2 and table above, we can observe following key points:

- Event Render is taking most of the processing time as we can see it is almost equivalent to event TotalRender, however, we have considerable amount of outliers in both Render and TotalRender.
- On an average Rendering takes 42 seconds and interquantile range is from 38 to 45 approx.
- Saving Config, Tiling, and Uploading doesn't take much computation time.
- Uploading event exhibits significant number of outliers.

Interplay between Performance and temperature.

In [42]:

```
# Line plot to show relation between power draw (performance) and GPU temperature...
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
ax = sns.lineplot(data=gpu, x="powerDrawWatt", y="gpuTempC")
sns.set(font_scale = 1.5) # Text font scale...
plt.xlabel('Power Draw by GPU in Watt', fontsize = 15)
plt.ylabel('GPU Temperature in Celsius', fontsize = 15)
plt.title("Fig 3: Power Draw vs Temperature of the GPU", fontsize = 15)
plt.show()
```



From Fig 3, we can perceive following pointers:

- With increase in power draw(Performance) there is an increase in temperature as well.
- We can also observe that at around 60 watt we have huge fluctuations in the temperatures, so we can say that there are some low performing GPUs which are reaching higher temperatures even at lower power draw.
- Even though temperatures are increasing with increase in performance it is not going absurdly high. Hence, in general, we can say that cooling infrastructure is good. However, this doesn't tell the full story so, we will further look into variation in gpu utilization and gpu memory utilization as opposed to power draw.

In [43]:

```
# Line plot to see the relation between Power draw (Performance) and gpu utilization and gp
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
ax = sns.lineplot(data=pd.melt(gpu[["powerDrawWatt", "gpuUtilPerc", "gpuMemUtilPerc"]], 'po
plt.xlabel('Power Draw by GPU in Watt', fontsize = 15)
plt.ylabel('Percent Utilization', fontsize = 15)
plt.title("Fig 4: Power Draw vs Utilization", fontsize = 15)
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0, fontsize = 10)
plt.show()
```

To support the findings from Fig 3 we also plotted Fig 4 and below are findings:

- With increase in power draw (performance) we can see both GPU utilization and GPU memory utilization is also increasing which makes sense because to increase the performance GPU and its components' utilization will increase. Hence we can say that with increase in power draw there is increase in performance.
- However, performance gain after around 60 watt is very marginal, furthermore, at higher power draw (160+ watts) we can see drop/fluctuation in gpu utilization as well as gpu memory utilization which is a clear sign of GPU throttling.
- Also, in previous plot (Fig 3) we observed at around 60 watt there is a huge fluctuation in temperatures, so, it may not entirely be because of weaker GPUs but also because of the temperature hindering the clock speeds. We will further investigate this by plotting power draw against GPU serial number to see whether some GPUs are more powerful than others or not.

Performance of GPUs based on their serial number.

In [24]:

```
gpu['gpuSerial'] = gpu['gpuSerial'].astype(str) # Coverting gpuSerial to string from int...

# making a new dataframe with only gpuSerial and powerDrawWatt...
# and finding the mean of powerDrawWatt by grouping on gpuSerial...
gpu_powerdraw = gpu[['gpuSerial', 'powerDrawWatt']].groupby(['gpuSerial']).mean()

gpu_powerdraw.index.names = ['gpuSerial'] # Changing index of dataframe to column...
gpu_powerdraw.reset_index(inplace=True) # Resetting the index of the dataframe...
```

In [18]:

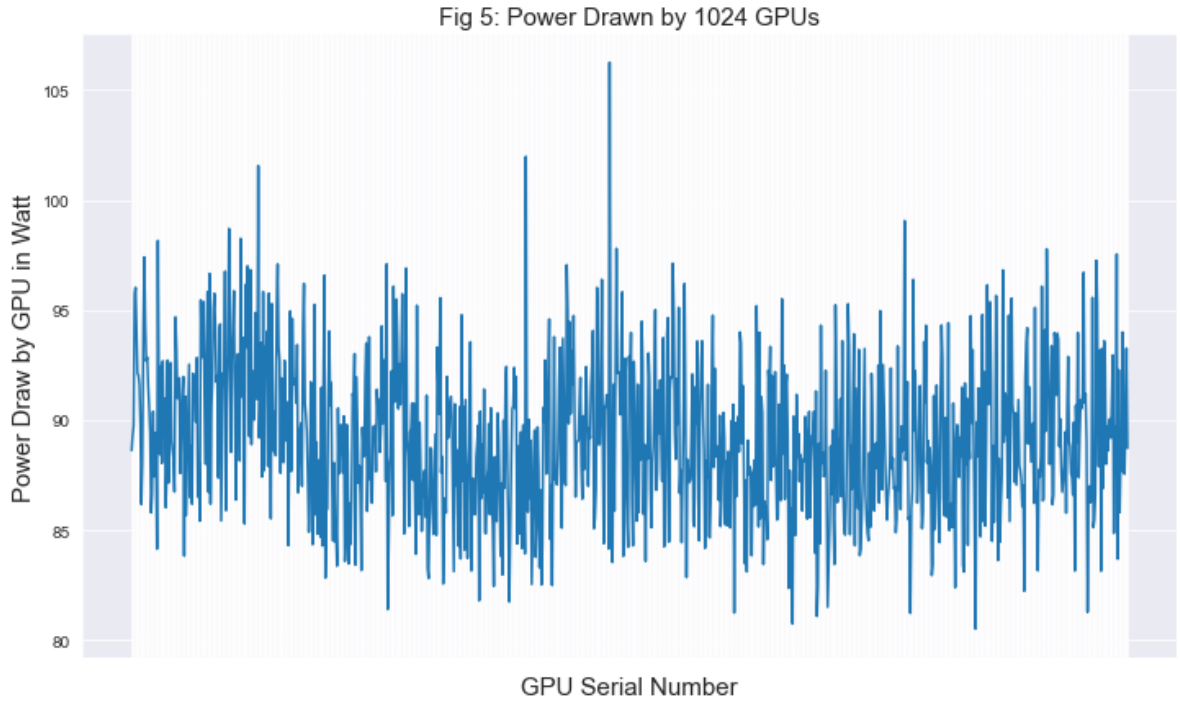
```
gpu_powerdraw.dtypes
```

Out[18]:

```
gpuSerial           int64
powerDrawWatt     float64
duration          float64
gpuUtilPerc       float64
gpuMemUtilPerc    float64
gpuTempC          float64
performance       float64
dtype: object
```

In [25]:

```
# line plot of power draw against serial number of the gpu...
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
sns.set_style("darkgrid")
s = sns.lineplot(data = gpu_powerdraw, x="gpuSerial", y="powerDrawWatt")
plt.xlabel('GPU Serial Number', fontsize = 15)
plt.ylabel('Power Draw by GPU in Watt', fontsize = 15)
plt.title("Fig 5: Power Drawn by 1024 GPUs", fontsize = 15)
s.set(xticklabels=[]) # hiding overlapped x labels (gpuSerial)
plt.show()
```



In [26]:

```
# We were trying to calculate the measure of performance by considering powerDraw, gpuTempC
# However, we found out powerDrawWatt alone is giving similar results so to save some compu
# stick to powerDrawWatt as a measure of performance...
# gpu_powerdraw['performance'] = (gpu_powerdraw['powerDrawWatt'] + gpu_powerdraw['gpuMemUti
# gpu_powerdraw['performance'] = (gpu_powerdraw['powerDrawWatt'] * gpu_powerdraw['gpuMemUti

gpu_powerdraw = gpu_powerdraw.sort_values(by = 'powerDrawWatt') # Sorting by powerDrawWatt.
print("Low Performing GPUs")
gpu_powerdraw.head()
```

```
Low Performing GPUs
```

Out[26]:

	gpuSerial	powerDrawWatt
867	325117171574	80.510313
679	325017017790	80.742673
704	325017018552	81.087075
800	325017049295	81.226755
619	324917052619	81.242179

In [27]:

```
print("High Performing GPUs")
gpu_powerdraw.tail()
```

```
High Performing GPUs
```

Out[27]:

	gpuSerial	powerDrawWatt
100	323217056123	98.698678
794	325017049041	99.057575
130	323217056368	101.549633
405	323617021202	101.974324
491	323617042596	106.247462

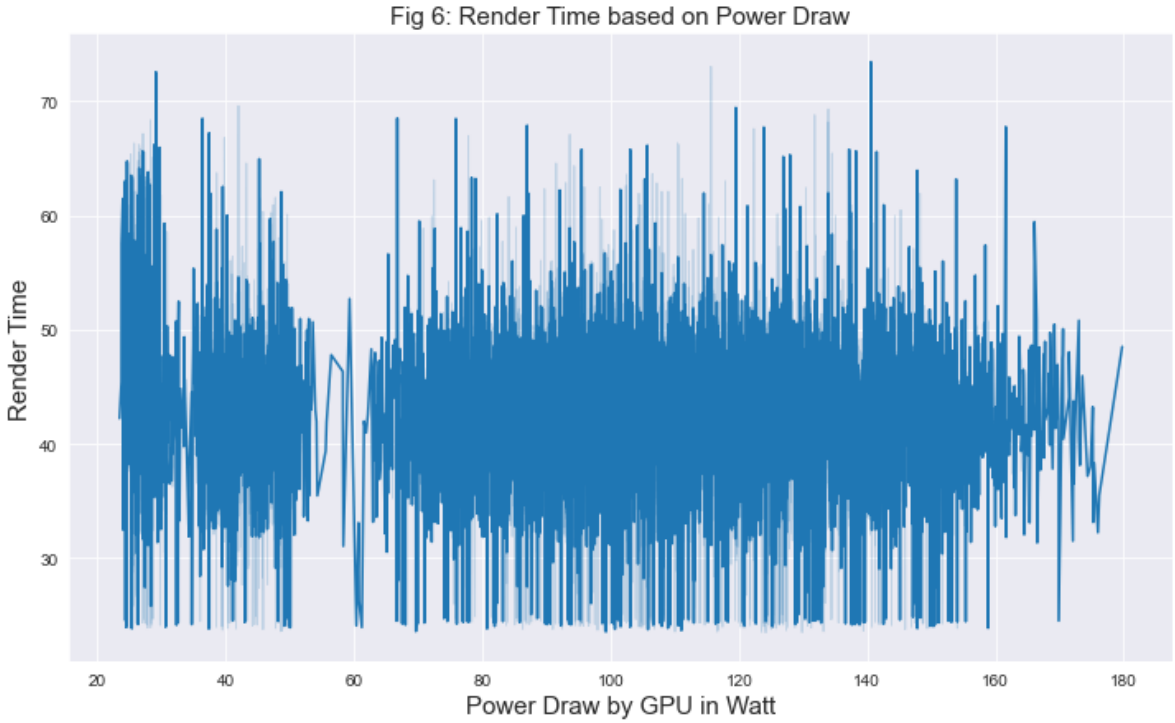
From Fig 5 and two tables, we can take out following points:

- Although there are few high performing GPUs, there isn't any set pattern of their performance based on their serial number.
- From above two tables we can see some of the serial numbers of high performing as well as low performing GPUs.
- On an average GPU with serial number **323617042596** tends to perform better than others.

Interplay between render time and power draw.

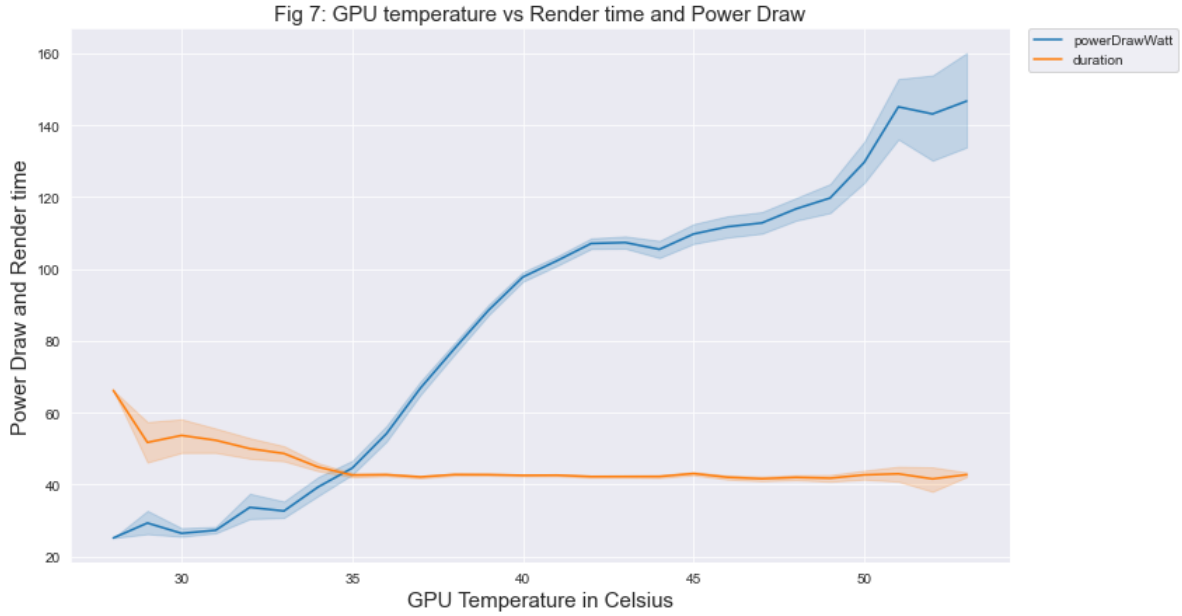
In [98]:

```
# line plot of render time against power draw...
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
ax = sns.lineplot(data=merged_data.query("eventName == 'TotalRender'"), x="powerDrawWatt",
plt.xlabel('Power Draw by GPU in Watt', fontsize = 15)
plt.ylabel('Render Time', fontsize = 15)
plt.title("Fig 6: Render Time based on Power Draw", fontsize = 15)
plt.show()
```



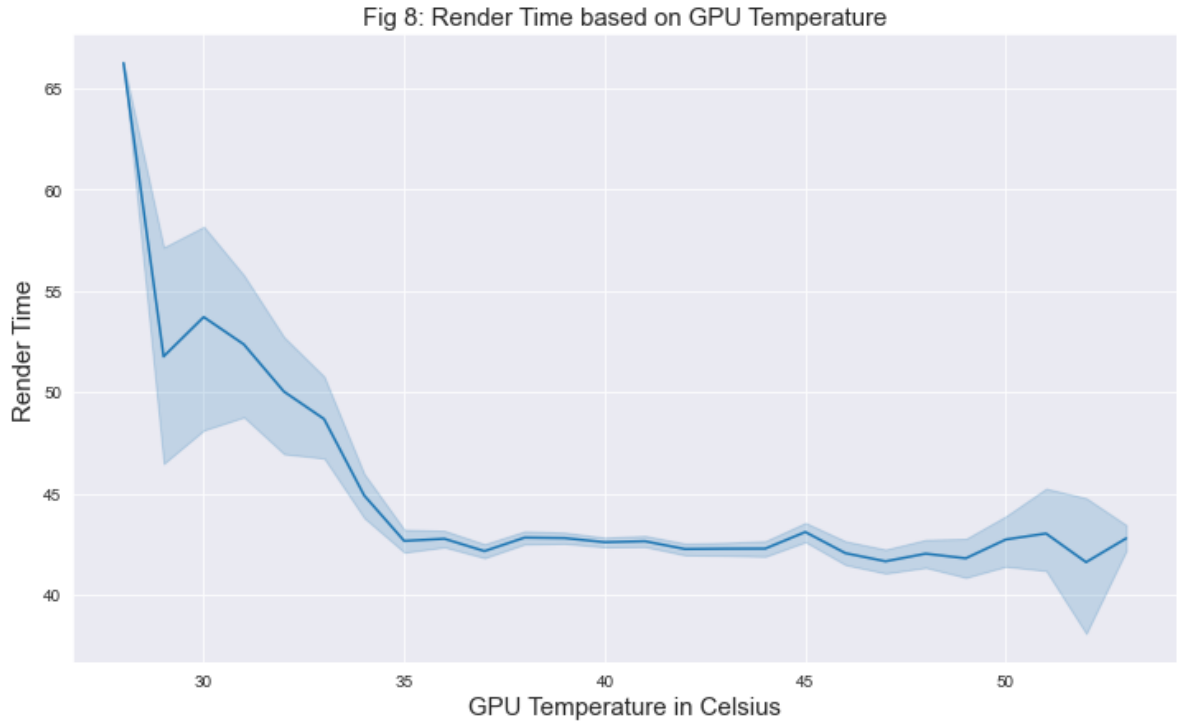
In [102]:

```
# Line plot to see the relation between GPU temperature, Power draw (Performance) and Rende
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
ax = sns.lineplot(data=pd.melt(merged_data.query("eventName == 'TotalRender'")[["powerDrawW
plt.xlabel('GPU Temperature in Celsius', fontsize = 15)
plt.ylabel('Power Draw and Render time', fontsize = 15)
plt.title("Fig 7: GPU temperature vs Render time and Power Draw", fontsize = 15)
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0, fontsize = 10)
plt.show()
```



In [103]:

```
# Line plot to see the variation in render time when GPU temperature increases...
fig, ax = plt.subplots(figsize=(12, 7)) # Figure size...
ax = sns.lineplot(data=merged_data.query("eventName == 'TotalRender'"), x="gpuTempC", y="du
plt.xlabel('GPU Temperature in Celsius', fontsize = 15)
plt.ylabel('Render Time', fontsize = 15)
plt.title("Fig 8: Render Time based on GPU Temperature", fontsize = 15)
plt.show()
```



From Fig 6, 8, and 8 we can infer following key points:

- From fig 6: there isn't any improvement in render time when power draw by the GPUs increased.
- From fig 7: As temperature is increasing we can see power draw is increasing and render time is decreasing hence, we can say for certain that cooling infrastructure is good and heating or increase in temperature does not affect the performance much.
- From fig 8: render time is decreasing drastically when temperature is approaching 35 degree celsius and then it is fairly consistent afterwards and not improving much.
- Overall, there is a significant change in the power draw but not much improvement in the render time where as, with increase in temperature we can see some improvement in render time.

Variation in computation requirements for particular tiles.

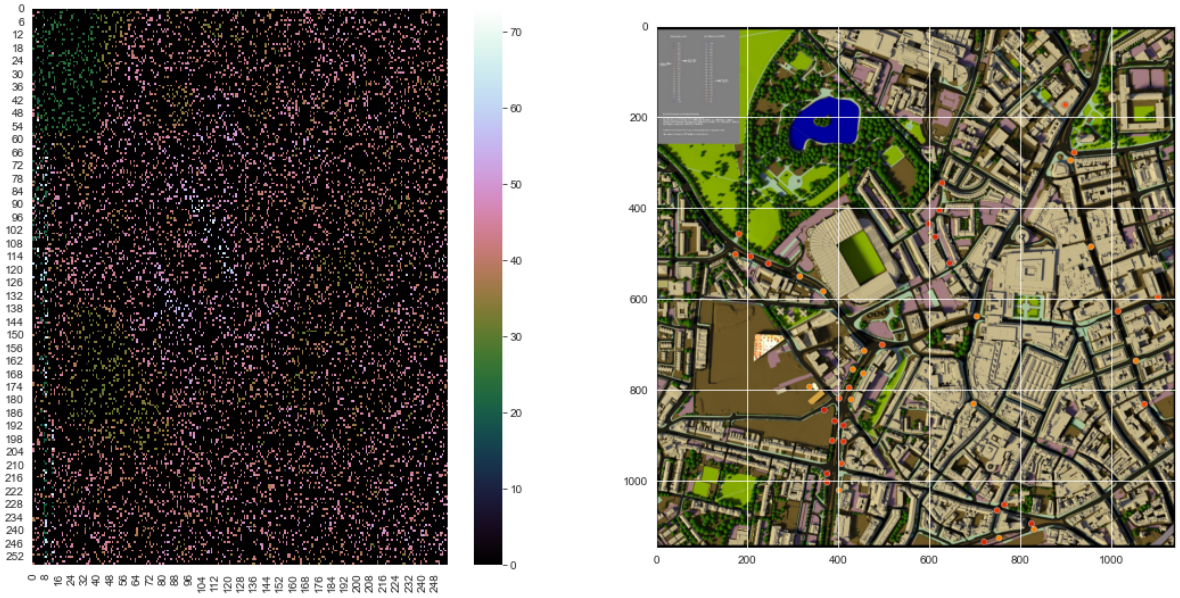
In [150]:

```
# Dropping duplicate data based on x and y values...
heatmap_data = merged_data.drop_duplicates(
  subset = ['x', 'y'],
  keep = 'last').reset_index(drop = True)

# Selecting rows with zoom lvl 12 and columns x, y, duration, and powerDrawWatt...
heatmap_data = heatmap_data[heatmap_data['level'] == 12][['x', 'y', 'powerDrawWatt', 'durat

# Converting long data(tabular format) to wide data(matrix format) which is suitable for he
heatmap_data = coo_matrix((heatmap_data.duration,(heatmap_data.x,heatmap_data.y)))
heatmap_data = heatmap_data.toarray()
```

In [154]:

```
# Heatmap of Newcastle upon tyne, pixel value as duration...
fig, ax = plt.subplots(figsize=(18, 9)) # Figure size...
fig.suptitle('Fig 9: Heatmap of Newcastle Upon Tyne', fontsize=20)
colormap = sns.color_palette("cubehelix", as_cmap=True) # Color scheme...
rows, cols = 1, 2 # figure has 1 row and 2 columns...
plt.subplot(rows, cols, 1) # Adding heatmap to 1st placeholder of the figure...
p = sns.heatmap(heatmap_data, cmap = colormap) # plotting Heatmap...
plt.subplot(rows, cols, 2) # Adding Newcastle Upon Tyne map to 2st placeholder of the figur
img = plt.imread('../plots/map.png')
plt.imshow(img)
plt.show()
```

Fig 9: Heatmap of Newcastle Upon Tyne



From Fig 9, we can observe following pointer(s):

- There isn't much set pattern of tiles taking more or less time to render. However, we can see more darker/greenish(less rendering time) pixels near legend and USB and brighter pixels on left edge and near St James' Park which makes sense because, area near USB doesn't have much detailing where as area near St James' Park has many buildings and streets which is taking time to render.

Conclusions and Future Scope

- Most of the computation time is being used by Rendering event.
- With increase in power draw(performance) GPU temperature is also increasing as more and more GPU and its components will be put to work.
- There isn't much improvement in the render time when power draw is increased however, with increase in gpu temperature we observed improvement in render time. This can further be analyzed as to what are the reasons that are not letting render time to improve with power draw whereas it is improving with increase in GPU temperature.
- There are 1024 GPUs and there is a very slight difference in their performance and very few are significantly better than others.
- Tiles associated with streets and building(crowded areas) are taking more time to render as compared to tiles with empty spaces. So, allocation of those high performing GPUs in crowded areas will be more efficient.

References

- Video to Demonstrate the application of terapixel visualization (https://vimeo.com/456923131)
- Visualizing Urban IoT Using Cloud Supercomputing by Nicolas Holliman, Manu Antony, Stephen Dowsland & Mark Turner (https://uksystems.org/workshop/2018/21-dowsland-visualising.pdf)
- Professor Nick Holliman, Manu Antony, Stephen Dowsland, Professor Philip James, Mark Turner, "Petascale Cloud Supercomputing for Terapixel Visualization of a Digital Twin" in arXiv, Online Publication 2019.