Performance evaluation of Terapixel rendering in Cloud (Super)computing

Submittive Assignment

Rishabh Malik(220512619)

1. Introduction

Terapixel images, which are composed of a vast number of pixels, can present information to stakeholders in an effective manner, but they require a high amount of computational power. To reduce costs and optimize the process, it's useful to use cloud-based supercomputing resources to process the data collected during the creation of a terapixel image of Newcastle Upon Tyne. The issue at hand is how to access the necessary supercomputing resources for the calculation of this large-scale image in the cloud. The objective is to analyze the data from the application checkpoint and system metric output to improve the cost-efficiency of the process.

2. Business Understanding

The business understanding phase focuses on understanding the requirement from the business perspective of the project. This is an important aspect of business for this project as it helps organizations understand the capabilities and limitations of their cloud-based supercomputing infrastructure in handling large-scale image rendering tasks.

This information can be used to make informed decisions about the use of cloud computing resources for large-scale image rendering, such as whether to invest in additional resources or optimize existing ones. Additionally, it can also help organizations identify bottlenecks in their workflow and make adjustments to improve the performance and cost-efficiency of the process.

Furthermore, the performance evaluation can also be used to compare the performance of different cloud providers or different configurations, which can be useful for organizations to choose the best solution for their specific needs. In short, this helps organizations to optimize their cloud-based supercomputing infrastructure to handle large-scale image rendering tasks efficiently, make informed decisions about the use of cloud resources, and improve the overall performance and cost-efficiency of the process.

Through this analysis we will gain insights into the following:

- Which event types dominate task runtimes?
- What is the interplay between GPU temperature and performance?
- What is the interplay between increased power draw and render time?
- Can we identify particular GPU cards (based on their serial numbers) whose performance differs from other cards? (i.e. perpetually slow cards).

3. Data Understanding

Adding to the foundation of Business Understanding, it drives the focus to identify, collect, and analyze the data sets that can help you accomplish the project goals. Here we acquire the necessary data and load it into our analysis tool and get a field overview of the dataset and its format, number of records, and other fields.

The data presented in this report was obtained from an experiment that employed a massive 1024 GPU setup. The rendering process was broken down into three distinct levels, specifically, levels 4, 8, and 12 of the final output. The information provided in the dataset gives an in-depth insight into the timing of the rendering performance, the performance of the GPU, and the exact area of the image that was being handled during each job.

Our data consist of three files:

- *application-checkpoint.csv*
- *gpu.csv*
- *task-x-y.csv*

- **application-checkpoint.csv:** This file contains information on the various events that occurred during the rendering process, including the type of event, the timestamp it occurred, and the hostname, taskId, and jobId associated with it.
  - **timestamp:** this column records the precise moment the event began or ended.
  - **hostname:** this column lists the 1024 unique GPUs used in the operation.
  - **eventName:** this column includes 5 different types of events: Tiling, Render, Saving Config, Uploading, and TotalRender.
    - **TotalRender:** this represents the overall task.
    - **Render:** this is when specific tiles of the image are being rendered.
    - **Tiling:** is the post-processing of the rendered tiles.
    - **Saving Config:** is when the specific configurations for each task are saved
    - **Uploading:** is when the post-processed tiles are uploaded to Azure Blob Storage.
  - **eventType:** This column indicates whether the event is starting or ending
  - **jobId:** this column indicates the specific level of the visualization output (level 12, level 8, and level 4)
  - **taskId:** this column provides a unique identification for each set of events, including their start and stop timestamps. There will be n/10 unique taskIds in the file, where n is the number of rows in the application-checkpoint.csv.
- **gpu.csv:** This file contains information about the status of the graphics processing units (GPUs) at a specific point in time.
  - **gpuSerial:** this file includes details such as the unique serial number for each of the 1024 GPUs.
  - **gpuUUID:** This is the system-assigned unique identifier for each GPU.
  - **powerDrawWatt:** the amount of power being used by the GPU in watts.
  - **gpuTempC:** the temperature of the GPU in degrees Celsius
  - **gpuUtilPerc:** the percentage of the GPU's cores that are being used.
  - **gpuMemUtilPerc:** the percentage of the GPU's memory that is being utilized.
- **task-x-y.csv:** This file contains information about the specific sections of an image that are being rendered.
  - **x and y:** it includes the x and y coordinates of the tiles being processed.
  - **level:** this is the level of zoom being used. The visualization is designed to be similar to a "google maps style" map, with 12 levels of zoom available. The data only includes levels 4, 8, and 12, as the intermediate levels are created during the post-processing, or tiling, phase.

4. Data Processing

This phase is also called data munging. Here we will now merge the 3 data files into one to make a meaningful dataset.

- Firstly left join the 2 data files *task-x-y.csv* and *application-checkpoint.csv* based on two columns *jobId* and *taskId*.
- Secondly inner join the *gpu.csv* file with the data from the previous step based on column *timestamp*.

Now on the final data file data cleaning is to be performed, as there are some duplicate values and we can see that the timestamp column is not in the proper format. Hence we delete the duplicates in the final data file. Also, we use the *datetime* library to format the timestamp column to make it standard time for further calculations.

As we have already understood in the Business Understanding phase we need to calculate the duration of the events, hence we need to perform munge on the timestamp column. Now we create 2 data frames **df_start** and **df_stop**. For the *df_start* we take the eventType = 'START' and put the timestamp in the as *start_time* and similarly for eventType = 'STOP' we put the timestamp in *stop_time*. Merged start and stop data frames of respective events on taskId and then subtract start and stop time to compute the duration and store it in a 'duration' column. Now the final merged dataset has the duration time as well for further computations.

5. Exploratory Data Analysis

In [79]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from matplotlib import pyplot
from pandas import Series
import statistics
from datetime import datetime as dt, date
from dateutil import parser
```

In [80]:
```
#reading the csv files
gpudf = pd.read_csv('gpu.csv')
appdf = pd.read_csv('application_checkpoints.csv')
taskdf = pd.read_csv('task-x-y.csv')
```

In [81]:
```
app_task_df = pd.merge(taskdf, appdf, on = ['jobId','taskId'], how='left')#applying left join
app_task_gpu_df = pd.merge(app_task_df, gpudf, on = ['timestamp'])#applying inner join
```

In [82]:
```
app_task_gpu_df.head()
```

Out[82]:

	taskId	jobId	x	y	level	timestamp	hostname_x	eventName	eventType	hostname_y	gpuSerial	gpuUUID	powerD
0	00004e77-304c-4fbd-88a1-1346ef947567	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	116	178	12	2018-11-08T08:07:10.688Z	0745914f4de046078517041d70b22fe7000001	Tiling	STOP	265232c5f6814768aeefa66a7bec6ff600000Q	323617020812	GPU-f8ed40fb-e2c2-d6e2-9a66-4b6eaae14912	
1	00004e77-304c-4fbd-88a1-1346ef947567	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	116	178	12	2018-11-08T08:07:10.688Z	0745914f4de046078517041d70b22fe7000001	Tiling	STOP	0d56a730076643d585f77e00d2d8521a00000Q	325117171574	GPU-d84a1024-9381-c725-3b85-dd7143e64c35	
2	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:48.855Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	Uploading	START	d8241877cd994572b46c861e5d144c8500000W	323617021323	GPU-f5716e40-9e60-f0af-7a7a-95d2fc4d2fd9	
3	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:48.855Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	Render	STOP	d8241877cd994572b46c861e5d144c8500000W	323617021323	GPU-f5716e40-9e60-f0af-7a7a-95d2fc4d2fd9	
4	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:49.863Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	TotalRender	STOP	8b6a0eebc87b4cb2b0539e81075191b900000U	323617021463	GPU-f955049b-a429-3eed-f7f1-30a072341123	

In [83]:
```
app_task_gpu_df = app_task_gpu_df.drop_duplicates(subset=None, keep='first', inplace=False)# deleting duplicate rows
```

In [84]:
```
app_task_gpu_df.shape
```

Out[84]:

```
(336488, 16)
```

In [85]:
```
app_task_gpu_df["time"] = app_task_gpu_df["timestamp"].apply(lambda x: parser.isoparse(x)) # extracting only time part and storing it in "time" column
```

In [86]:
```
app_task_gpu_df.head()
```

Out[86]:

	taskId	jobId	x	y	level	timestamp	hostname_x	eventName	eventType	hostname_y	gpuSerial	gpuUUID	powerD
0	00004e77-304c-4fbd-88a1-1346ef947567	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	116	178	12	2018-11-08T08:07:10.688Z	0745914f4de046078517041d70b22fe7000001	Tiling	STOP	265232c5f6814768aeefa66a7bec6ff600000Q	323617020812	GPU-f8ed40fb-e2c2-d6e2-9a66-4b6eaae14912	
1	00004e77-304c-4fbd-88a1-1346ef947567	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	116	178	12	2018-11-08T08:07:10.688Z	0745914f4de046078517041d70b22fe7000001	Tiling	STOP	0d56a730076643d585f77e00d2d8521a00000Q	325117171574	GPU-d84a1024-9381-c725-3b85-dd7143e64c35	
2	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:48.855Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	Uploading	START	d8241877cd994572b46c861e5d144c8500000W	323617021323	GPU-f5716e40-9e60-f0af-7a7a-95d2fc4d2fd9	
3	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:48.855Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	Render	STOP	d8241877cd994572b46c861e5d144c8500000W	323617021323	GPU-f5716e40-9e60-f0af-7a7a-95d2fc4d2fd9	
4	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08T08:14:49.863Z	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	TotalRender	STOP	8b6a0eebc87b4cb2b0539e81075191b900000U	323617021463	GPU-f955049b-a429-3eed-f7f1-30a072341123	

In [87]:
```
#For calculating the duration (stop time - start time)
df_start = app_task_gpu_df[app_task_gpu_df['eventType'] == 'START']
df_stop = app_task_gpu_df[app_task_gpu_df['eventType'] == 'STOP']

df_start = df_start.rename(columns={"timestamp": "start_time"})
df_stop = df_stop.rename(columns={"timestamp": "stop_time"})

df_start = df_start.drop('eventType', axis = 1)
df_stop = df_stop.drop('eventType', axis = 1)

# convert columns to datetime
df_start["start_time"] = pd.to_datetime(df_start["start_time"])
df_stop["stop_time"] = pd.to_datetime(df_stop["stop_time"])

df = pd.merge(df_start, df_stop, on=['eventName', 'x', 'y', 'level'])

# calculate duration
df['duration'] = (df['stop_time'] - df['start_time']).dt.total_seconds()
df
```

Out[87]:

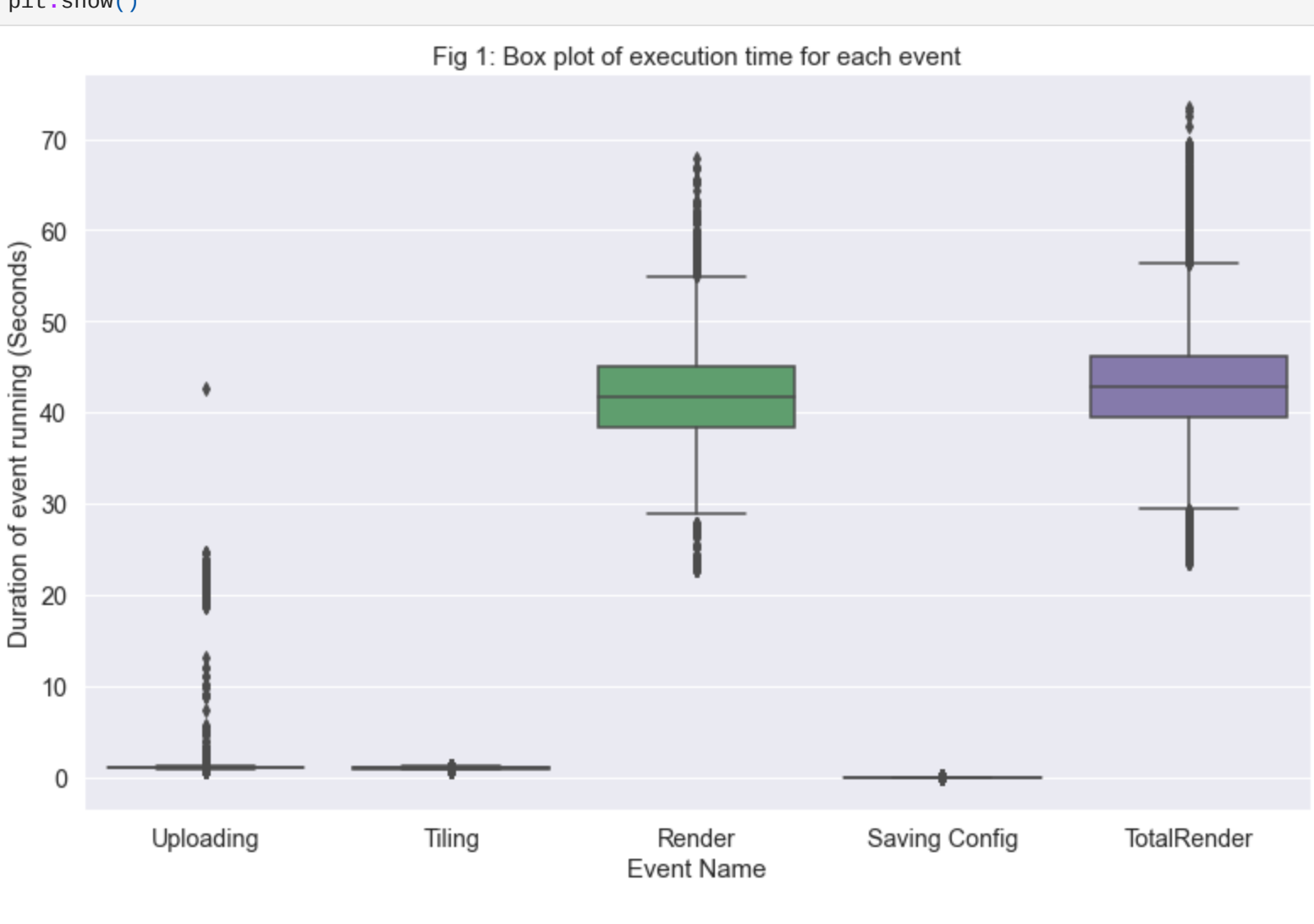
	taskId_x	jobId_x	x	y	level	start_time	hostname_x_x	eventName	hostname_y_x	gpuSerial_x	...	
0	0002afb5-d05e-4da9-bd53-7b6dc19ea6d4	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	190	12	2018-11-08 08:14:48.855000+00:00	83ea61ac1ef54f27a3bf7bd0f41ecaa700001A	Uploading	d8241877cd994572b46c861e5d144c8500000W	323617021323	...	83ea61ac1ef54f27a
1	0003c380-4db9-49fb-8e1c-6f8ae466ad85	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	86	12	2018-11-08 07:49:17.082000+00:00	83ea61ac1ef54f27a3bf7bd0f41ecaa7000004	Tiling	0745914f4de046078517041d70b22fe7000014	325117064385	...	83ea61ac1ef54f27a
2	0003c380-4db9-49fb-8e1c-6f8ae466ad85	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	86	12	2018-11-08 07:49:17.082000+00:00	83ea61ac1ef54f27a3bf7bd0f41ecaa7000004	Tiling	8b6a0eebc87b4cb2b0539e81075191b900000T	323617021011	...	83ea61ac1ef54f27a
3	0003c380-4db9-49fb-8e1c-6f8ae466ad85	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	142	86	12	2018-11-08 07:49:17.082000+00:00	83ea61ac1ef54f27a3bf7bd0f41ecaa7000004	Tiling	265232c5f6814768aeefa66a7bec6ff600000T	323617020801	...	83ea61ac1ef54f27a
4	000d1def-1478-40d3-a5e3-4f848daee474	1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705	179	226	12	2018-11-08 08:21:27.481000+00:00	b9a1fa7ae2f74eb68f25f607980f97d700000V	Tiling	b9a1fa7ae2f74eb68f25f607980f97d7000019	324917052134	...	b9a1fa7ae2f74eb6
...	...	...	...	...	...	...	...	...	...	...	...	
85800	ecb08d02-79b5-43ef-b58c-b55b4b84e39c	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63	12	4	8	2018-11-08 07:42:39.161000+00:00	83ea61ac1ef54f27a3bf7bd0f41ecaa700000J	Uploading	8b6a0eebc87b4cb2b0539e81075191b900001D	323617021151	...	83ea61ac1ef54f27a
85801	f7b53cc0-d2f1-421b-a7a1-4949ddc38075	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63	2	6	8	2018-11-08 07:42:50.906000+00:00	cd44f5819eba427a816e7ce648adceb2000001	Uploading	265232c5f6814768aeefa66a7bec6ff6000001	323217056104	...	cd44f5819eba427a8
85802	f8559617-7298-4ee0-9742-9e9e7b5f54df	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63	14	3	8	2018-11-08 07:43:14.194000+00:00	0d56a730076643d585f77e00d2d8521a00000R	Tiling	6139a35676de44d6b61ec247f0ed8657000018	325017018337	...	0d56a730076643d58
85803	fb9f46a2-a940-4773-996f-6db2204350d2	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63	0	0	8	2018-11-08 07:42:13.045000+00:00	04dc4e9647154250beeee51b866b071500000X	Uploading	04dc4e9647154250beeee51b866b071500000T	323217056004	...	04dc4e9647154250be
85804	fb9f46a2-a940-4773-996f-6db2204350d2	1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63	0	0	8	2018-11-08 07:41:50.165000+00:00	04dc4e9647154250beeee51b866b071500000X	Render	95b4ae6d890e4c46986d91d7ac4bf082000006	324917146982	...	04dc4e9647154250be

85805 rows × 29 columns

Which event types dominate task runtimes?

In [88]:
```
# Box plot
fig, ax = plt.subplots(figsize=(13, 8))
sns.boxplot(x="eventName", y="duration", data=df, ax=ax)
sns.set(font_scale = 1.1)
sns.set_theme(style="darkgrid", palette = 'gist_earth_r')
ax.set_xlabel('Event Name', fontsize = 15)
ax.set_ylabel('Duration of event running (Seconds)', fontsize = 15)
ax.set_title("Fig 1: Box plot of execution time for each event", fontsize = 15)
plt.show()
```

Fig 1: Box plot of execution time for each event



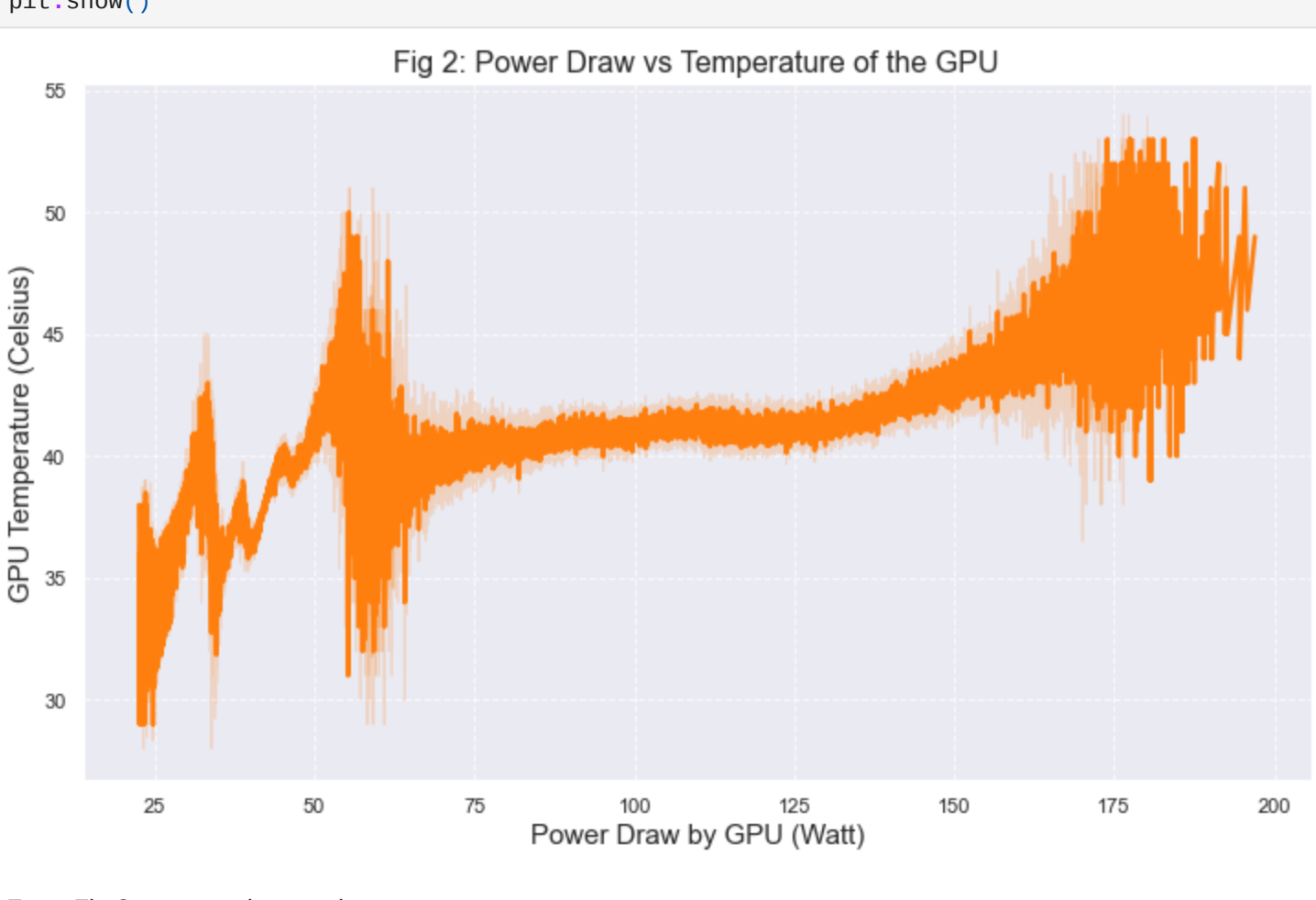
From Fig1, we can observe that

- The majority of the processing time is being consumed by the Event Render function, which is almost equal to the Event TotalRender function. However, there are a significant number of outliers present in both Render and TotalRender.
- On average, rendering takes 42 seconds.
- Saving Config, Tiling, and Uploading do not require a lot of computational time.
- There is also a notable amount of outliers present in the Uploading event.

What is the interplay between GPU temperature and performance?

In [89]:
```
# Line plot
fig, ax = plt.subplots(figsize=(12, 7))
ax = sns.lineplot(data=gpudf, x="powerDrawWatt", y="gpuTempC",
                  color='#ff7f0e', linewidth=2.5)
ax.set_xlabel('Power Draw by GPU (Watt)', fontsize = 15)
ax.set_ylabel('GPU Temperature (Celsius)', fontsize = 15)
ax.set_title("Fig 2: Power Draw vs Temperature of the GPU", fontsize = 16)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Fig 2: Power Draw vs Temperature of the GPU



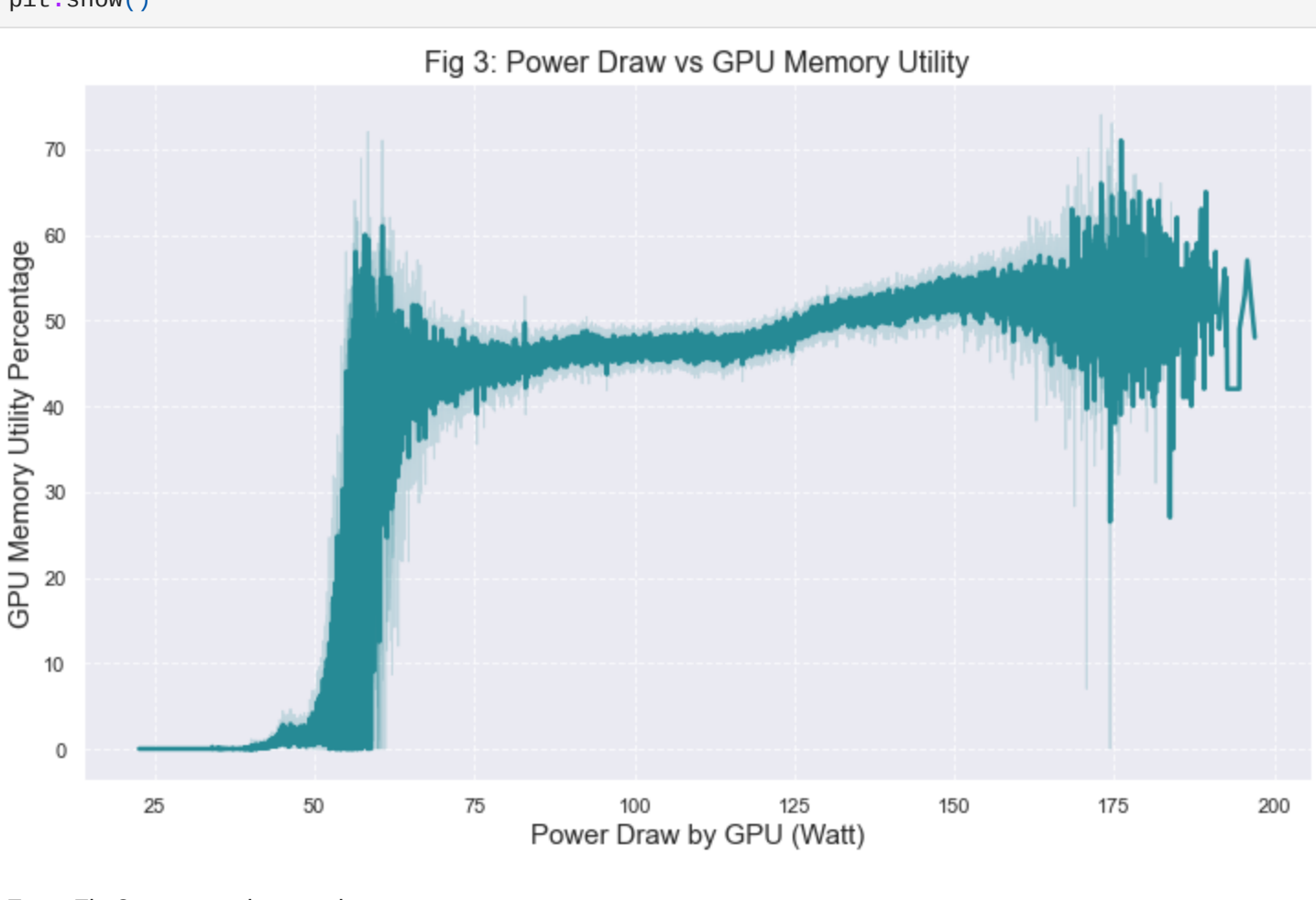
From Fig 2, we can observe that

- As the powerDraw increases the temperature of the gpu also rises.
- We can see that the temperatures fluctuate significantly around 60 watts, indicating that some GPUs with lower performance are reaching higher temperatures at lower power draw.
- We can see that the temperature is increasing with the increase in powerDraw but the increase is not significantly high. This brings us to an conclusion that the cooling infrasture for the GPU is very good.

As this does not depict the complete picture we will dig more into utilization and memory.

In [90]:
```
#line plot
fig, ax = plt.subplots(figsize=(12, 7))
ax = sns.lineplot(data=gpudf, x="powerDrawWatt", y="gpuMemUtilPerc",
                  color='#268a95', linewidth=2.5)
ax.set_xlabel('Power Draw by GPU (Watt)', fontsize = 15)
ax.set_ylabel('GPU Memory Utility Percentage', fontsize = 15)
ax.set_title("Fig 3: Power Draw vs GPU Memory Utility", fontsize = 16)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Fig 3: Power Draw vs GPU Memory Utility



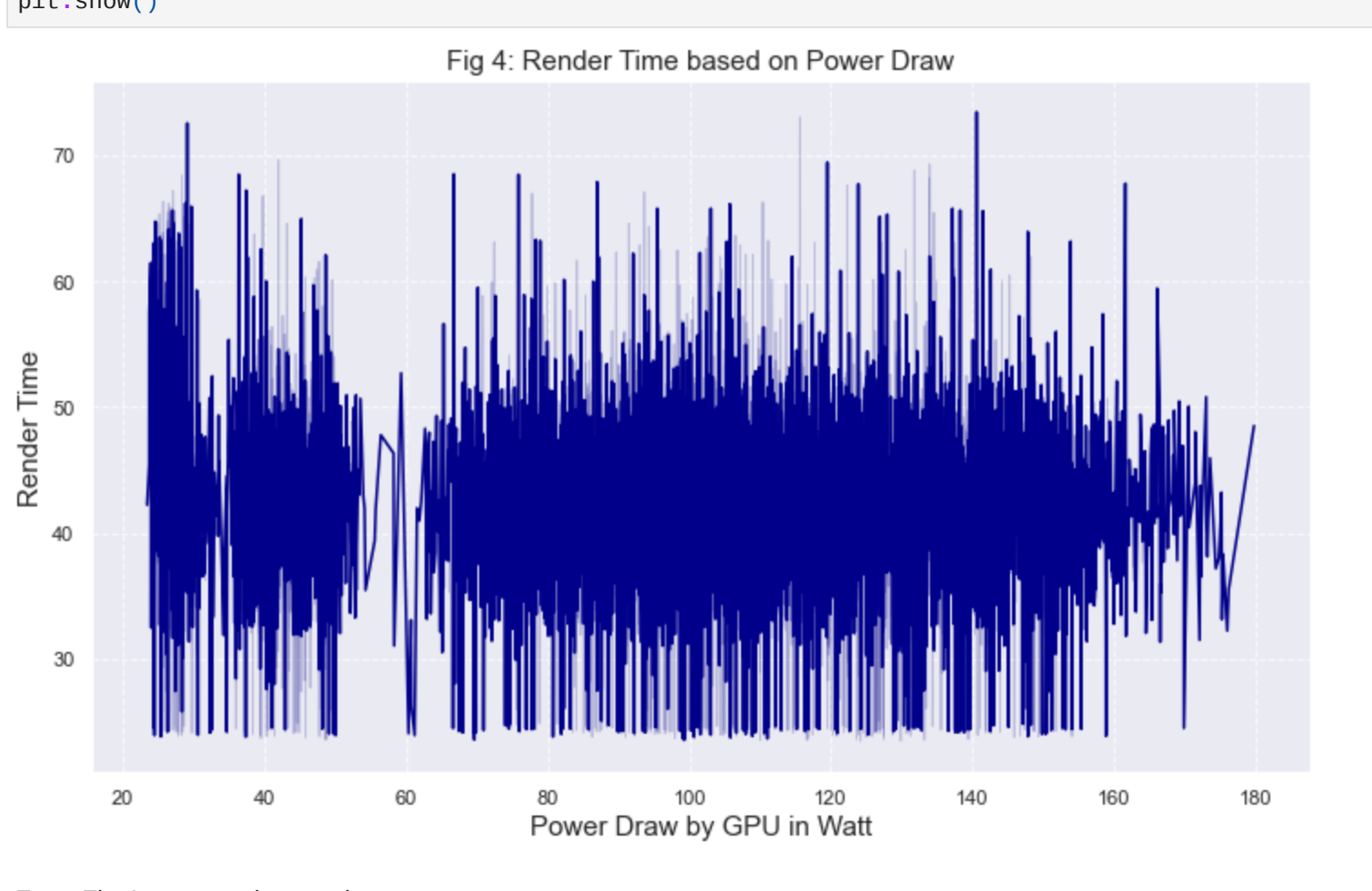
From Fig 3, we can observe that

- Although this graph also depicts the same story, we can see that with the increase in powerDraw more gpu memory is being utilised.
- Same as the previous graph we can see that at 60W few of the GPU's have reached the peak utilization. So we can infer those are low performing GPU's.
- Also at 160W we can see GPU throttling happen, which can result in performance drop.

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

In [91]:
```
# line plot
fig, ax = plt.subplots(figsize=(12, 7))
ax = sns.lineplot(data=df.query("eventName == 'TotalRender'"), x="powerDrawWatt_y", y="duration",
                  markers=True, dashes=False, color='darkblue')
ax.set_xlabel('Power Draw by GPU in Watt', fontsize = 15)
ax.set_ylabel('Render Time', fontsize = 15)
ax.set_title("Fig 4: Render Time based on Power Draw", fontsize = 15)
ax.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Fig 4: Render Time based on Power Draw



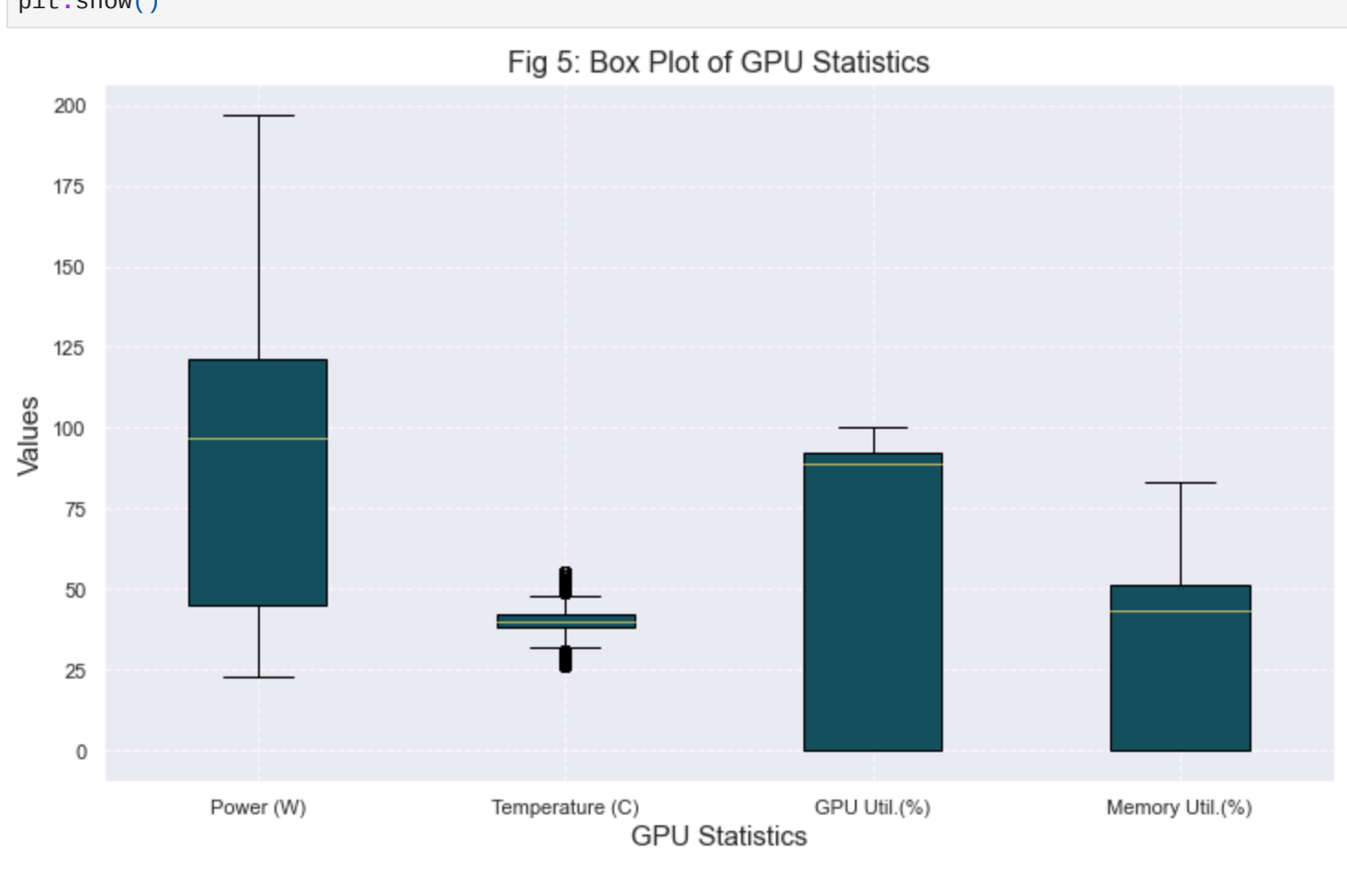
From Fig 4, we can observe that

- We see no increase in render time despite the increase in power draw from the GPUs.
- Although the power demand has changed significantly overall, the render time hasn't improved much, whereas the render time has improved slightly as the temperature has increased.

Can we identify particular GPU cards whose performance differs from other cards?

In [92]:
```
colors = '#134f5c'
fig, ax = plt.subplots(figsize=(12, 7))
ax.boxplot([gpudf['powerDrawWatt'], gpudf['gpuTempC'], gpudf['gpuUtilPerc'], gpudf['gpuMemUtilPerc']], patch_artist=True, labels = gpu_stats, boxprops = dict(facecolor
ax.set_title('Fig 5: Box Plot of GPU Statistics', fontsize=16)
ax.set_xlabel('GPU Statistics', fontsize=15)
ax.set_ylabel('Values', fontsize=15)
ax.grid(True, linestyle='--', alpha=0.7)
plt.rcParams['figure.figsize'] = [12, 7]
plt.show()
```

Fig 5: Box Plot of GPU Statistics



From Fig 5, we can observe that

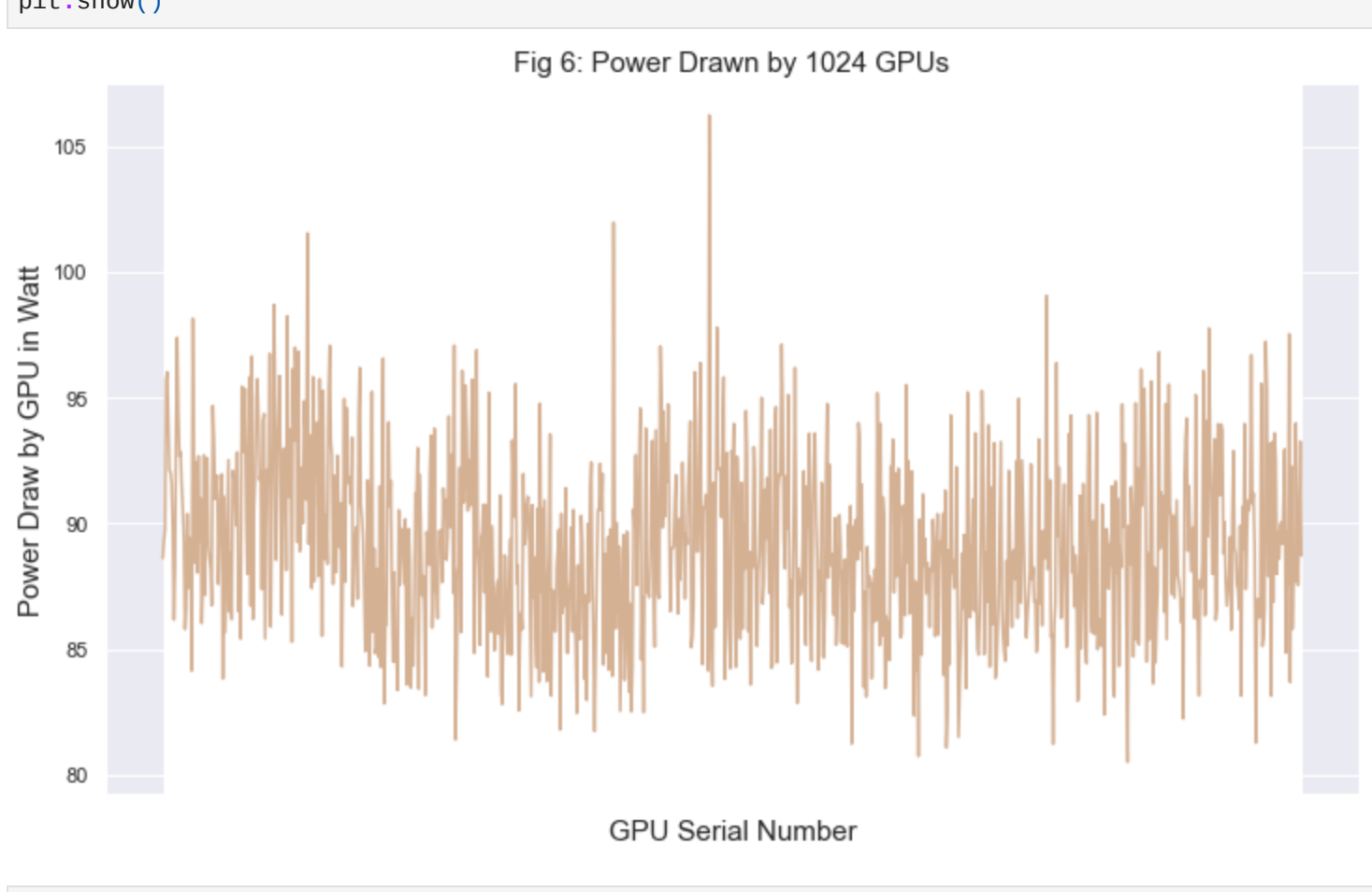
- The average power consumption is around 95 watts.
- The average temperature is 35 degrees Celsius.
- The average GPU utilization is 80%.
- The average memory utilization is 40%.

This implies that there is a roughly equal number of underperforming and high-performing GPUs, as they all seem to be maintaining an average performance level at high GPU utilization.

In [93]:
```
gpudf['gpuSerial'] = gpudf['gpuSerial'].astype(str)
gpu_powerdraw = gpudf.groupby('gpuSerial', as_index=False)['powerDrawWatt'].mean()
```

In [94]:
```
# line plot
fig, ax = plt.subplots(figsize=(12, 7))
sns.set_style("whitegrid")
plot = sns.lineplot(data = gpu_powerdraw, x="gpuSerial", y="powerDrawWatt", ax=ax)
plt.xlabel('GPU Serial Number', fontsize = 15)
plt.ylabel('Power Draw by GPU in Watt', fontsize = 15)
plt.title("Fig 6: Power Drawn by 1024 GPUs", fontsize = 15)
plot.set(xticklabels=[])
plt.show()
```

Fig 6: Power Drawn by 1024 GPUs



In [95]:
```
#Creating Min and Max GPU's
gpu_powerdraw = gpu_powerdraw.sort_values(by = 'powerDrawWatt') # Sorting by powerDrawWatt...
print("Low Performing GPUs")
df_low = gpu_powerdraw.head(5)
df_low
```

```
Low Performing GPUs
```

Out[95]:

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

In [96]:
```
print("High Performing GPUs")
df_high = gpu_powerdraw.tail()
df_high
```

```
High Performing GPUs
```

Out[96]:

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

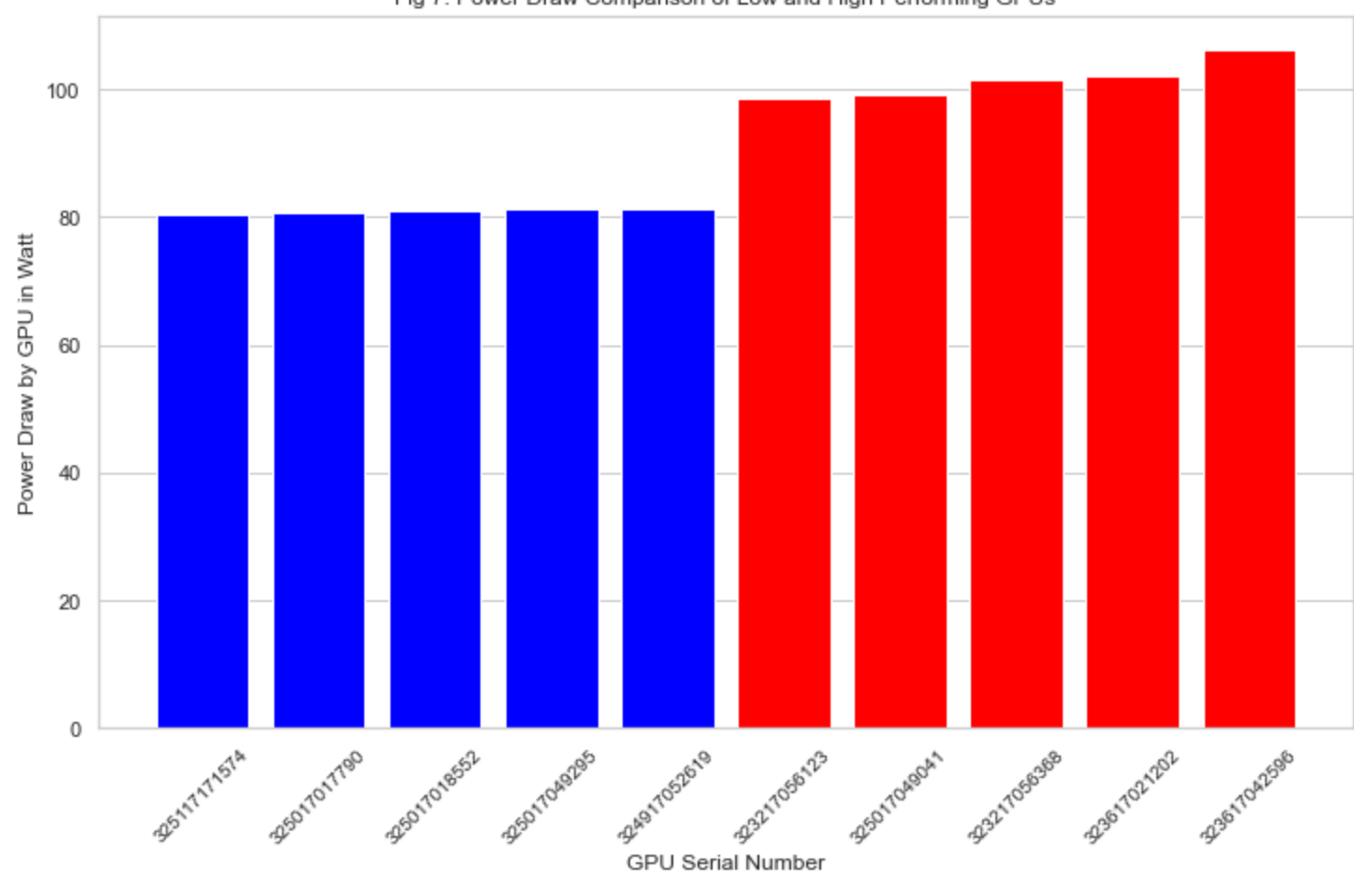
In [97]:
```
df_gpu_min_max = pd.concat([df_low, df_high])
df_gpu_min_max
```

Out[97]:

	gpuSerial	powerDrawWatt
867	325117171574	80.510313
679	325017017790	80.742673
704	325017018552	81.087075
800	325017049295	81.226755
619	324917052619	81.242179
100	323217056123	98.698678
794	325017049041	99.057575
130	323217056368	101.549633
405	323617021202	101.974324
491	323617042596	106.247462

In [98]:
```
# Set the x-axis to be the GPU serial number
x = df_gpu_min_max['gpuSerial']

# Set the y-axis to be the power draw in watts
y = df_gpu_min_max['powerDrawWatt']

# Create a new figure and axis
fig, ax = plt.subplots(figsize=(12, 7))

# Plot the low-performing GPUs as blue bars
ax.bar(x[:5], y[:5], color='blue')

# Plot the high-performing GPUs as red bars
ax.bar(x[5:], y[5:], color='red')

# Label the x and y axes
ax.set_xlabel('GPU Serial Number')
ax.set_ylabel('Power Draw by GPU in Watt')

# Add a title to the plot
ax.set_title('Fig 7: Power Draw Comparison of Low and High Performing GPUs')

ax.yaxis.grid(True)
ax.xaxis.grid(False)

# Rotate the x-axis labels by 45 degrees
plt.xticks(rotation=45)

# Reduce the font size of the x-axis labels
plt.tick_params(axis='x', labelsize=10)
# Show the plot
plt.show()
```

Fig 7: Power Draw Comparison of Low and High Performing GPUs



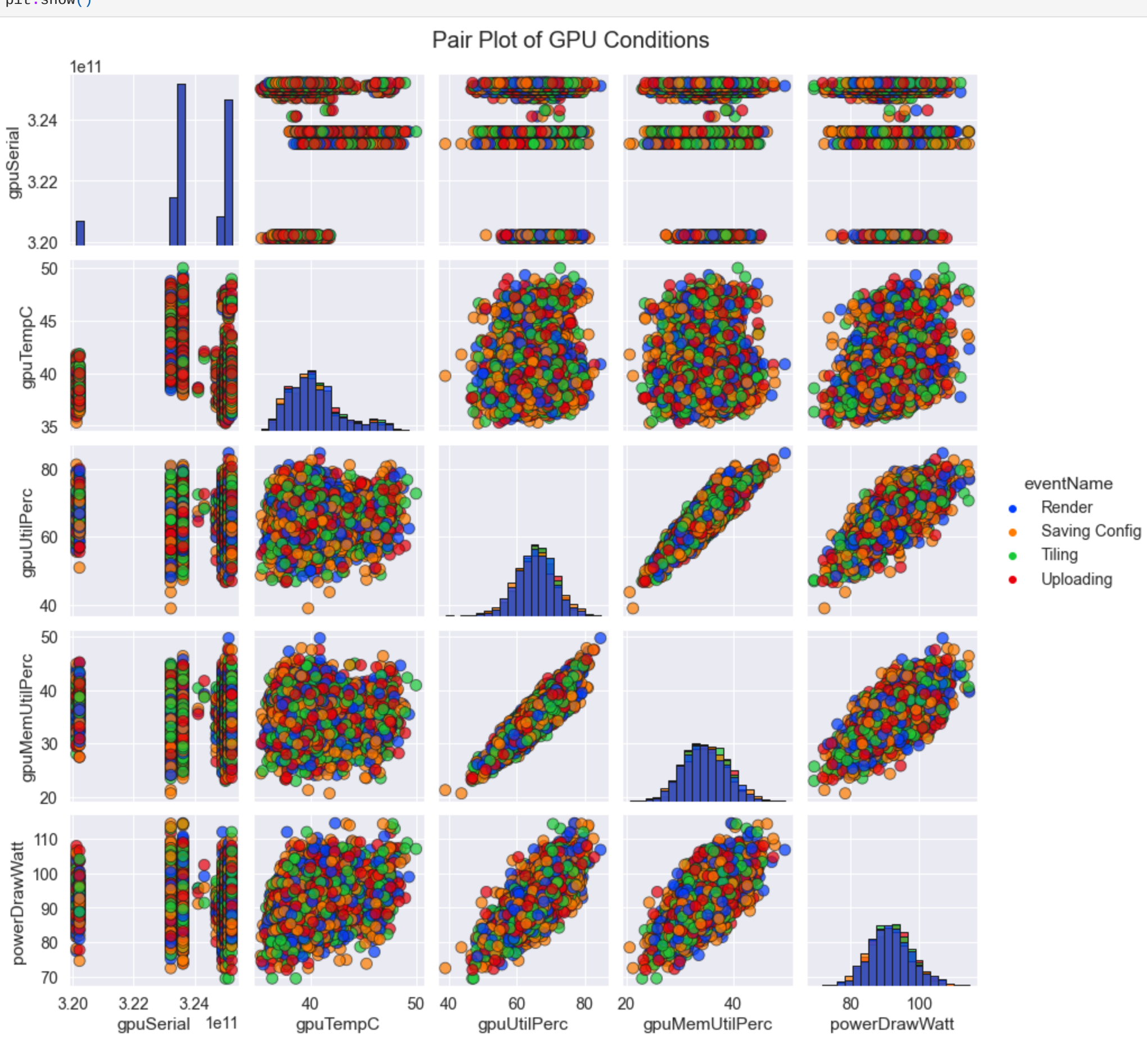
From Fig 6 and Fig 7, and also the GPU tables we can observe that,

- Although some GPUs perform well, there doesn't appear to be a clear correlation between their serial numbers and their performance.
- The two tables above show some examples of both high-performing and low-performing GPUs.
- GPU serial number 323617042596 is the highest performing GPU, while GPU serial number 325117171574 is the lowest performing GPU.

In [99]:
```
# Pair plot of all the GPU conditions for the Abstract Report.
# 1 picture = 1000 words

render_data = app_task_gpu_df[app_task_gpu_df['eventName'] != 'TotalRender']
avg_data = render_data[['gpuSerial', 'eventName', 'gpuTempC', 'gpuUtilPerc', 'gpuMemUtilPerc', 'powerDrawWatt']].groupby(['gpuSerial','eventName']).mean()
avg_data.reset_index(inplace=True)
```

In [100…
```
# Pairplot of averaged data to see relationship between various GPU conditions.

sns.set(font_scale = 1.3, rc={'figure.figsize':(15, 10)})
f = sns.pairplot(avg_data, hue="eventName", diag_kind='hist',
                 diag_kws=dict(bins=20, edgecolor="k", linewidth=1,alpha=0.7),
                 plot_kws=dict(s=100, alpha=0.7, edgecolor="k"),palette='bright')
f.fig.suptitle("Pair Plot of GPU Conditions", fontsize = 20, y=1.01)
plt.xticks(rotation=45)
plt.show()
```

Pair Plot of GPU Conditions



6. Reflection:

The dataset provided was pretty good and did not require much munging or data cleaning. The data was very much wrangled as the data was able to make sense on it own.
So, everything comes with bad and good experiences, and working on this dataset was the same. Got to learn a lot from the cloud perspective. Got familiar with many new terminology and methadology. Got familier with new tools like Git and Jupyter notebook. All these tools made the work easier and streamlined. With no prior knowledge about Git at first it felt a bit tedious but as the work progressed, got a hang of it and these tools made my work easier.

Although most of the things are good but struggled while converting the ipynb file to pdf. As working on a macbook and safari made it more tedious as the support for safari browser is not that good. Coped with this huddle and was able to find a good solution to implement for this problem.

If I rose again and had to do the same thing again, I would defiantly do it differently. I would like to make use of Git more and it's version control features which I was not able to explore that much this time around. Apart from all that it was a good experience working on this project and learned a lot about the cloud infrastructure side of things and the different tools and techniques available.

7. Conclusion and Future Scope

The majority of the computation time is consumed by the rendering process. As power draw increases, GPU temperature also rises as more of the GPU and its components are utilized. Despite this increase in power draw, there is little improvement in render time. However, render time does improve as GPU temperature increases. This discrepancy warrants further investigation to understand why render time does not improve with increased power draw. Out of 1024 GPUs, only a small number of them have significantly better performance than the rest.

The future scope of performance evaluation of Terapixel rendering in cloud (super)computing is broad and multifaceted. Some potential areas of research and development include:

- Developing more advanced and efficient algorithms for rendering large-scale images and videos, which can handle the massive amounts of data involved in Terapixel rendering.
- Improving the energy efficiency of cloud (super)computing systems, as Terapixel rendering is a computationally and power-intensive task.
- Exploring the use of edge computing and distributed computing techniques to bring the rendering process closer to the data source, reducing the amount of data transfer required.
- Investigating the use of artificial intelligence and machine learning to optimize the rendering process and reduce the time required to complete large-scale rendering tasks.
- Developing more advanced visualization tools to help users explore, analyze and interpret the large-scale images and videos generated by Terapixel rendering.
- Investigate the integration of distributed and parallel computing technologies to accelerate the process of Terapixel rendering.

Overall, the future of performance evaluation of Terapixel rendering in cloud (super)computing is promising, with many opportunities for research and development in a wide range of areas.