Performance evaluation of Terapixel rendering

UNPKalyan

26/01/2021

##Introduction

Terapixel pictures offer an natural, open way to show data sets to partners, permitting watchers to intuitiveness browse enormous information over different scales. The challenge we tended to 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. Our arrangement may be a scalable engineering for cloud-based visualization that we will deploy and pay for as it were as required. The three key goals of this work are 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. We illustrate that it is possible to deliver a high-quality terapixel visualization employing a way following renderer in beneath a day utilizing open IaaS cloud G

About Data

A run using the 1024 GPUs resulted in the creation of the dataset which shows the performance timing of the render application, performance of the GPU card, and the details of which part of the image of the was being rendered in each task.

```r
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

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

```r
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.

```r
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
```

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. You will only see levels 4, 8 and 12 in the data as the intermediate level are derived in the tiling process.

Data Preprocessing

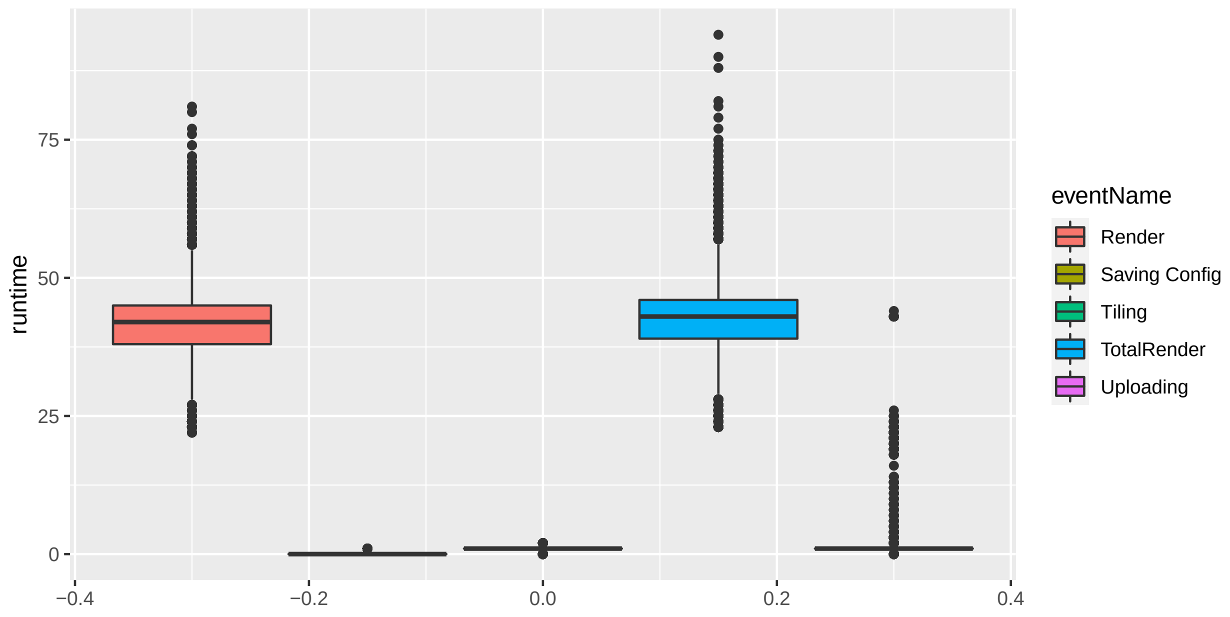
Data Preprocessing is a crucial step before proceeding with any analysis as presence of outliers, duplicate and wrong data affects the quality of the analysis' outcome.

- All the available datasets are loaded and renamed accordingly for the ease of usability.
- Removing duplicate entries from the datasets
- Application checkpoints dataset and Tasks dataset both are merged and used to calculate the individual run times of each event, also linking the x and y tile values with their task ids and GPUs.

Analysis of the Data

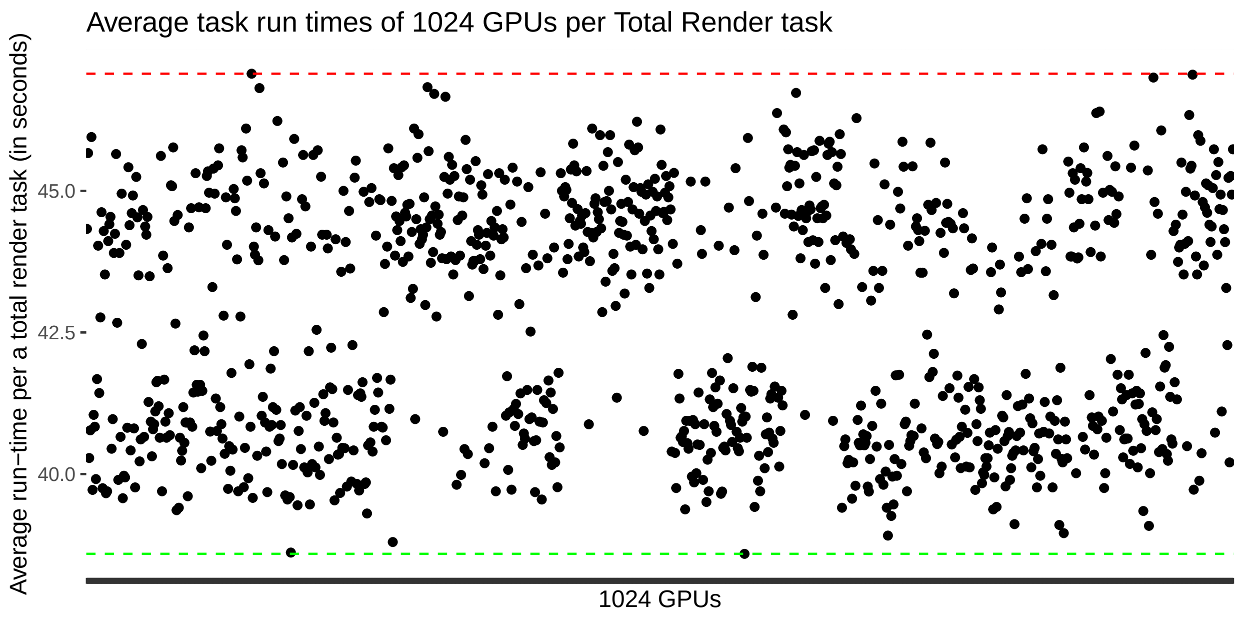
The prime focus of the analysis here has been on identifying how few GPUs have performed differently with respect to rest of the bunch, with the graphical summaries supporting the argument accordingly.

1. Which event type dominates task run-times?

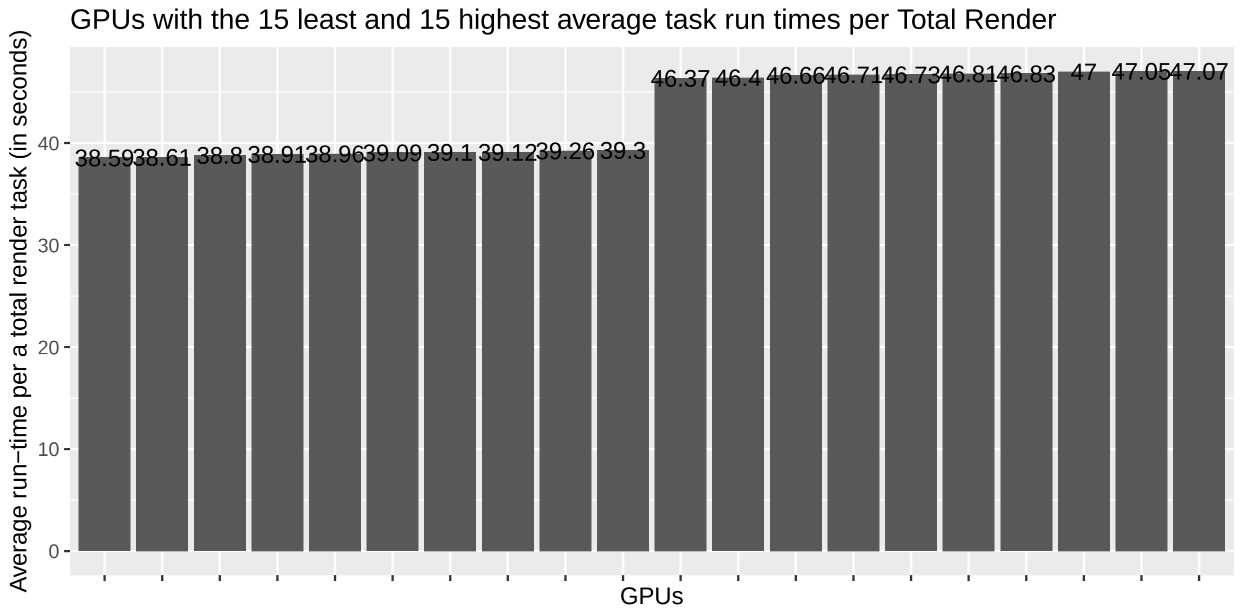


The boxplot provides a strong evidence to support the argument that of all the event types, maximum run time is consumed in rendering the tiles in contrast to the events tiling, uploading and saving config adding up to only a fraction of the total run time or the total render time of the task.

2. Identifying particular GPU cards that are slower or faster than the rest of the cards?

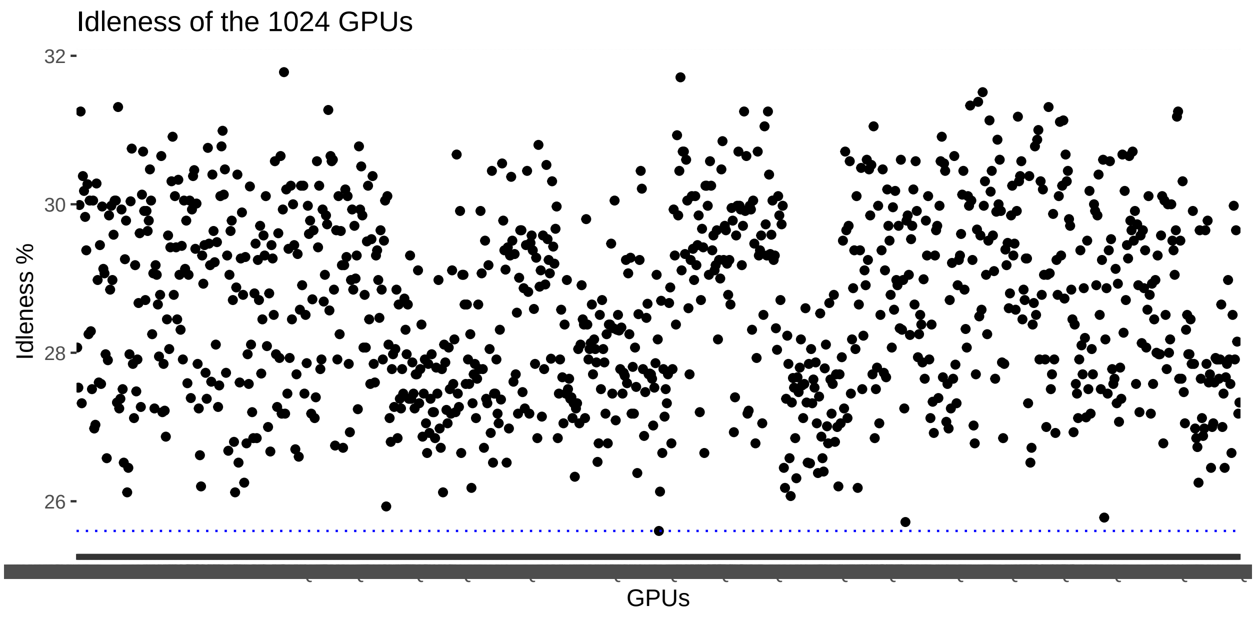


This scatter plot shows the average run time taken by each of the 1024 GPUs to complete one total rendering event i.e., to render, saving config, tiling and uploading of a tile at level 12. The green line at the bottom of the plot shows the fastest average run time of less than 40seconds to finish a 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.



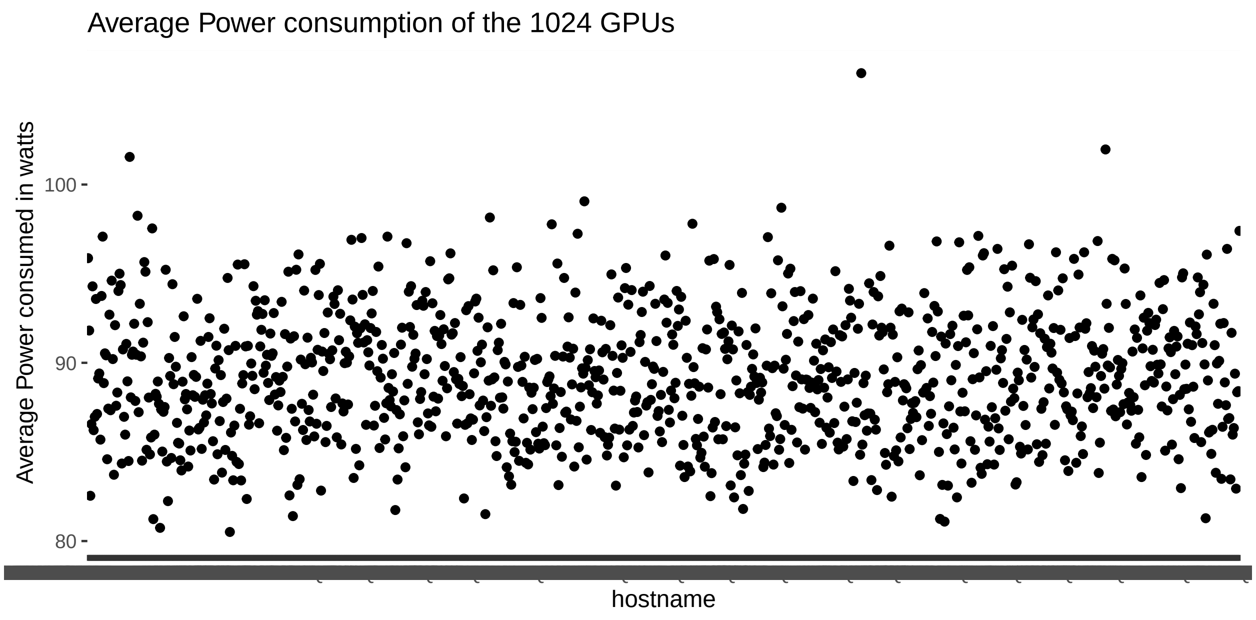
The bar graph focuses only on the GPUs with the fastest and the slowest average times taken to complete the total rendering. The plot also shows the average time taken in seconds for the completion of the task with the fastest being 38.59 seconds and the slowest average being 47.07 seconds.

3. What can we learn about the efficiency of the task scheduling process?



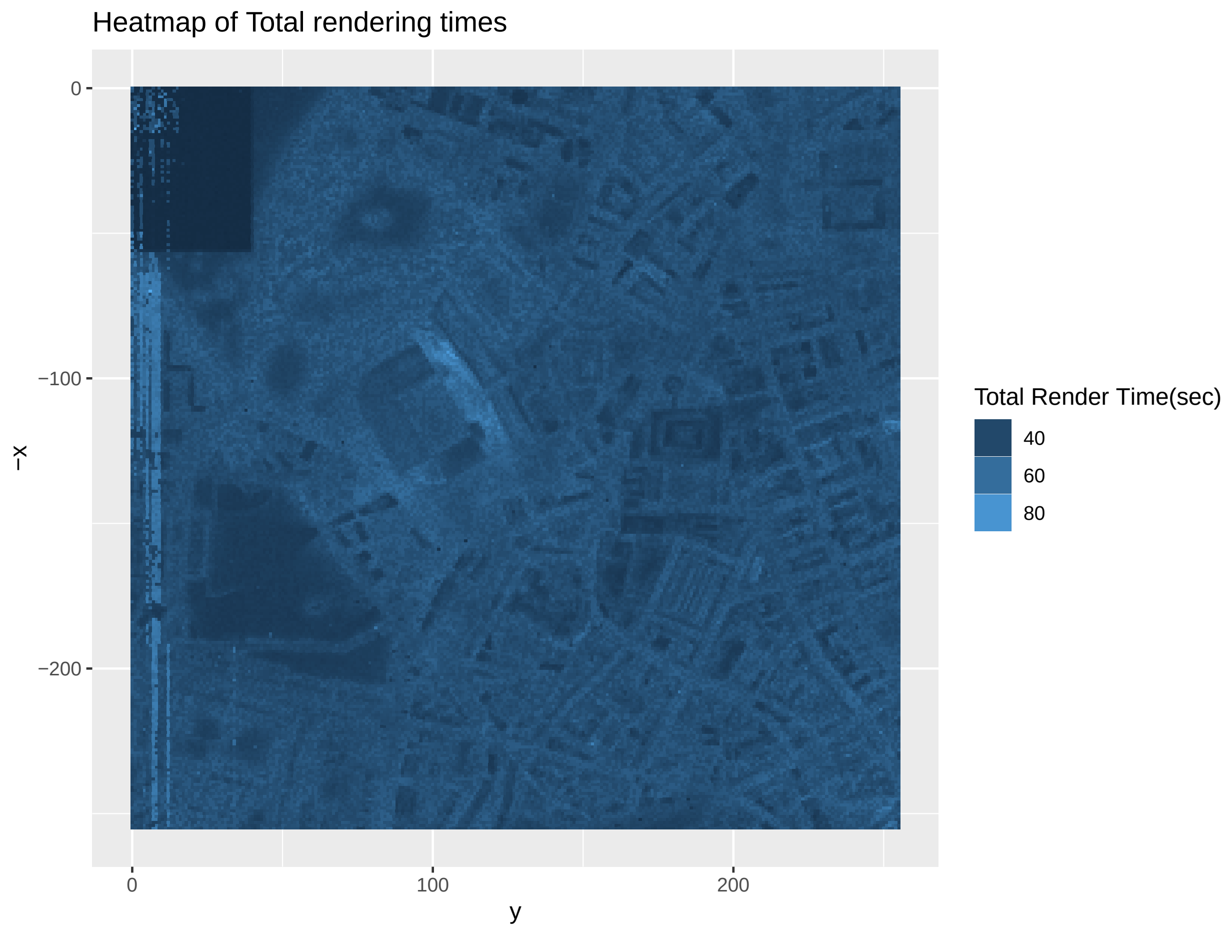
A GPU is said to be 100% efficient if its continuously under use with out being idle through out the run time, but the analysis done on the GPU dataset shows other wise, i.e., the above plot shows the GPUs being idle about 26 to 32% of their complete run time. This result has been obtained from the total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

4. What can we say about the power consumption of the GPUs?



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.

5. Overview of the Tera pixel image's rendering times.



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.

Report 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.