Performance evaluation of Terapixel rendering in Cloud (Super)computing

Peiqiang Li - 200987503

Introduce

"Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services"[1].Cloud computing concentrates computing and storage at the heart of the service, and high-bandwidth connections are adopted to link high-performance machines, so that all these resources could be carefully managed. The user sends a computing request to the cloud computing service and obtains the final result[2]. In this project, Cloud (super)computing is used for high quality terapixel visualization. The visualization of trillions of pixels requires powerful computing resources. Therefore, cloud-based supercomputer resources provide greater flexibility for visualization.

Methodology

The structure of the project is built by **ProjectTemplate**. As we all know, the correctness and reproducibility of data science code is very important. Thus, an appropriate project template can not only improve the efficiency of development, but also make the code and documents more concise and orderly.

Git is used as a version control tool for this project. All development activities are being recorded using Git and stored in a private repository. Due to the size and privacy of dataset, the raw data will not be uploaded to the GitHub repository.

Rstudio is the development tool of this project, The project process adopts the data exploratory analysis steps of **CRISP-DM** model. And, the literate programming framework is used to record the results of data analysis and generate the final project report.

README file will be used to record the analysis steps of the project and the reproduction process of the analysis.

Business Understanding

The realistic terapixel visualization of the city of Newcastle upon Tyne is calculated by cloud-based supercomputer resources. In addition, the rendered 3D visual city also supports daily updates, hence it is necessary to evaluate the scalable architecture of cloud supercomputer for optimal performance.

The image rendering process is mainly completed by GPU. Therefore, the temperature, performance and power draw of GPU and the efficiency of computer task scheduling will have a great impact on the rendering efficiency. In order to improve the computing power of supercomputers, the analysis of the relationship between GPU and rendering tasks will be particularly important.

The project detail see link.

Data Understanding

The dataset collected from the project link above. It contains the running information of 1024 GPU nodes. The running task is divided into three parts, which are used to render the visualization output of level 4, 8 and 12 respectively. In this dataset we can get detailed information about the performance timings of the render application, the performance of the GPU card, and which part of the image is being rendered in each task. The size of the dataset, the number of fields and the specific explanations are shown in the following tables:

The application.checkpoints dataset contains 660400 data and 6 fields.

	Fields	Info
1	timestamp	Timestamp
2	hostname	Hostname of the virtual machine auto-assigned by the Azure batch system.
3	eventName	Name of the event occuring within the rendering application.
4	eventType	The "START" and "STOP" type of events.
5	jobId	ID of the Azure batch job(corresponding to three job tasks).
6	taskId	ID of the Azure batch task.

Table 1: dataset information of application.checkpoints

The explain of `eventName` values:

- **TotalRender** is the entire task duration(sum of other events running time).
- **Render** is when the image tile is is being rendered.
- **Saving Config** is simply a measure of configuration overhead.
- **Tiling** is where post processing of the rendered tile is taking place.
- **Uploading** is where the output from post processing is uploaded to Azure Blob Storage.

examples:

```
## # A tibble: 6 x 6
##   timestamp           hostname      eventName  eventType jobId        taskId
##   <dttm>              <chr>         <chr>      <chr>     <chr>        <chr>
## 1 2018-11-08 07:41:55 0d56a7300766~ Tiling     STOP      1024-lvl12-~ b47f0263-~
## 2 2018-11-08 07:42:29 0d56a7300766~ Saving Co~ START     1024-lvl12-~ 20fb9fcf-~
## 3 2018-11-08 07:42:29 0d56a7300766~ Saving Co~ STOP      1024-lvl12-~ 20fb9fcf-~
## 4 2018-11-08 07:42:29 0d56a7300766~ Render     START     1024-lvl12-~ 20fb9fcf-~
## 5 2018-11-08 07:43:13 0d56a7300766~ TotalRend~ STOP      1024-lvl12-~ 20fb9fcf-~
## 6 2018-11-08 07:43:56 0d56a7300766~ Render     STOP      1024-lvl12-~ 3dd4840c-~
```

The GPU dataset contains 1543681 data and 8 fields.

	Fields	Info
1	timestamp	Timestamp
2	hostname	Hostname of the virtual machine auto-assigned by the Azure batch system.
3	gpuSerial	The serial number of the physical GPU card.
4	gpuUUID	The unique system id assigned by the Azure system to the GPU unit.
5	powerDrawWatt	Power draw of the GPU in watts.
6	gpuTempC	Temperature of the GPU in Celsius
7	gpuUtilPerc	Percent utilisation of the GPU Core(s).
8	gpuMemUtilPerc	Percent utilisation of the GPU memory.

Table 2: dataset information of gpu.csv

examples:

```
## # A tibble: 6 x 8
##   timestamp           hostname      gpuSerial gpuUUID     powerDrawWatt gpuTempC
##   <dttm>              <chr>             <dbl> <chr>               <dbl>    <int>
## 1 2018-11-08 08:27:10 8b6a0eebc87b~   3.23e11 GPU-1d1602~         132.        48
## 2 2018-11-08 08:27:10 d8241877cd99~   3.24e11 GPU-04a2de~         117.        40
## 3 2018-11-08 08:27:10 db871cd77a54~   3.23e11 GPU-f45979~         122.        45
## 4 2018-11-08 08:27:10 b9a1fa7ae2f7~   3.25e11 GPU-ad773c~          50.2       38
## 5 2018-11-08 08:27:10 db871cd77a54~   3.23e11 GPU-2d4eed~         142.        41
## 6 2018-11-08 08:27:10 265232c5f681~   3.24e11 GPU-717654~         120.        43
## # ... with 2 more variables: gpuUtilPerc <int>, gpuMemUtilPerc <int>
```

The task dataset contains 65793 data and 5 fields and the specific information of **task-x-y.csv**:

- **jobId**: Id of the Azure batch job.
- **taskId**: Id of the Azure batch task.
- **x**: X co-ordinate of the image tile being rendered.
- **y**: Y co-ordinate of the image tile being rendered.
- **level**: The visualisation 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.

examples:

```
## # A tibble: 6 x 6
##   taskId                               jobId           x     y level hostname
##   <chr>                                <chr>       <int> <int> <int> <chr>
## 1 00004e77-304c-4fbd-88a1-1346ef947567 1024-lvl12~   116   178    12 0745914f4d~
## 2 0002afb5-d05e-4da9-bd53-7b6dc19ea6d4 1024-lvl12~   142   190    12 83ea61ac1e~
## 3 0003c380-4db9-49fb-8e1c-6f8ae466ad85 1024-lvl12~   142    86    12 83ea61ac1e~
## 4 000993b6-fc88-489d-a4ca-0a44fd800bd3 1024-lvl12~   235    11    12 cd44f5819e~
## 5 000b158b-0ba3-4dca-bf5b-1b3bd5c28207 1024-lvl12~   171    53    12 8b6a0eebc8~
## 6 000d1def-1478-40d3-a5e3-4f848daee474 1024-lvl12~   179   226    12 b9a1fa7ae2~
```

Data Preparation

At the First, We need to make sure that the dataset is not affected by duplicate data. We found that the dataset of application.checkpoints have 2470 duplicate data, and the gpu dataset have 9 duplicate data. After remove the duplicate data the following information can be summarized.

1. The task.x.y dataset contains 65793 data.
    - These all 65793 tasks is undering the 3 jobs.
2. The application.checkpoints dataset contains 657930 data.
    - every task have ten events: "Tiling", "Saving Config", "Render", "TotalRender", "Uploading". Each event have "START" and "STOP" two types.
3. the gpu dataset contains 1543672 data.
    - every gpu have a unique `hostname`, `gpuSerial` and `gpuUUID`.

```
unique(task.x.y$jobId)
# find how many gpu render this level
length(unique(task.x.y[task.x.y$level==12,]$hostname))
length(unique(task.x.y[task.x.y$level==4,]$hostname))
length(unique(task.x.y[task.x.y$level==8,]$hostname))
```

The 3 `jobId` corresponding to the rendering of 3 levels.

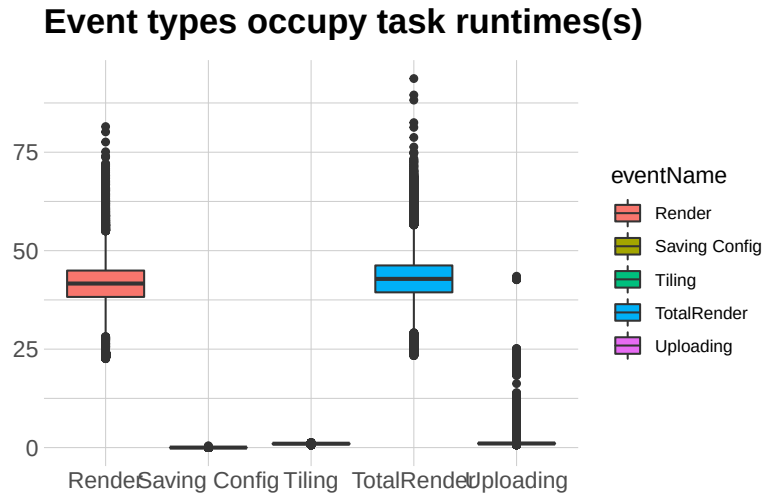
- **jobId** "1024-lvl12-7e026be3-5fd0-48ee-b7d1-abd61f747705" is the rendering of level 12. This job has 65536 task records, and 1024 gpu rendering.
- **jobId**" 1024-lvl4-90b0c947-dcfc-4eea-a1ee-efe843b698df"is the rendering of level 4. This job has 1 task records, and 1 gpu rendering.
- **jobId** "1024-lvl8-5ad819e1-fbf2-42e0-8f16-a3baca825a63" is the rendering of level 8. This job has 256 task records, and 256 gpu rendering.

Data Analysis

Before expanding the analysis task. We have did some calculations on the dataset. We found the duration of each task and its event, the gpu metrics information of each task, and find the GPU running state of specific task duration.

The event types dominate task runtimes

A complete rendering process consists of four parts: `Saving Config`, `Render`, `Tiling` and `Uploading`. This four events of each rendering task, as well as their start and stop times, are recorded in the `application.checkpoints` dataset. We can find out which event dominates the task by comparing the duration of these four events in each task with the total rendering of this task. After calculating the duration of each event, we can compare the duration by boxplot.

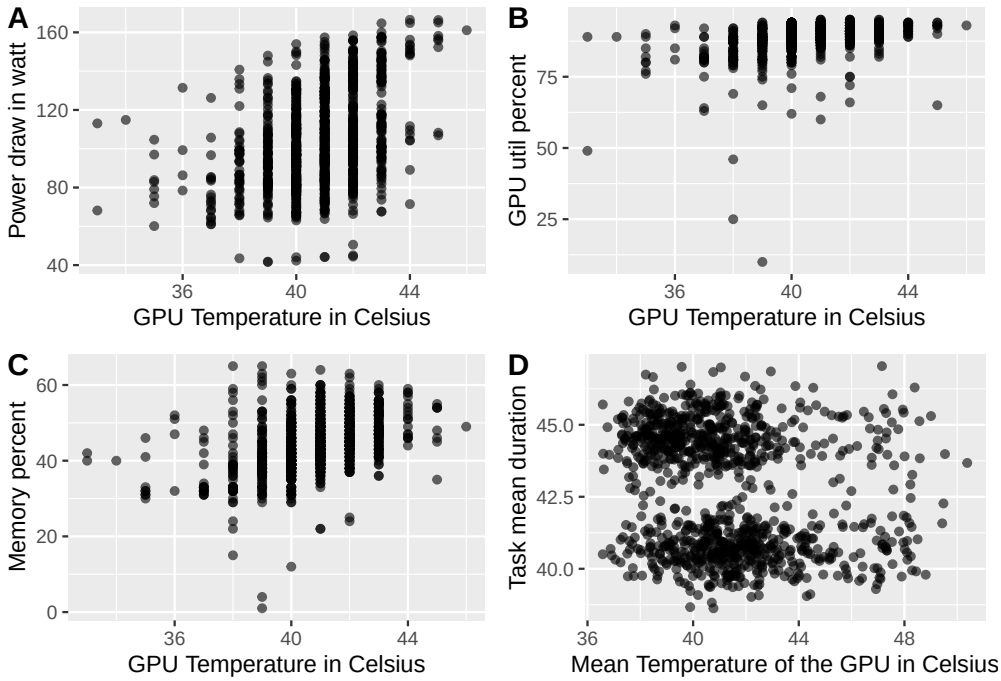


As we can see from the boxplot above, the running time of rendering occupies almost the duration of the whole task. Also, we can see a large number of outliers around the boxplot. That means the rendering time between different tasks will be quite different, and the distribution of rendering time is irregular.

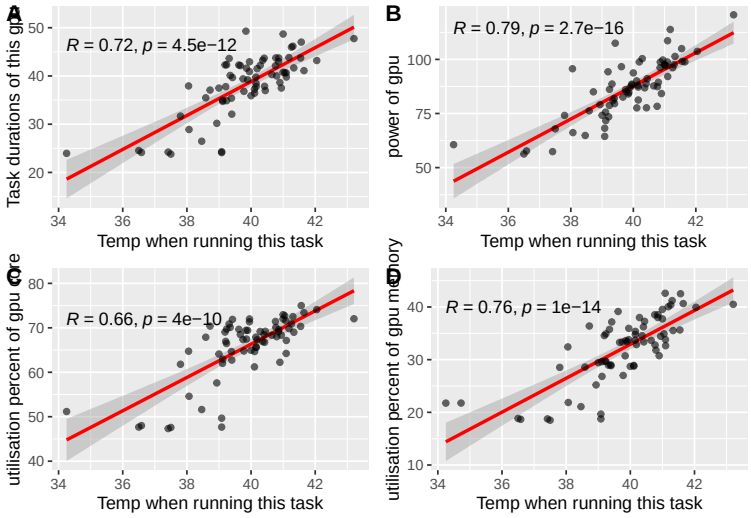
Interplay between GPU temperature and performance

Before we look for the relationship between GPU temperature and performance, it is necessary to find a good metric to measure the performance of GPU. In my opinion, the average execution time of each task by GPU is a good measure of its performance. The average task rendering duration comes from the number of tasks rendered by the GPU and the sum of the duration of all these tasks. In addition, temperature, power draw, core utilization and memory utilization of GPU also can be used to measure the performance of GPU.

The relationship between these metrics and GPU operating temperature is shown in the figure below:



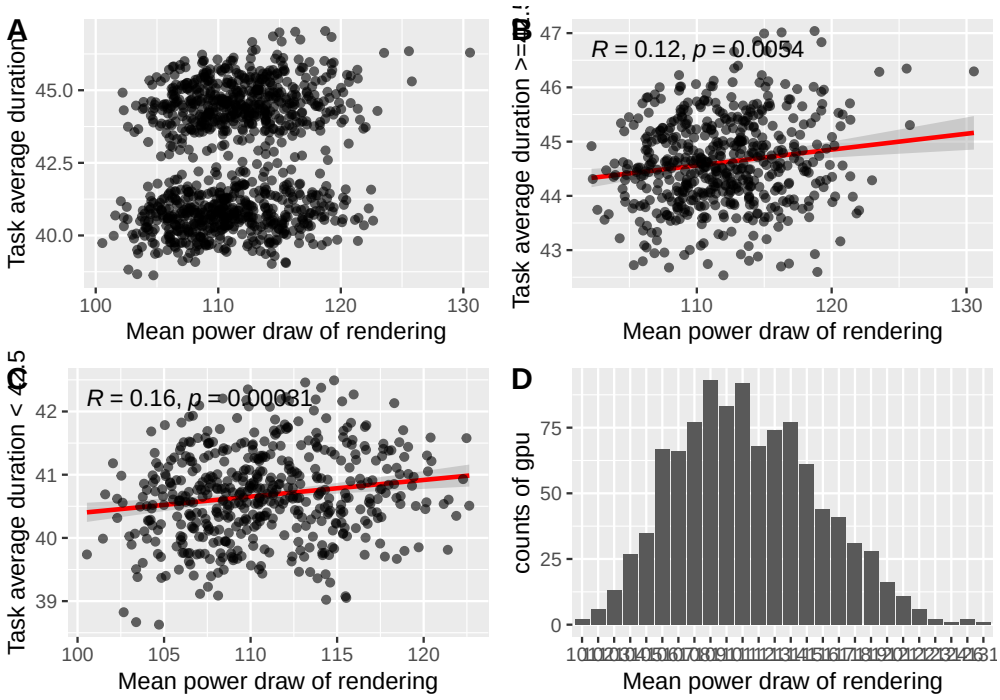
As a result, based on these four figures, there is no obvious correlation between GPU temperature and GPU performance. However, these four figures above is showing the average situation of all GPUs, so it's a good idea to find the relationship between GPU temperature and other metrics in one particular gpu. Since the GPU is at a low temperature at the beginning of the first rendering task, the impact of the first rendering task on the statistical results will be removed here.



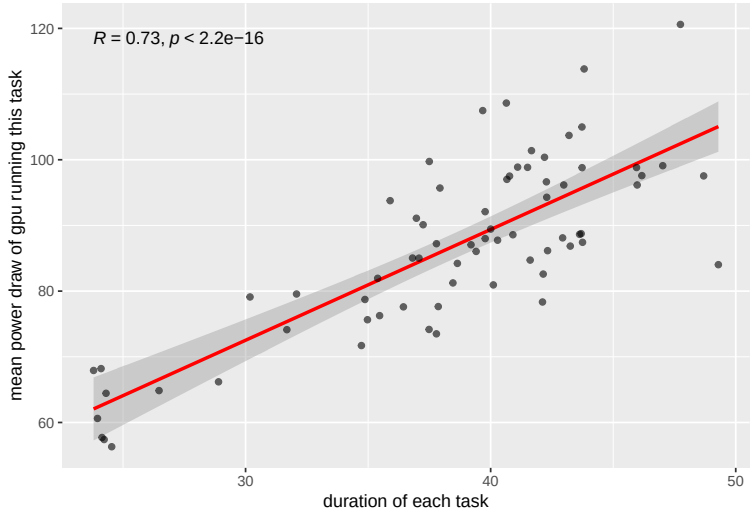
Obviously, there is a strong linear relationship between GPU temperature and GPU power draw, core utilization and memory utilization. When the computing performance of GPU is enhanced, the temperature of GPU will also increase.

Interplay between increased power draw and render time

As before, firstly, we observe the relationship between the average rendering time and the average power draw of certain GPU. A GPU is performing dozens of rendering tasks, The power draw is the average power draw when GPU rendering these dozens of tasks. And the duration is the average duration of these tasks.



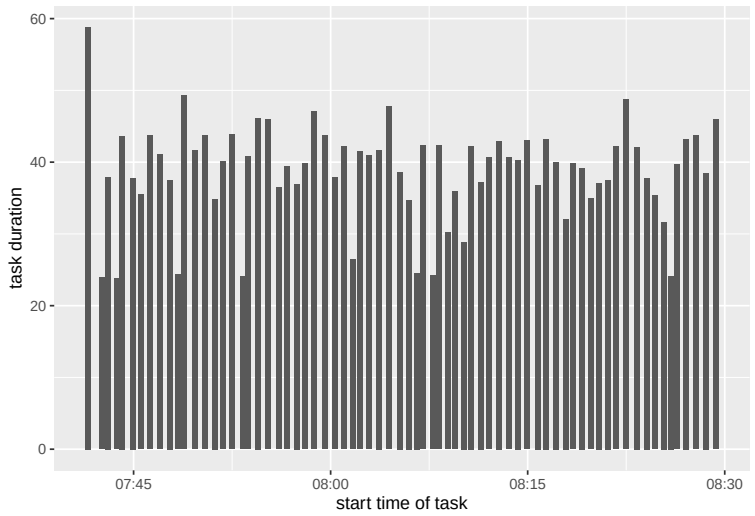
From the above figures, we can not find obviously relationship between pwer draw and task duration. But, we found the power draw of most GPUs are between 103-121 watts. Then, we take one gpu as an example, and find the relationship of durations and power draw.



Consequently, the longer the duration of the task, the higher its average power draw. There is a positive relationship between rendering duration and gpu power draw.

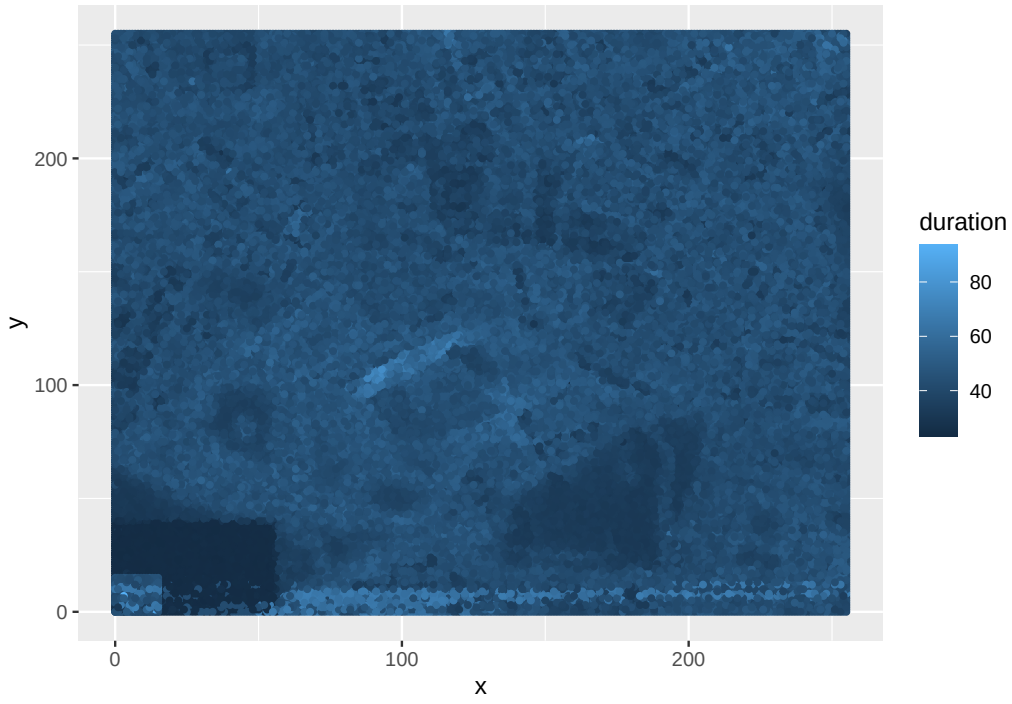
variation in computation requirements for particular tile

When we take one GPU as an example, the duration of each task rendered by this GPU is distributed as shown in the figure below.



Due to each task corresponds to the rendering of a specific tile. The duration of the task explains the changes in the calculation requirements of a specific tile. The outcomes tell us that, the duration of each task rendering by the gpu can be very different, the rendering time of each tile is also different.

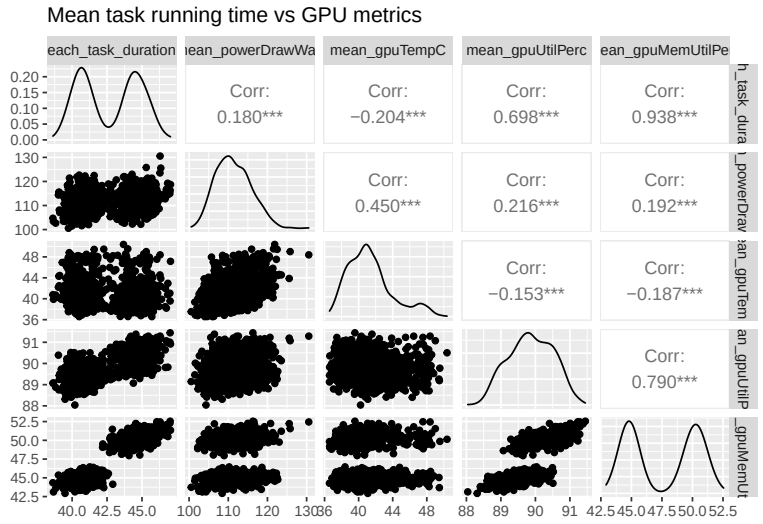
After the rendering time required for each tile corresponds to the coordinates of the pixel, we can get the following figure:



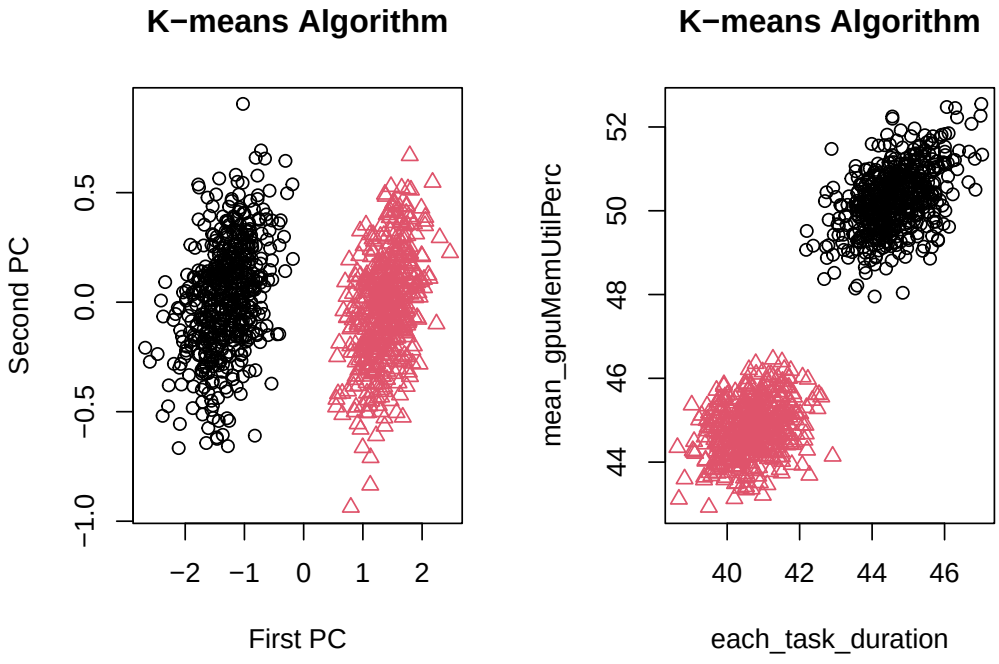
The outline of the Newcastle city map can be clearly observed. Some blocks are significantly darker than others, because tiles have the same true color are easy to render, tiles that contain building edge details take longer to render. This is why the calculation requirements of each task are different. In addition, there is a darker block in the lower left corner, which belongs to level 8 and is rendered by 256 GPUs.

Identify particular GPU cards

If there are performance differences between GPU cards, there will be some differences in the distribution of parameters between them. Thus, through the GPU running temperature, power draw, core utilization and memory utilization, we can find the GPU with significant differences.



Obviously, GPUs are divided into two categories by task duration and average memory utilization. The duration here refers to the average time each GPU takes to complete a task, the memory utilization refers to the average memory utilization percent by GPU in running these tasks. By using these two data, a clustering analysis of gpu can be carried out.



As a result, Whether in the PCA graph or the memory utilization vs task duration graph, The GPUs are clearly divided into two parts. Then, we are trying to identify the performance of these GPUs by their serial numbers. Firstly, the results of cluster analysis will be used as the category identification of GPUs, and The serial number will be converted to numeric type for discriminant analysis.

```
duration_memory$classify = km_duration_memory$cluster
duration_memory$gpuSerial = as.numeric(duration_memory$gpuSerial)
head(duration_memory[,c(1,4)])
```

```
##      gpuSerial classify
## 1 323217056165        1
## 2 323617042956        1
## 3 323617021222        2
## 4 323617021168        2
## 5 323217056664        1
## 6 323617020234        2
```

LDA:

```
##
##      1  2
##   1 42 11
##   2 44  3
```

924 GPU serial numbers are used as the training set, and the remaining 100 GPU serial numbers are used as the test set. The linear discriminant analysis was fitted to classify. It can be seen from the confusion matrix that the results are not satisfactory. Then, the quadratic discriminant analysis model and Logistic regression models were also used for classifying.

QDA:

```
##
##      1  2
##   1 42 11
##   2  8 39
```

From the confusion matrix, we can see that most GPUs are correctly classified. The accuracy is 81%.

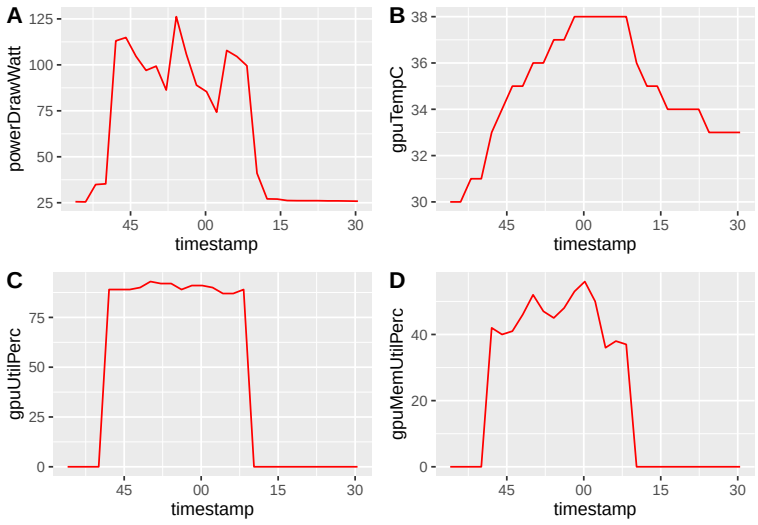
logistic regression:

```
##    pred
##      0  1
##   0 42 11
##   1 44  3
```

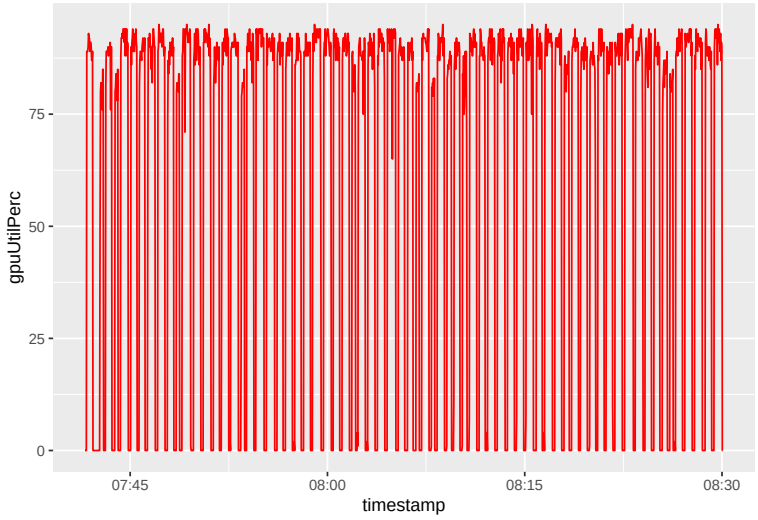
From the confusion matrix that the results are also not satisfactory.

What can we learn about the efficiency of the task scheduling process

To understand the task scheduling process of GPU, it is useful to clarify the running cycle of each task. Generally speaking, the start and end time of each task is the task scheduling process of GPU. We can find this process by giving an example of task. The execution cycle of a task is as follows:



The figure above shows the GPU status when a task is executed. The core utilization of the gpu is a very good indication that the gpu is starting to actually perform the task. For this task, about 40% of the duration is in the process of task scheduling. In order to understand more clearly the task scheduling and the real execution time of the task, we can plot all tasks vs timestamp rendering by this GPU.



We can clearly see the gaps between each task and the time spent on task scheduling. After calculating, the total time of task scheduling accounts for about 26% of the time that the GPU executes all tasks. Therefore, if we can find a way to improve the efficiency of GPU task scheduling, it will shorten the overall rendering time to a great extent.

Conclusoin

To sum up, in this project, We explored those factors that affect GPU rendering performance, we analyzed the relationship between the GPU metrics and adopted some classification models to identify particular GPU cards. We have illustrated the relationship between the GPU metrics very well by means of general and example analysis. Finally, in order to optimize the rendering performance of this architecture, it is a good approach to replace the poorly performing graphics card and improve the efficiency of gpu task scheduling.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M., 2010. A view of cloud computing. Communications of the ACM, 53(4), pp.50-58.

[2] Hayes, B., 2008. Cloud computing.