Performance evaluation of Terapixel rendering

Naga Pavan Kalyan Uppu

02/06/2021

Introduction

Terapixel pictures offer a natural, open way to show data sets to partners, permitting watchers to intuitiveness browse enormous information over different scales. The challenge addressed here is how to provide the supercomputer scale assets required to compute a practical terapixel visualization of the city of Newcastle upon Tyne and its natural information as captured by the Newcastle Urban Observatory. The arrangement may be a scalable engineering for cloud-based visualization that will deploy and pay for only as required.

The three key goals of this work were to: form a supercomputer design for adaptable visualization utilizing the open cloud; create a terapixel 3D city visualization supporting every day overhauls; embrace a rigorous assessment of cloud super-computing for compute-intensive visualization applications. It is possible to deliver a high-quality terapixel visualization employing a way following renderer in beneath a day utilizing open IaaS, in all sort of devices irrespective of their computational power. The data associated with the image generation is used to evaluate its performance. This work is to analyse different GPUs inorder to improve the efficiency/ performance of the super computer in the image generation.

Data Description

A run using the 1024 GPUs resulted in the creation of the three datasets(application.checkpoints, gpu and task.x.y) which shows the performance timing of the render application, performance of the GPU card, and the details of which part of the image was being rendered in each task.

```
head(application.checkpoints)
```

Description of variables:

```
## # A tibble: 6 x 6
##   timestamp     hostname        eventName  eventType jobId          taskId
##   <chr>         <chr>           <chr>      <chr>     <chr>          <chr>
## 1 2018-11-08T0~ 0d56a730076643~ Tiling     STOP      1024-lvl12-7e~ b47f0263-ba~
## 2 2018-11-08T0~ 0d56a730076643~ Saving Co~ START     1024-lvl12-7e~ 20fb9fcf-a9~
## 3 2018-11-08T0~ 0d56a730076643~ Saving Co~ STOP      1024-lvl12-7e~ 20fb9fcf-a9~
## 4 2018-11-08T0~ 0d56a730076643~ Render     START     1024-lvl12-7e~ 20fb9fcf-a9~
## 5 2018-11-08T0~ 0d56a730076643~ TotalRend~ STOP      1024-lvl12-7e~ 20fb9fcf-a9~
## 6 2018-11-08T0~ 0d56a730076643~ Render     STOP      1024-lvl12-7e~ 3dd4840c-47~
```

timestamp: time stamp at which the IoT sensor captured the data

hostname: Hostname of the virtual machine auto-assigned by the Azure batch system.

eventName: name of the event occurring within the rendering process

eventType: tells whether the event starts or stop

jobId & taskId: ID of the Azure batch job and task

There are 660400 observations about these 6 variables

```
head(gpu)
```

```
## # A tibble: 6 x 8
##   timestamp hostname gpuSerial gpuUUID powerDrawWatt gpuTempC gpuUtilPerc
##   <chr>     <chr>        <dbl> <chr>           <dbl>    <int>       <int>
## 1 2018-11-~ 8b6a0ee~   3.23e11 GPU-1d~         132.        48          92
## 2 2018-11-~ d824187~   3.24e11 GPU-04~         117.        40          92
## 3 2018-11-~ db871cd~   3.23e11 GPU-f4~         122.        45          91
## 4 2018-11-~ b9a1fa7~   3.25e11 GPU-ad~          50.2       38          90
## 5 2018-11-~ db871cd~   3.23e11 GPU-2d~         142.        41          90
## 6 2018-11-~ 265232c~   3.24e11 GPU-71~         120.        43          88
## # ... with 1 more variable: gpuMemUtilPerc <int>
```

hostname: Hostname of the virtual machine auto-assigned by the Azure batch system.

gpuSerial: The serial number of the physical GPU card.

gpuUUID: The unique system id assigned by the Azure system to the GPU unit.

powerDrawWatt: Power draw of the GPU in watts.

gpuTempC: Temperature of the GPU in Celsius

gpuUtilPerc: Percent utilisation of the GPU Core(s).

gpuMemUtilPerc: Percent utilisation of the GPU memory.

There are 1543681 observations about 8 variables - timestamp,hostname,gpuSerial, gpuUUID, powerDrawnWatt, gpuTempC, gpuUtilPerc, gpuMemUtilPerc.

```
head(task.x.y)
```

```
## # A tibble: 6 x 5
##   taskId                     jobId                                  x     y level
##   <chr>                      <chr>                              <int> <int> <int>
## 1 00004e77-304c-4fbd-88a1-1~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   116   178    12
## 2 0002afb5-d05e-4da9-bd53-7~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   142   190    12
## 3 0003c380-4db9-49fb-8e1c-6~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   142    86    12
## 4 000993b6-fc88-489d-a4ca-0~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   235    11    12
## 5 000b158b-0ba3-4dca-bf5b-1~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   171    53    12
## 6 000d1def-1478-40d3-a5e3-4~ 1024-lvl12-7e026be3-5fd0-48ee-b7~   179   226    12
```

jobId & taskId: ID of the Azure batch job and task

x: X co-ordinate of the image tile being rendered.

y: Y co-ordinate of the image tile being rendered.

level: Zoomable "google map style" map is created with 12 levels.

There are 65793 observations about these 5 variables.

Data Preprocessing

Data Preprocessing is a crucial step before proceeding with any analysis as presence of outliers, duplicate and wrong data affects the quality of the analysis' outcome. The preprocessing is carried out in the following steps:

- All the available datasets are loaded and renamed accordingly for the ease of usability.
- Removing duplicate entries from the datasets.
- Merging the Application checkpoints dataset and task_x_y dataset data on taskId and ordering the resulting dataset.
- Splitting the time stamp to calculate the run times of each task.
- Linking the x and y tile values with their task ids and GPUs, which is going to map the gpu to the respective tile it renders.

The master dataset obtained after the data preprocessing:

```
head(prep_data_piv2)
```

```
##                                 taskId                                 hostname
## 1 00390eee-c26c-41da-a02d-556bb7fcac67 04dc4e9647154250beeee51b866b0715000000
## 2 00390eee-c26c-41da-a02d-556bb7fcac67 04dc4e9647154250beeee51b866b0715000000
## 3 00390eee-c26c-41da-a02d-556bb7fcac67 04dc4e9647154250beeee51b866b0715000000
## 4 00390eee-c26c-41da-a02d-556bb7fcac67 04dc4e9647154250beeee51b866b0715000000
## 5 00390eee-c26c-41da-a02d-556bb7fcac67 04dc4e9647154250beeee51b866b0715000000
## 6 dbc599f6-694b-46c4-a864-e09ab881af37 04dc4e9647154250beeee51b866b0715000000
##       eventName   x  y level runtime
## 1   TotalRender  21 10    12      26
## 2 Saving Config  21 10    12       0
## 3        Render  21 10    12      24
## 4     Uploading  21 10    12       2
## 5        Tiling  21 10    12       1
## 6 Saving Config 224  4    12       0
```

Initial observations

- All the 1024 GPUs are encoded by hostname, gpuSerial and gpuUUID
- There are five events associated to each taskId.
- The five different EventNames are

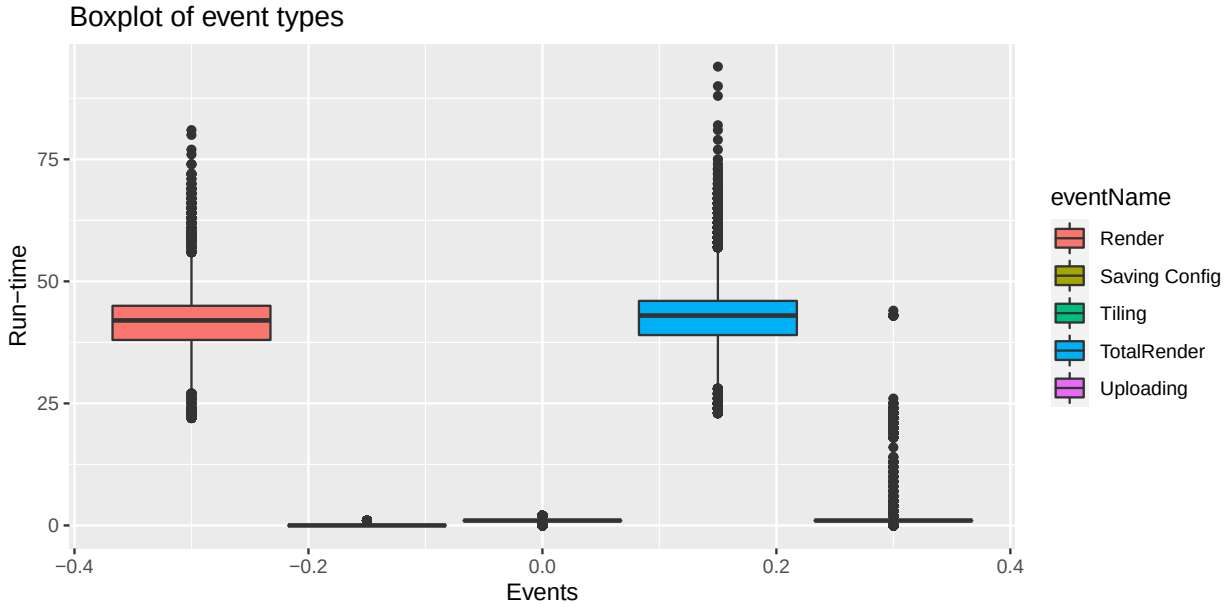
a. TotalRender: Time taken by the entire task
b. Render: Time taken by the image tile to render
c. Saving Config: Time taken for the configuration,
d. Tiling: Time taken for the post processing of the rendered tile
e. Uploading: Time taken for the output from post processing to be uploaded to the Azure cloud storage.

- Time stamps are logged at the start and end of each event.
- The gpu variables like the utilisation, memory utilisation, power and temperature are measured for every 2 seconds during the rendering of the image.

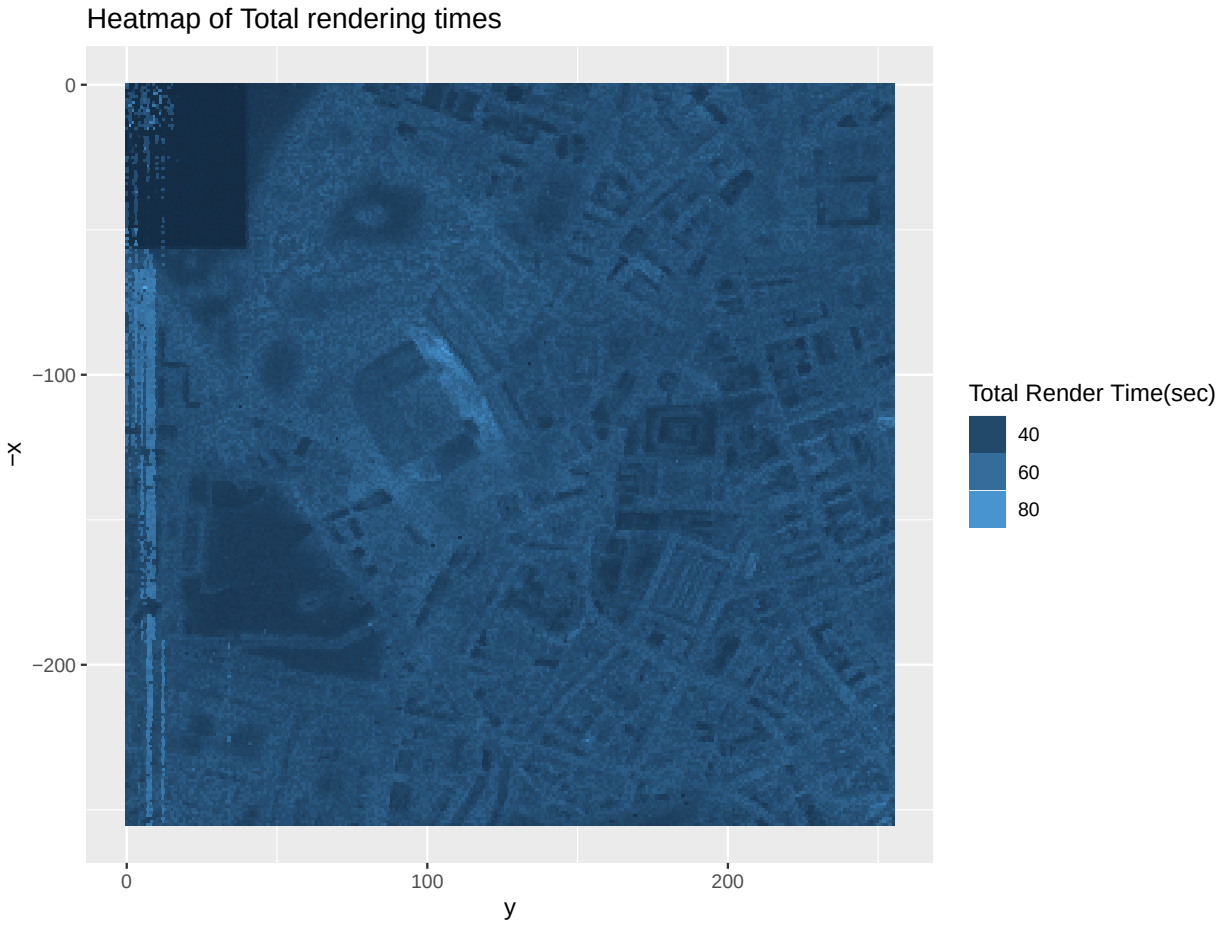
- The dataset consists of 3 levels; 4, 8 and 12 in the data as the intermediate level are derived in the tiling process. 4 being the top level and 12 being the zoomed in level.
- There are a total of 65,793 tasks required to render the tera pixel image.

Analysis of the Data

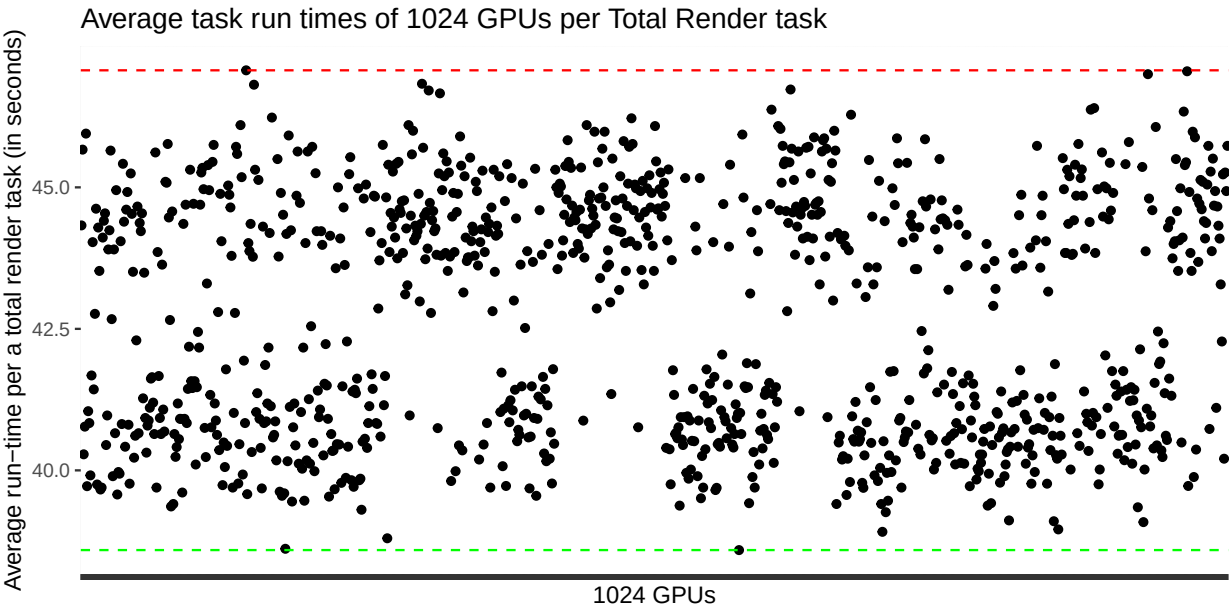
The prime focus of the analysis here has been on identifying how different GPUs have performed with the graphical summaries supporting the argument accordingly.



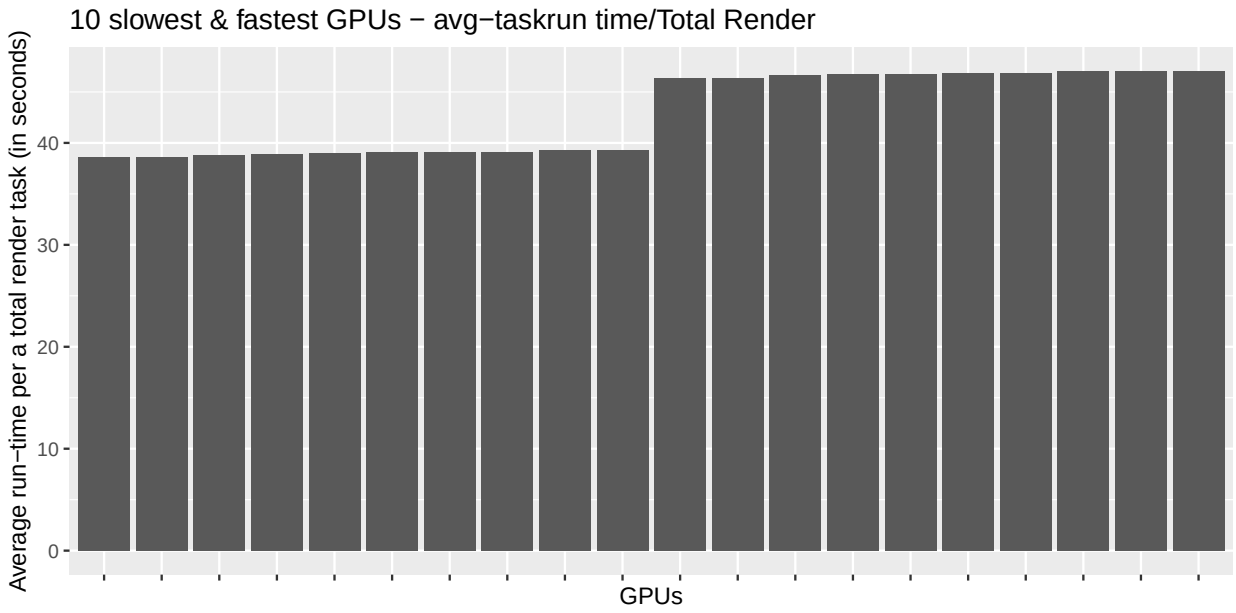
The above analysis employs the Box plot to represent each EventNames based on the time difference, when the particular event has completed its process. The boxplot provides a strong evidence to support the argument that of all the event types, maximum run time is consumed in rendering the tiles in contrast to the events tiling, uploading and saving config adding up to only a fraction of the total run time or the total render time of the task. The box plot shows a mean time of around 40 seconds for a render event whereas the outliers extend to a minimum of 20 seconds to a maximum of 80 seconds. The outliers in the uploading event are considerable which shows that there are GPUs that are taking longer to upload the processed output to the azure cloud.



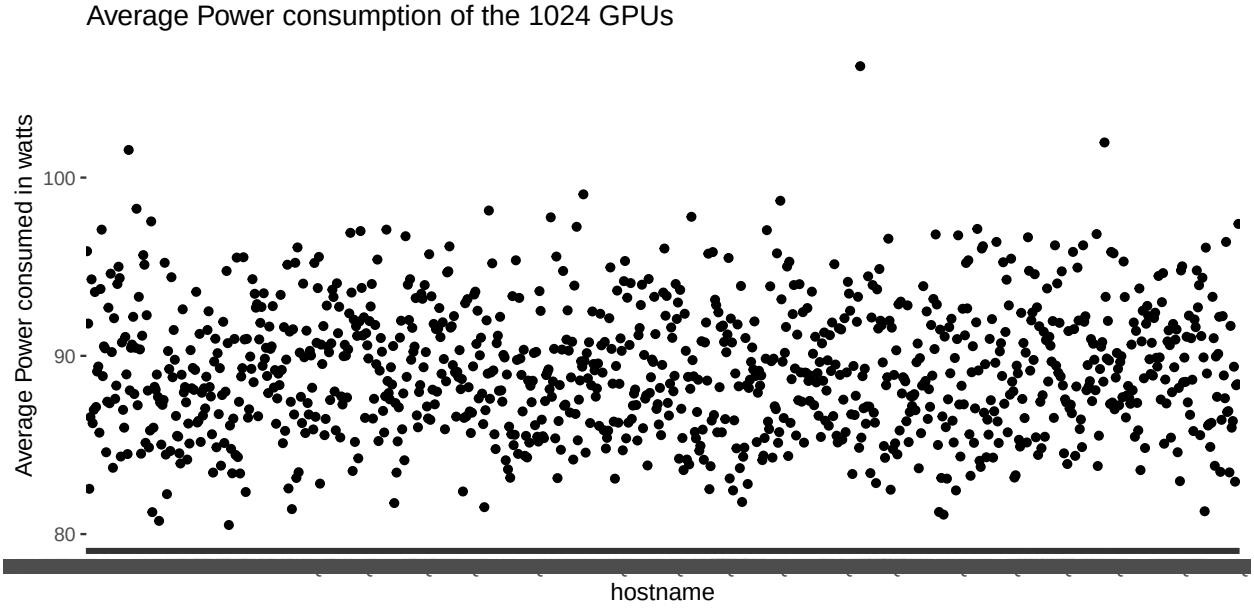
This heat map provides the stake holder with the basic insights as to which tiles of the image take longer time to render, which allows for a further and deeper investigation in to what variables influence this delayed render times.



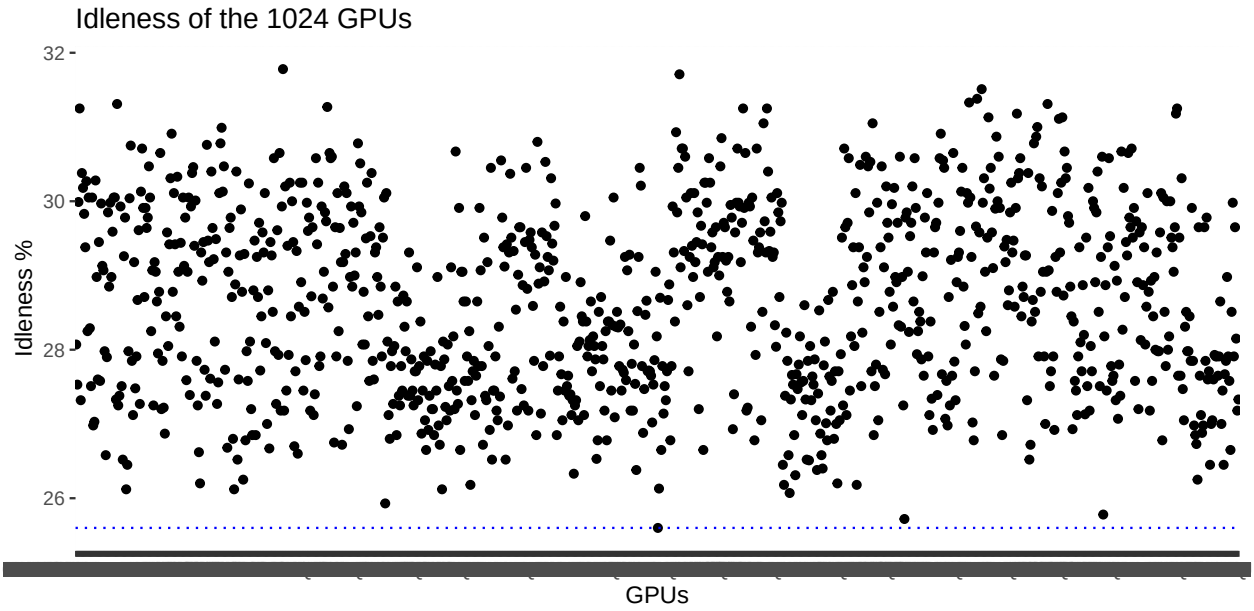
This scatter plot shows the average run time taken by each of the 1024 GPUs to complete one total rendering event i.e., to render, saving config, tiling and uploading of a tile at level 12. The green line at the bottom of the plot shows the fastest average run time of less than 40seconds to finish a total render task where as the red line on the top of the plot shows the highest average run time for a total render task to finish being more than 45seconds. Precise values can be seen from this next plot. It is also clear from the graph that a majority of the GPUs take about 40 or 45 seconds on average to complete one total render event.



The bar graph focuses only on the GPUs with the fastest and the slowest average times taken to complete the total rendering. The plot also shows the average time taken in seconds for the completion of the task with the fastest being 38 seconds and the slowest average being 47 seconds.



The plot shows the average power consumed by each individual GPU over the entire run time. It's clear that most of the GPUs consume an average power of around 90 watts, precisely ranging between 85 - 95 watts with very few outliers drawing more than 100 watts power.



A GPU is said to be 100% efficient if its continuously under use with out being idle through out the run time, but the analysis done on the GPU dataset shows other wise, i.e., the above plot shows the GPUs being idle about 26 to 32% of their complete run time. This result has been obtained from the total runtime of each GPU and the number of instances they were found idle or with a GPU utilisation percentage of zero.

Business Solution

The aim of this project was to analyse the available data and propose ways to improve the efficiency of the supercomputer in rendering the terapixel image. The results suggest focus to be put on two areas: power consumption and idleness.

- Replacing the GPUs that consume high power effectively improves the efficiency of the super computer in rendering the tera pixel.

Idleness is a key factor to be taken into consideration when dealing with processors. Having a high idle time refers to squandering of valuable computing resources.

- The results suggest that there is a great need in improving the task scheduling process of the super computer as 26% - 32% of idleness is a wastage of great deal of the computing power.

Reproducibility

- Project template is used to pre-process the data, generate graphical summaries, and report the same.
- Structure of data should be taken care before loading the project
- When the R markdown is run, it produces the necessary graphs and summary report based on the data.

Points to focus

In addition to the above analysis, further investigation can be launched into the following aspects:

- The huge time difference in the mean uploading time and its outliers.
- The factors influencing variable render times for different tiles.
- All the 1024 GPUs forming two large clusters(around 40 and 45 seconds) in the average task run times per total render task.

Conclusion

The data shows that the super computer rendering a tera pixel image compatible for viewing in any device irrespective of its computational power requires high energy. To improve the efficiency of this super computer, the primary focus should be on improving the task scheduler which directly influences the efficiency of the super computer.