CSC8634 Cloud Computing Report

Peter Bevan

18/01/2021

Business Understanding

This report documents an exploratory data analysis of application checkpoint and system metric output data collected during the rendering of a 3d terapixel image of the city of Newcastle, UK. A terapixel is defined as 10^12 pixels (1,000,000 megapixels), and so are extremely large images and therefore very computationally costly to produce. A path tracing renderer coupled with a scalable public cloud supercomputing architecture was successfully used to render the terapixel image in under a day using 1024 public cloud GPU nodes. The completed terapixel image is accessible for users to explore the city in 3D with zoomable scales ranging from a complete overview to minute detail.

Since the rendering of such large images is extremely demanding, optimization of the architecture and process is essential for keeping the time frames manageable. Since the rendering is carried out on the public cloud on a pay as you go basis, this optimization also helps to keep costs down, especially if continuous updating of the visualization is to take place. One way to increase the rendering speed is to select the specific GPU cards which perform best for the required task.

The data collected during the initial rendering can be analysed to view the performance of each type of GPU card in order to decide on the optimal cards for the task, as well as revealing which GPUs seem to be unsuitable for the task. The data can be manipulated to allow easy comparison of GPU performance, followed by dimensionality reduction techniques and clustering to separate the desirable and undesirable GPU cards.

The tools used in the project were Rstudio and Rmarkdown combined with various packages (see config file for details on packages). ProjectTemplate was used to organise and link the files, and Git/Github was used for version control.

Data Understanding

The data was downloaded directly from the Newcastle data science student projects GitHub repository, which is a private repository. The data comprises 3 large datasets: `application.checkpoints`, `gpu` and `task.x.y`. The shape of this data is shown below:

```
dim(application.checkpoints)
```

```
## [1] 660400      6
```

```
dim(gpu)
```

```
## [1] 1543681       8
```

```
dim(task.x.y)
```

```
## [1] 65793     5
```

It's clear that the data is very large, with over 650 thousand observation in the `application.checkpoints` dataset across 6 variables, over 1.5 million observations in the `gpu` dataset across 8 variables, and over 65 thousand observations across 5 variables in the `task.x.y` dataset. Each dataset is summarised below:

application.checkpoints

The `application.checkpoints` data is a comprehensive record of timestamped checkpoint events from the rendering process, with one timestamped entry for the start of the event and one for the end. The checkpoint events are as follows (taken from the `eventName` column): + `TotalRender`: entire rendering task (excluding tiling) + `Render`: rendering of image tile + `Saving Config`: configuration overhead + `Uploading`: uploading of output to Azure Blob Storage + `Tiling`: post processing of rendered tile

```
#printing head of application checkpoint data
head(application.checkpoints)
```

```
## # 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~
```

gpu

The `gpu` data comprises periodical recording of status of the GPU on each virtual machine. Recorded GPU metrics are as below, which are paired with the timestamp of the measurement, the hostname of the virtual machine, the GPU card serial number and the system ID assigned to the GPU card: + `powerDrawWatt`: power draw of the GPU card in Watts + `gpuTempC`: temperature of GPU in Celsius at specified timestamp + `gpuUtilPerc`: utilisation of the GPU **core/s** (%) + `gpuMemUtilPerc`: utilisation of the GPU **memory** (%)

```
#printing head of gpu data
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>
```

task.x.y

the x and y coordinates of the section of image being rendered for each task. `level` represents the zoom level, of the 12 zoom levels of the visualisation (12 being most zoomed in and 1 being completely zoomed out). In the dataset there is only levels 4, 8 and 12 since the other levels are derived in the tiling process.

```
#printing head of task.x.y data
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
```

Each dataset was visualised with the `naniar` library, using `vis_miss` to quickly get an overview of any missing values. None were present in any of the datasets and so the visualisations were omitted from this report. This lack of missing data is likely because the data is mostly machine collected measurements and so the data was consistently recorded.

Data Preparation

Each event type was separated from `application.checkpoints` and grouped by task ID (one group per stop and start of each task). These paired start and stop times were then converted to the duration of each task by finding the absolute difference between the two timestamps in each pair using `difftime`. This resulted in 5 new dataframes, one for each event type previously listed under `eventName`. These new dataframes were then joined by `taskId` and `hostname` to create a dataset containing a dataframe containing the runtime breakdowns for each event of each rendering task. It's clear from summing the individual event runtimes for a sample of tasks that `TotalRender` comprises all events except for `Tiling`, which is useful to know for later analysis.

Geometric mean used to summarise the percentage utilisation of the GPU memory and core(s) since these are ratios and we have no access to the costs that these ratios are based on. (Hoefler 2015)

There are 1024 different hostnames, which means 1024 different GPUs. It can be seen after filtering by level and grouping by hostname that all 1024 virtual machines in the dataset has carried out rendering on level 12, with 256 of these also rendering on level 8 and only one rendering on level 4. The sum total of the duration of the rendering for each task and level, was calculated and is shown below. This shows that the vast majority of the computing is done rendering level 12. The avenue of most value is likely in focusing analysis on this with the hope of generating insights that lead to optimisation to reduce the time/money/energy spent in the future.

```
str_glue('Summed total render runtime for level 12 accross all parallel GPUs: {as.numeric(sum(Runtimes_
```

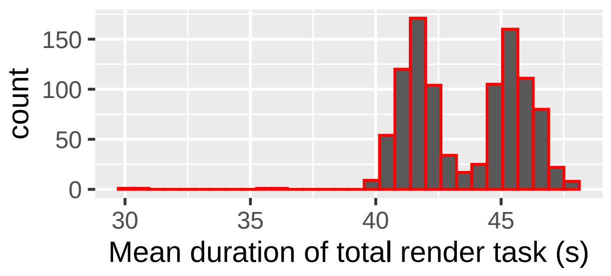
```
## Summed total render runtime for level 12 accross all parallel GPUs: 792.076728888485 hours
```

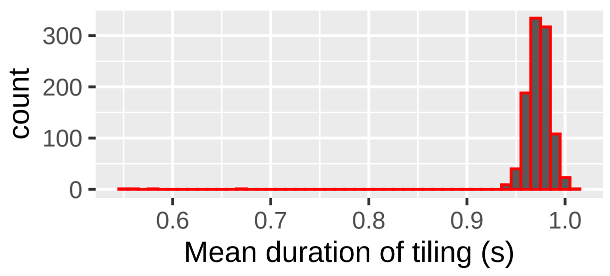
```
str_glue('Summed total render runtime for level 8 accross all parallel GPUs: {as.numeric(sum(Runtimes_8
```

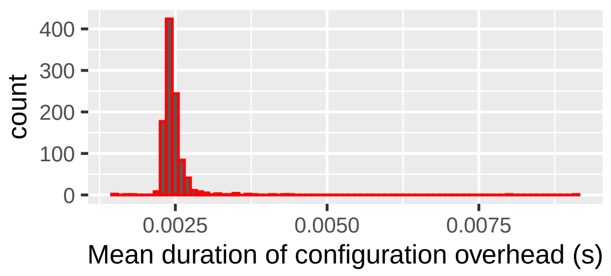
```
## Summed total render runtime for level 8 accross all parallel GPUs: 3.50675416582161 hours
```

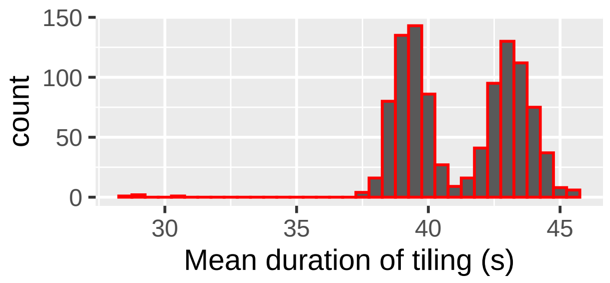
```
str_glue('Summed total render runtime for level 4 accross all parallel GPUs: {as.numeric(sum(Runtimes_4
```

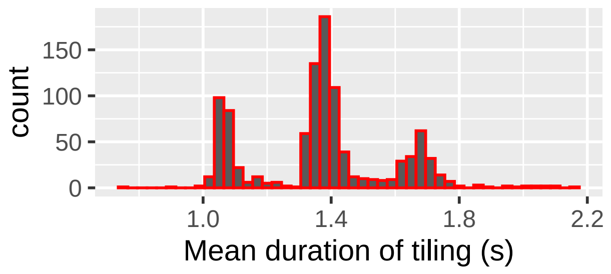
```
## Summed total render runtime for level 4 accross all parallel GPUs: 0.0148411111036936 hours
```

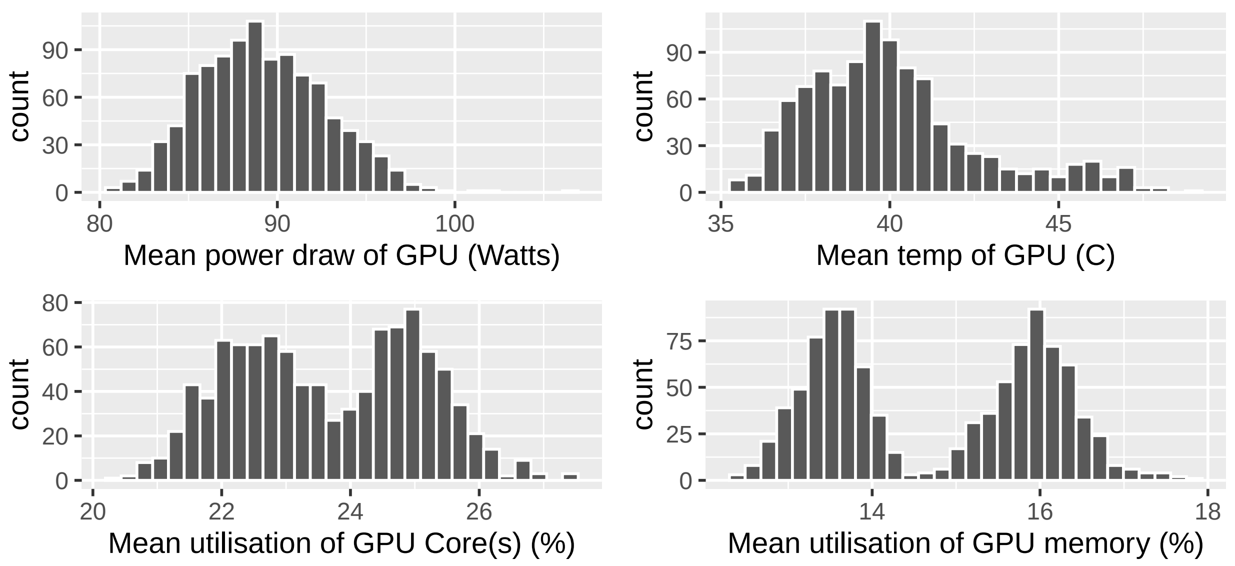

count
150
100
50
0
30
35
40
45
Mean duration of total render task (s)


count
300
200
100
0
0.6
0.7
0.8
0.9
1.0
Mean duration of tiling (s)


count
400
300
200
100
0
0.0025
0.0050
0.0075
Mean duration of configuration overhead (s)


count
150
100
50
0
30
35
40
45
Mean duration of tiling (s)


count
150
100
50
0
1.0
1.4
1.8
2.2
Mean duration of tiling (s)



The correlation matrix below was computed to visualise the correlation and investigate the relationships between pairs of variables in the newly constructed `gpu_performance` dataset. Since there are so many overlapping observations, the transparency of the points was increased to highlight areas of overlapping points and better highlight trends. It's easy to notice even in these pairwise plots there is often distinct groups of GPUs. Obvious positive linear relationships can be seen between `mean_gpuMemUtilPerc` and `mean_gpuUtilPerc` as well as between `mean_TotalRender` and `mean_Render`. There are also other potential linear relationships between other variables but it is harder to tell without further investigation. These linear relationships between at least some variables indicates that there is a chance the variables are not providing independent information. This in turn means we may be able to reduce the dimensionality of the data without losing too much of the total variation. Principle component analysis (PCA) will be carried out on the data to reduce it's dimensionality.

Since there are no target/response variables in the data, we can use an unsupervised learning technique to group the data into clusters. K-means clustering was selected since the clusters look to be highly spherical, which k-means are know to be able to differentiate well.

```
#{r} #pairs(gpu_performance_12[,-c(1:3)], col = rgb(red = 0, green = 0, blue = 0, alpha
= 0.1)) #
```

```
#{r, echo=FALSE, fig.cap="A caption", out.width = '100%'} #knitr::include_graphics("graphs/heatmap.png"
#
```

References

http://www.gigamacro.com/worlds-first-terapixel-macro-image/

hoefler scientific benchmarking

Joe mattews statistical learning

https://www.geeksforgeeks.org/difference-between-k-means-and-hierarchical-clustering/#:~:text=A%20hierarchical%20clus

https://blog.bioturing.com/2018/09/24/heatmap-color-scale/#:~:text=Color%2Dblind%20people%20tend%20to,%2Dblind%