Terascope Terapixel Deployment: Computational Analysis

Mark R. Tyrrell

23/01/2019

1. Introduction

The visualisation of multiscale urban data accessible on low cost thin client devices offers increasing benefits in many contexts including urban planning and disaster management. Distributed super computing makes this possible by outsourcing pixel rendering to cloud resources. The computational requirements of rendering images to this level are not insignificant, with pixels representing terrestrial topography to the millimeter. Newcastle University's recently deployed Terascope Terapixel project provides an interesting opportunity to analyse performance of such a system in order to further understanding of the underlying computational processes.

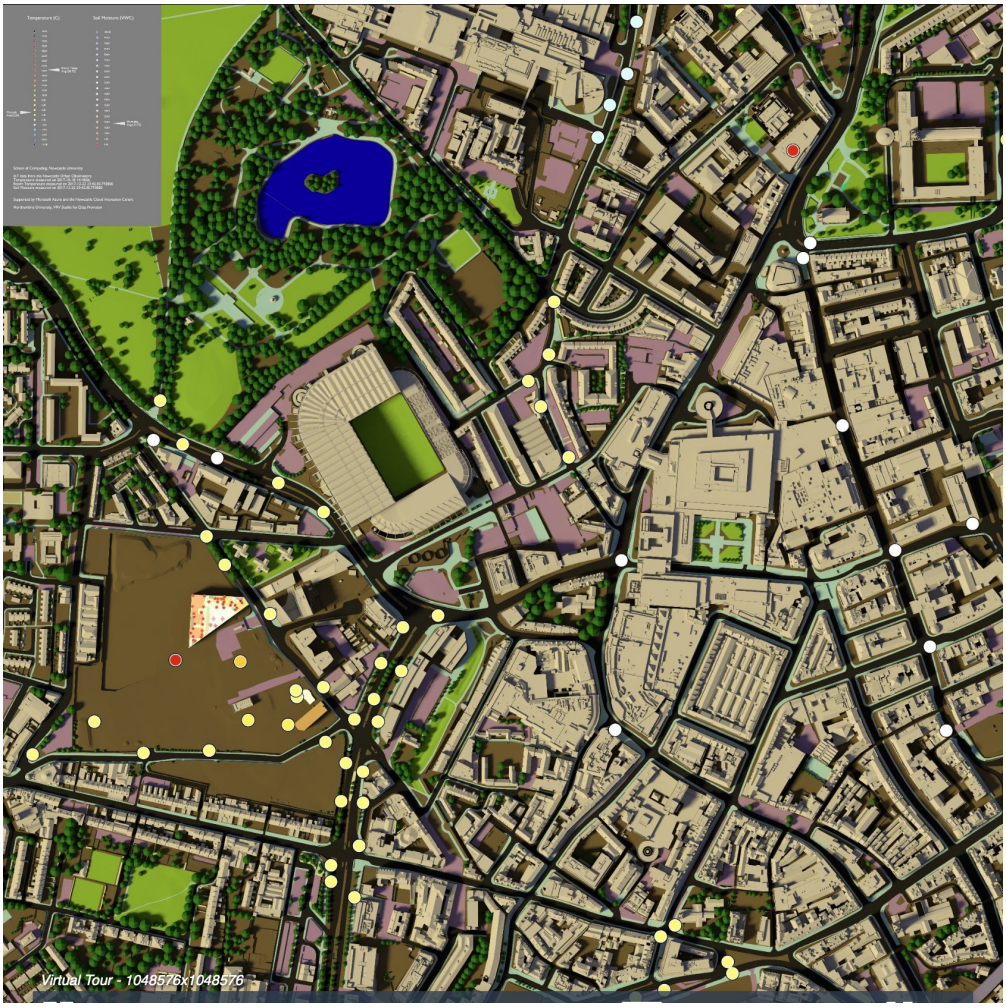

Figure 1: Newcastle University Terascope

2. Background

The Terapixel project offers an intuitive, accessible way to present information sets to stakeholders, allowing viewers to interactively browse big data across multiple scales. The challenge addressed during the project was delivering the supercomputer scale resources needed to compute a realistic terapixel visualization of the city of Newcastle upon Tyne and its environmental data as captured by the Newcastle Urban Observatory[2].

The developed solution was a scalable architecture for cloud-based visualization that can be deployed and payed for as required. The three key objectives of the project were to: 1) create a supercomputer architecture for scalable visualization using the public cloud; 2) produce a terapixel 3D city visualization supporting daily updates; 3) undertake a rigorous evaluation of cloud supercomputing for compute intensive visualization applications.

The project demonstrated that it is feasible to produce a high quality terapixel visualization using a path tracing renderer in under a day using public IaaS cloud GPU nodes. Once generated the terapixel image supports interactive browsing of the city and its data at a range of sensing scales from the whole city to a single desk in a room, accessible across a wide range of thin client devices.

Computational Analysis

The rendering process produced numerous data documenting the various phases of the process. Analysis of these data provides insight into the underlying dynamics of the computational process in order to aid with the design and implementation of future terapixel rendering systems, as well as to provide an analytic basis for designing future benchmarking tests.

Following consultation with the system developers, the analysis focused on the following key areas of inquiry:

- 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 quantify the variation in computation requirements for particular tiles?
- Can we identify particular GPU cards (based on their serial numbers) whose performance differs to other cards? (i.e. perpetually slow cards).
- What can we learn about the efficiency of the task scheduling process?

This inquiry characterises the system through various analyses, but primarily relies on execution time as the key performance metric. As per Hoefler, “execution time is often the best metric for accessing computer performance because it has an undebatable meaning”[1].

3. Methodology

Data analysis was conducted using the R programming language and the R Studio IDE, with the report composed in Rmarkdown. The ProjectTemplate library was used to organise the project and ensure reproducability. Version control was managed using git and Github.

An archived folder is available containing the original Rmarkdown .rmd file and git log. The folder is structured for standard data mining projects, including subfolders for data, cache, munge and source files. As the data analysis involves operations which require extended processing time, key objects have been cached as .Rdata files. The objects are called along the progression of the script (and Rmarkdown report), while the original code is included in the munge folder. Detailed instructions for reproducing the analysis are included in the main README file.

The original csv data files are too large for Github at 300+ MB. Therefore they are not included in the archived project folder. A download link can be found in the README file.

3.1 Data Understanding

The data made available for this analysis consists of 3 tabular data files in comma-separated value (.csv) format covering multiple aspects of the terapixel rendering process. The data files are detailed below.

3.1.1 Application Checkpoints (application-checkpoints.csv 111.2MB)

This file contains 660,400 observations over 6 variables detailing application checkpoint events throughout the execution of the render job.

1. **timestamp** Timestamp of observation entry
2. **hostname** Unique hostname of the virtual machine auto-assigned by the Azure batch system (Total unique: 1024)
3. **eventName** Name of the event occurring within the rendering application. Possible Values:
    - **Tiling** Post processing of the rendered tile
    - **Saving** Measure of configuration overhead
    - **Render** Actual rendering operation
    - **TotalRender** Complete task checkpoints
    - **Uploading** Output from post processing uploaded to Azure Blob Storage
4. **eventType** Possible Values: **START**, **STOP**
5. **jobId** ID of the Azure batch job (Total: 3 one for each level 4, 8 and 12)
6. **taskId** ID of the Azure batch task (Total: 65793 level 4 1x1; level 8 16x16; level 12 256x256)

3.1.2 GPU (gpu.csv 208.7MB)

This file contains 1,543,681 observations over 8 variables detailing logged metrics for each GPU. The log entries occur on a schedule of approximately every 2 seconds for each GPU.

1. **timestamp** Timestamp of observation entry
2. **hostname** Unique hostname of the virtual machine auto-assigned by the Azure batch system (Total unique: 1024)
3. **gpuSerial** Unique serial number of the physical GPU card (Total unique: 1024)
4. **gpuUUID** Unique system id assigned by the Azure system to the GPU unit (Total unique: 1024)
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

3.1.3 Task X Y (task-x-y.csv 6.2MB)

This file contains 65,793 observations over 5 variables detailing the tile x,y co-ordinates of which part the image was being rendered for each task.

1. **jobId** ID of the Azure batch job (Total: 3 one for each level 4, 8 and 12)
2. **taskId** ID of the Azure batch task (Total: 65793 level 4 1x1; level 8 16x16; level 12 256x256)
3. **x** X co-ordinate of the image tile being rendered (Total unique: 256)
4. **y** Y co-ordinate of the image tile being rendered (Total unique: 256)
5. **level** 3 zoomable “google maps style” map levels. 12 levels in total. Level 1 is zoomed right out and level 12 is zoomed right in. Only levels 4, 8 and 12 in the dataset as the intermediate levels are derived during the tiling process.

3.2 Data Preparation

The csv files were read into R and reviewed. Data types were changed as required (eg. timestamps, factors to char etc). In the case of the GPU dataset, the extraneous column **gpuUUID** was dropped, as it was redundant with **gpuSerial** as a unique identifier. The Task X Y dataset was fully merged into the Application Checkpoint dataset, as it contained only 3 unique columns. The resulting object **app_task** was cached along with the **gpu** object for the GPU dataset, after checking for NAs. These cleaned and cached .Rdata objects form the basis of the analysis.

The analysis made heavy use of the Dplyr package to transform and filter **app_task** and **gpu**. Multiple analyses were based on transformed datasets that required extensive processing time. These datasets were cached as .Rdata objects, with the original code included in the munge folder. This function was particularly exemplified in the form of the **gpu_task** dataset, which was the output of the computation of mean resource usage for each event under each of the 65793 tasks in the **app_task** dataset. This computation took three hours on a Corei5 system.

4. Exploratory Data Analysis

As detailed in section 2.2, the data show 1024 unique VMs, each with an attached GPU. These 1024 GPUs were used to render 65,793 pixel generation operations identified by a unique task ID. The four phases of pixel generation in order of execution are: Saving Config, Rendering, Tiling and Uploading. Each of these phases is logged in the **app_task** dataset, once at the start of the phase, and once at the end. In addition, a fifth entry is made for the total rendering process. This results in 10 observations for each pixel generation. Execution times for each phase of each pixel render were derived from the application checkpoint timestamps (**app_task**)

4.1 GPU Single Unit Examination

In order to understand the dataset better, a single GPU was selected at random from the **gpu** dataset (S/N: 323617042631). The logged data for this GPU over the first 10 minutes of processing are displayed in Figure 2. The cycles for each pixel render can be clearly seen in the plot for each metric. Each pixel render appears to take approximately 40 seconds. GPU utilization percentage appears relatively uniform for each pixel render at approximately 90%, whereas there is more variation on the other metrics. This is possibly a result of a percentage usage software limit on the GPU. Unsurprisingly, the temperature builds up from a cold start over successive pixel renders.

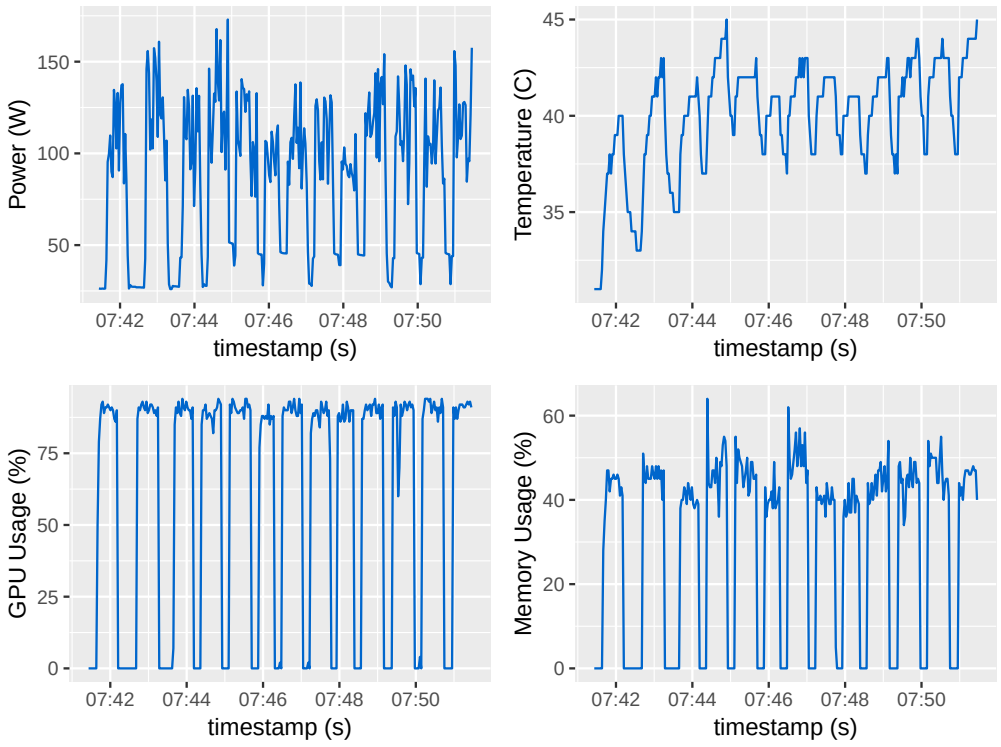

Figure 2: Sample GPU Log Metrics over Time

4.2 GPU Performance and Log Metrics Correlation

Figure 3 displays scatter plots of the GPU log metrics correlations. As the GPU dataset is extremely large at 1.5 million observations, a random sample of 100,000 was selected for the plots. As demonstrated, there is a positive correlation between all of the metrics. However, there is also significant variation. This finding matches the observations taken from Figure 2. The positive correlation is to be expected, as the increased workload on the GPU in the form of GPU utilisation percentage will naturally require associated memory for the tiling job, and the demands of this utilisation on the hardware components will result in increased power consumption and rising temperature.

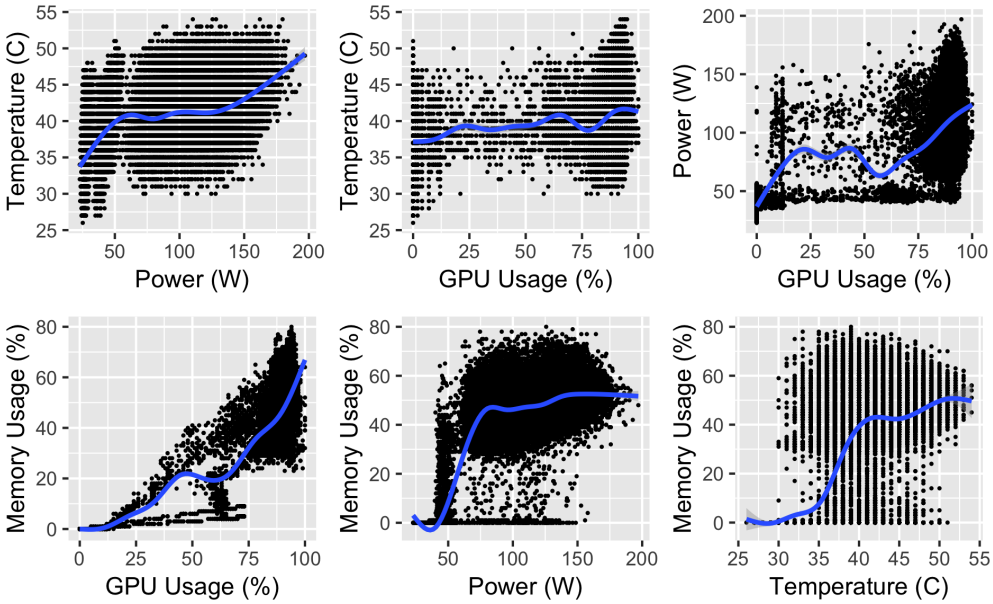

Figure 3: GPU Log Metrics Correlation

4.3 GPU Performance and Log Metrics Analysis

4.3.1 Summary Statistics

Table 1 displays the mean of each GPU log metric by each phase in the pixel rendering process. The rendering task is easily identifiable as the most energy-intense phase of pixel generation, with power consumption more than double that of the second highest phase. There is much less variation in relative temperatures, though rendering is still notably higher in mean temperature than the other phases.

Table 1: Mean GPU Log Metrics by Phase (All Levels)

Event	Power (W)	Temperature (s)	GPU Usage (%)	Memory Usage (%)	n
Render	95.80	40.56	71.467	37.438	65546
Saving Config	36.09	37.96	0.000	0.000	108
Tiling	44.76	39.50	0.000	0.000	31694
Uploading	44.01	39.34	0.000	0.000	35080

There are notable missing data in Table 1 for the 'Saving Config', 'Uploading' and 'Tiling' phases over the GPU and memory utilisation metrics. The dataset used to compute the table was taken from **gpu_task**. This dataset is a join of **gpu** with **app_task**, where each gpu observation is matched by the VM hostname and timestamp for each phase. Despite the reasonably high resolution of the GPU logging operation (each GPU logged performance metrics approximately every 2 seconds), on average these task phases occur over very short intervals (around 1 second or less). As such, these phases were statistically less likely to occur during coincident GPU logging, and consequently have a much smaller sample size relative to the rendering phase.

Table 2 provides quantified insight into this effect by detailing the mean runtime durations for each event. The phases in question display very short average durations. This effect is particularly exemplified in the case of Saving Config. With an average interval of 2ms, it is matched in only 108 of the GPU observations.

The presence of power and temperature data for the under-represented phases in the absence of corresponding

GPU and memory data remains questionable. The power metric can assumed to be exhibiting baseline consumption. It is likely the temperature metric follows a heat loading dynamic, taking time to cool off after each rendering phase. The negative corollary of this effect was already observed (ref. Figure 1). With regard to GPU and memory, the phases in question can be assumed to put only relatively low strain on the hardware resources. Therefore it is not surprising to see zero utilisation on these metrics, as any small percent utilisation caught by the sampling process would be effectively zeroed out by the mean operation.

Table 2: Mean Execution Time by Phase (Level 12 only)

Event	Duration (s)	n
Render	41.206	65536
Saving Config	0.002	65536
Tiling	0.973	65536
Uploading	1.373	65536

4.3.2 Distribution

The summary statistics for the GPU performance and log metrics provide a succinct method for comparing the various phases in terms of impact on system resources. However, they provide only limited insight on actual system performance. A more robust method of benchmarking high performance computational systems is to analyse quantiles including the median[1].

Figure 4 displays the distributions of execution time for each pixel generation. The top plot presents the execution time for rendering and the bottom for the tiling phase. Both distributions appear bimodal, with the larger mode appearing mostly normal with a skew to the right. The smaller mode appears to represents a group of GPUs which exhibit significantly higher performance. However, the computational requirements of each pixel rendering are not uniform. Therefore it is impossible to confirm this from these visualisations (cf. Section 4.4)

The features of the render distribution highlight the importance of documenting system performance beyond summary statistics. The long tail to the right is typical of computational systems, where the accumulated effects of networking, scheduling and processing error push the upper bound towards a log-normal distribution[1]. Therefore the mean execution time is a misleading metric of evaluation. Were a normal distribution assumed in this case, the standard deviation for the render execution time distribution would be calculated as 6.047s. The maximum execution time at 81.51s would then be a highly improbable 6.7 sigma event.

In evaluating the log-normal execution time distribution of this system, a more meaningful measure are quantiles. For the render phase, 50 percent of operations exceeded 41.65s, with 5 percent exceeding 49.85s. For the tiling phase, 50 percent of operations exceeded 0.99s, with 5 percent exceeding 1.09s.

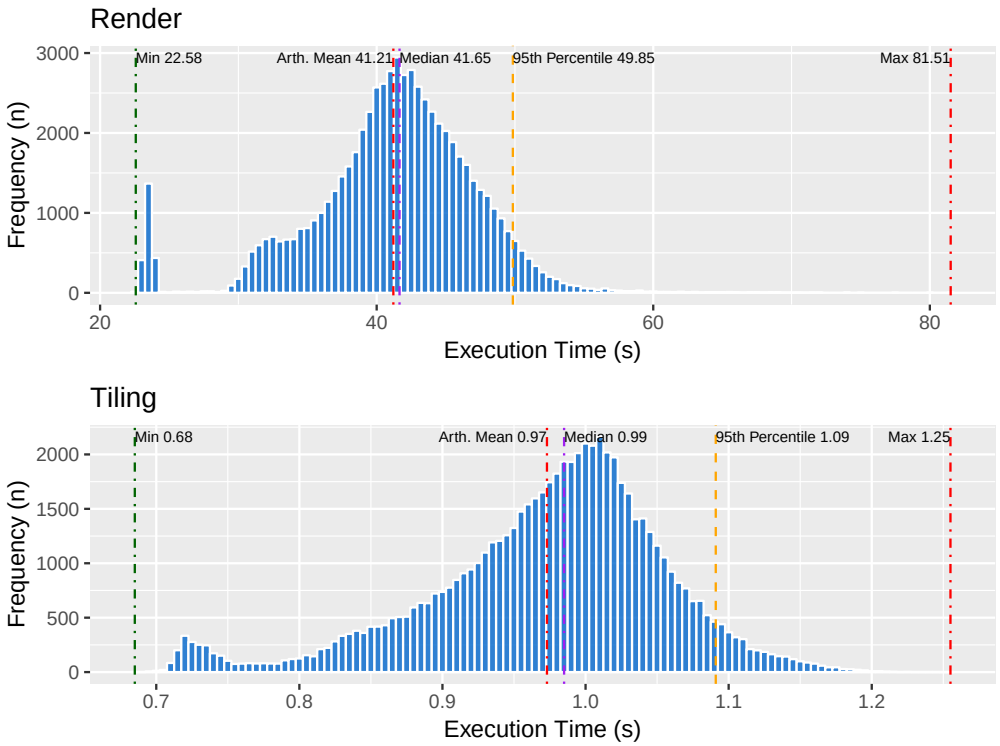

Figure 4: Distribution by Execution Time: Render & Tiling (Level 12)

The bimodal characteristics of the distributions in Figure 4 are more pronounced for the rendering phase distribution; a finding which is logical given the doubled sample size for the rendering phase, as well as the vastly larger quantity of GPU logging data available over the 40+ second phase duration. This much greater sample size for the rendering phase translates to greater significance in the use of this phase for system evaluation. Additionally, the time scale for the rendering phase is relatively large compared to the other phases, making it inherently easier to comprehend the effects of variance on the operation through visualisation. As the comparative performance and relationship between the render phase and the other three phases has already been established in previous sections, and the other three phases offer minimal value toward performance analysis, the *render phase will be used to evaluate system performance exlcusively* from this point.

Figures 5 and 6 display the distributions of power consumption, temperature, GPU and memory utilisation for the render phase. The distribution pairs in each figure appear to be similar. This finding matches the observations in Figure 2, where a high degree of correlation was shown between these respective pairs.

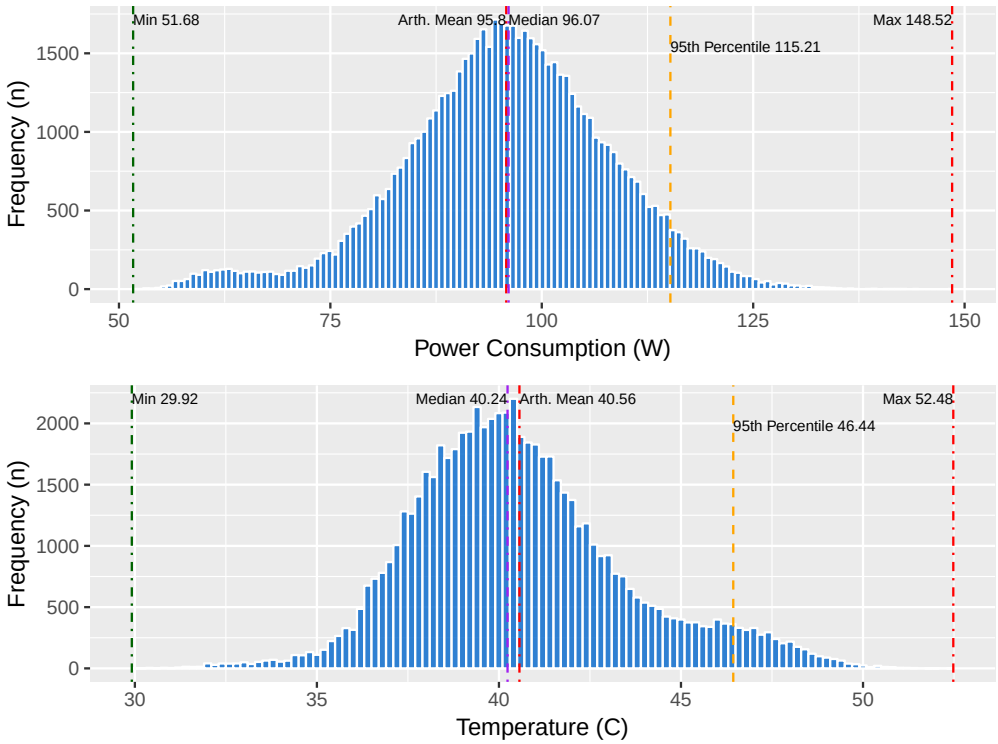

Figure 5: Distribution by Power Consumption & Temperature: Render

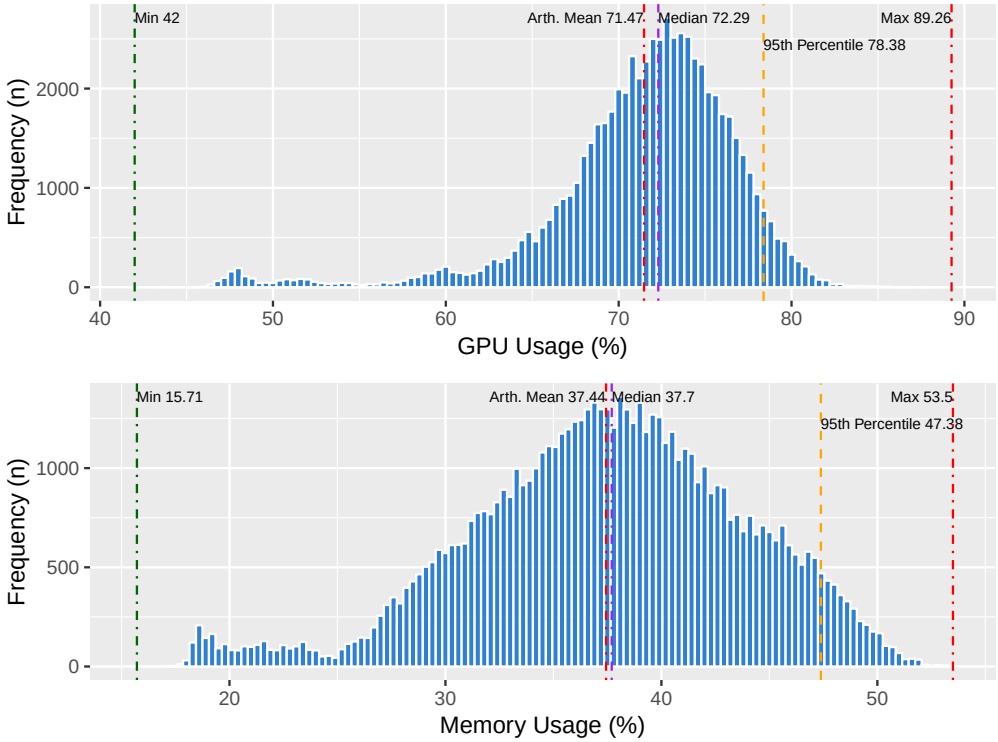

Figure 6: Distribution by GPU & Memory % Utilisation: Render

In contrast to the execution time analysis, the distributions for the GPU logged metrics appear to be more normally distributed. As these data are different are not computational performance measures, it is reasonable to assume they would more closely follow a normal distribution rather than log-normal. The QQ plots in Figure 6 confirm this finding. With the exception of GPU utilisation, the theoretical and sample quantiles for each distribution are closely aligned. Therefore it is safe to model system performance on these metrics based on a normal distribution.

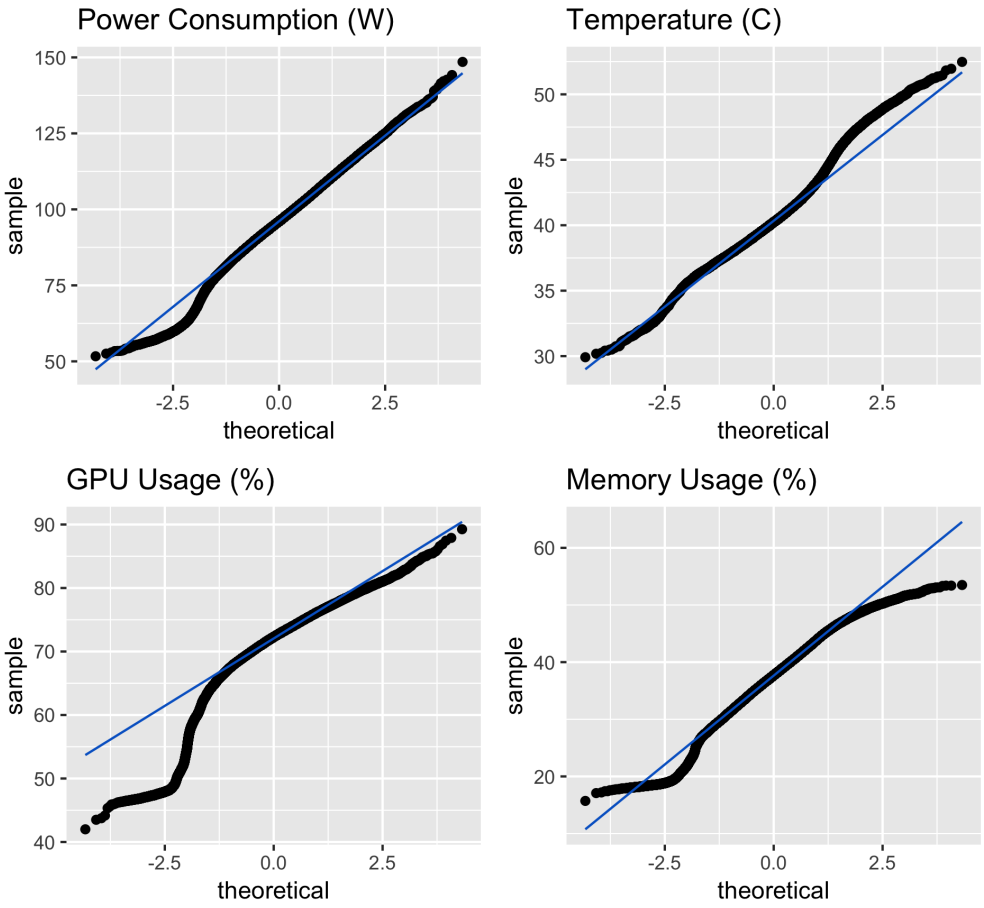

Figure 7: QQ Plot for GPU Log Metrics

4.4 Comparison of Pixel Rendering Complexity

Comparing the variation in GPU execution time and log metrics is problematic in that the workload assigned to each GPU is not identical. Each pixel is unique, and therefore the computational cost of rendering each pixel is not uniform. Various complexities in the input including topological structure affect the processing time and resources required to render the pixel.

A visualisation of the comparative difference in rendering each pixel is presented in Figure 8. The six images are heatmaps displaying the relative impact of rendering each pixel in terms of a particular GPU metric, with lighter colours indicating higher values. The colour mapping is only relative inside each visualisation, i.e. there is no inherent relationship between the colour intensities of the various heatmaps.

The top row of Figure 8 contains visualisations for the GPU log metrics. When compared with the terascope map (ref. Figure 1), the cartographic structure of each heatmap is clearly apparent. As can be seen, there are obvious areas common to all three plots in which the associated pixel rendering was less impactful on

the metric. For instance, the top left corner of each heatmap contains a rectangular region with darker colour. This region corresponds to the legend (ref. Figure 1) which was actually included in the rendering process. As the legend is a mostly uniform colour graphic, the rendering process for pixels in this area was significantly less intense.

Additionally the construction site to the south west of St. James Park is clearly articulated as a low-intensity region. This was possibly due to a lack of activity at the time the terascope input data was acquired. The area has since become a major construction site with many new buildings. In general, streets required higher processing inputs, making them stand out and articulating the buildings and structures they delineate. This is likely caused by the shadows cast on streets by surrounding buildings. Shadows cause gradient lighting effects, resulting in more information to process than for regions of comparatively uniform light and colour.

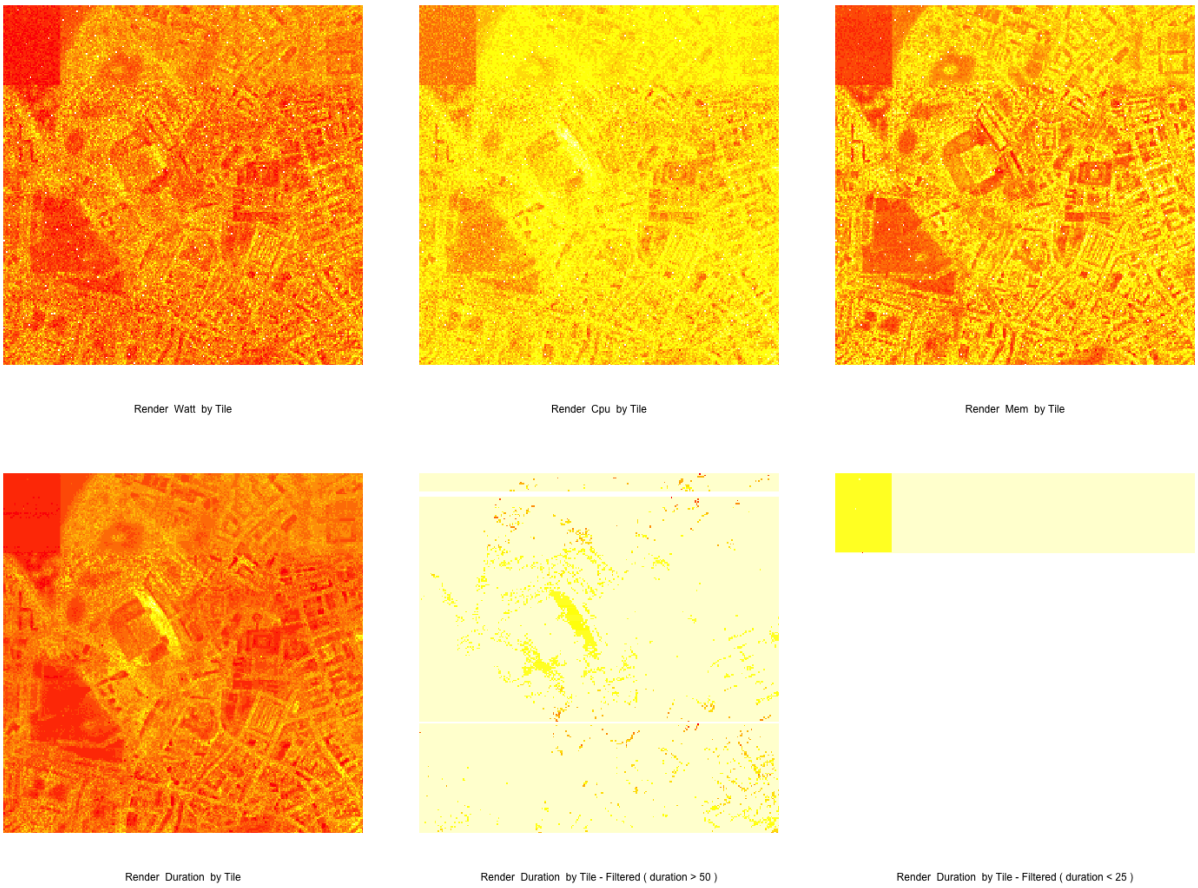

Figure 8: Distribution by GPU & Memory: Render

In section 4.3.2 the bimodal distribution of the GPU execution times (render phase) raised questions as to the cause of the secondary mode. This smaller density is centred at approximately 24s, slightly more than half the median of the primary density (41.65s) and contains approximately 3.4 percent of total samples. The heatmap visualisation can be used to identify the cause of this anomaly.

The bottom row of Figure 8 contains various visualisations for execution time. The far left plot contains all 65,536 observations and displays a highly defined representation of the pixel execution time variation. The middle plot is filtered to display only pixels rendered with an execution time greater than the 95th percentile of the distribution (approximately 50s). In alignment with the left hand plot, it appears the major source of execution time delay was light scatter around the St. James Park superstructure. This finding is unsurprising in that the superstructure is multi-faceted and made of highly reflective material.

The far right plot in Figure 8 is filtered to display only pixels with execution times less than 25s. This filter isolates the smaller of the densities in the bimodal distribution of execution times. As demonstrated, the smaller density is a result of the rendering of the terascope legend; already identified as computationally simple task.

4.5 Effect of GPU Enviornmental and Resource Factors on Execution Time

The variation in rendering by pixel can be elaborated by examining the relationship between the GPU log metrics and execution time. The mean of all log metrics are joined with the execution time for each task in the **task_master** dataset. This provides a generalised insight into how power consumption, temperature, GPU and memory utilisation affect execution time.

The results of this analysis are displayed in Figure 9. As illustrated, there are pronounced positive correlations between each metric and execution time. These correlations are primarily linear. Power consumption is a function of system resource utilisation, and can be expected to roughly follow the GPU and memory curves as execution time slows with increase resource usage. Temperature is also a function of system resource utilisation, given stable environmental conditions and thus demonstrates similar characteristics to power consumption. Memory usage shows a stable upwards growth in execution time as the demands on system memory increase.

The exception to the linearity displayed in Figure 9 is GPU utilisation which presents a concave exponential curve, which appears to exhibit plateau characteristics starting at approximately 80 percent utilisation. This is logical in that the GPU is designed for given operational and theoretical throughput, after which non-linear performance degradation would be expected. However, the most dense region of this plot is largely linear, falling between 60 through 80 percent utilisation.

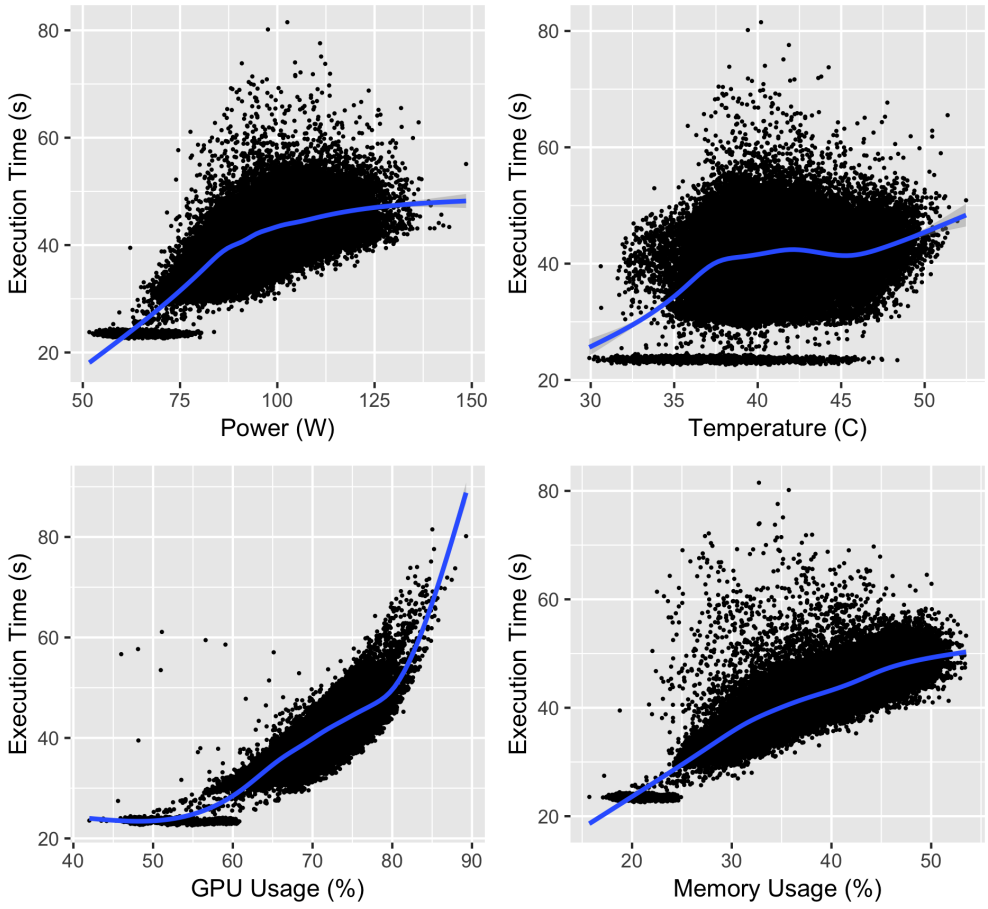

Figure 9: Execution Time by GPU Enviromental and Resource Metrics

4.6 Performance Comparison of Individual GPUs

Previous analyses demonstrated the variability in execution time and GPU log metrics over the 65,536 level 12 pixel rendering tasks. As detailed, the computational difficulty in rendering each pixel is variable. Therefore, it is not possible to benchmark and compare performance directly between GPUs using the Terascope rendering data. It is possible however, to observe broad trends between the GPUs by serial number (S/N).

Referring to Figure 10 and Table 3, four distinct series of S/N can be observed from left to right across the plot. Interestingly, the three series on the right side of the plot (Series 2, 3 and 4) appear to form pronounced clusters above and below the median execution time, indicating a strong bimodal distribution of tasks. In a random task allocation process, it would be expected to observe a more single-moded distribution. This is perhaps a result of distributed computing by Azure between geographically diverse datacentres. For instance, in the UK alone Microsoft has 2 separate Azure POPs.

Series 1 does not exhibit the clustering to the same degree as Series 2 through 4. However, referring to Table 3, this series is clearly the least sampled and therefore is not necessarily indicative of the underlying task allocation distribution.

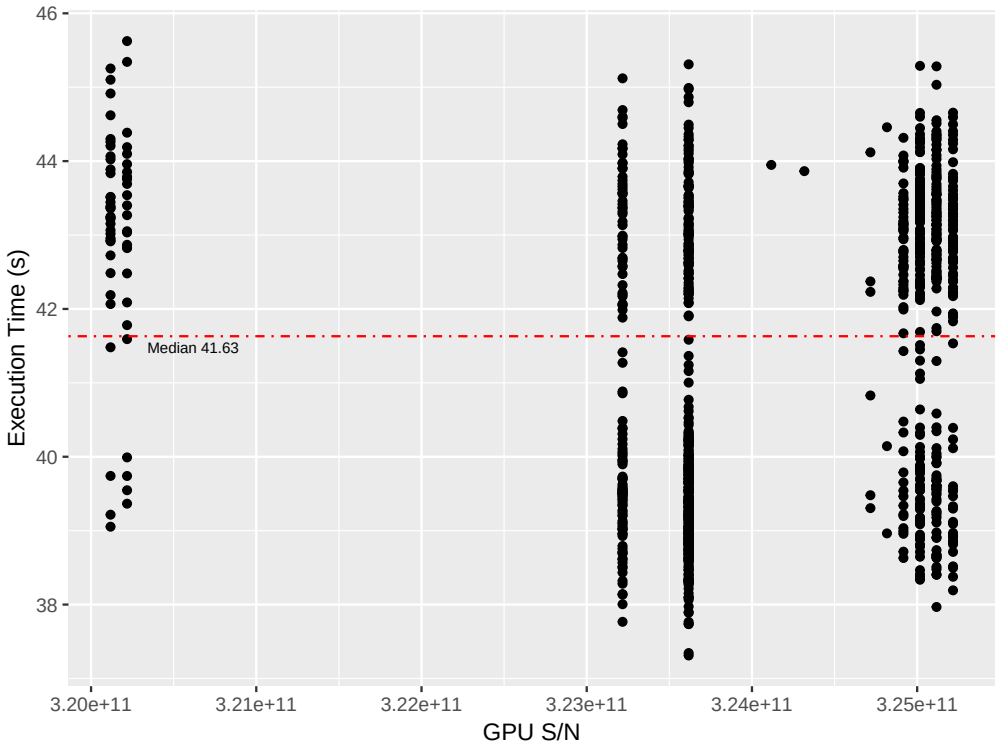

Figure 10: Mean Render Execution Time by GPU S/N

Table 3: GPU S/N Series

	Series 1	Series 2	Series 3	Series 4
Minimum S/N	320118118607	323217048782	323617020079	324117043793
Maximum S/N	320218056015	323217056712	323617043231	325217086519
n	61	121	403	439

5. Results Evaluation and Reccomendations

The analysis returned numerous findings relevant to system performance and future system benchmarking. These results are summarised and evaluated below. Summary conclusions and recommendations are included where applicable.

GPU Logging Frequency
Section 4.3.1 detailed missing data in Table 1 for the 'Saving Config', 'Uploading' and 'Tiling' phases over the GPU and memory utilisation metrics. The cause of this was identified as the 2s GPU logging frequency being too high to capture data during the brief runtime intervals associated with these phases. This was problematic in that it prevented any useful analysis of the system performance during the respective phases. Increased logging frequency during the 'Saving Config', 'Uploading' and 'Tiling' phases would ameliorate the issue, providing greater ability to evaluate the system performance.

GPU Logging Granuarity
Section 4.3.1 also questioned the possibility of logging granularity limits. The 'Saving Config', 'Uploading' and 'Tiling' phases power had matching temperature data for the in the GPU logs, but these entries had no corresponding GPU and memory data. While it is likely any small percent utilisation by these minimally impactful phases caught by the sampling process is being effectively zeroed out by the mean operation, it is

also possible that the log does not register utilization below a certain threshold. If this is the case, increasing the threshold would allow more insight into the processing of these phases.

Differing Distributions
Section 4.3.2 revealed the differing distributions between execution time, and the GPU log metrics. In contrast to the log-normal distribution of the execution time analysis, the distributions for the GPU logged metrics are more normally distributed with the exception of GPU utilization (ref. Figure 6 QQ Plots). With the exception of GPU. Therefore it is safe to model system performance on these metrics based on a normal distribution. Execution time should be modeled and characterised using non-parametric measures such as quantiles[1].

Performance Comparison Difficulties
Section 4.4 characterised the variation in rendering by pixel across execution time and GPU log metrics. Comparing the variation in GPU execution time and log metrics is problematic in that the workload assigned to each GPU is not identical. Each pixel is unique, and therefore the computational cost of rendering each pixel is not uniform. Various complexities in the input including topological structure affect the processing time and resources required to render the pixel. A comparison of system performance would require a structured test dataset, where the input data was the same across devices under test (DUTs)

Limited Insight into Performance Upper Bounds
Section 4.5 illustrated the correlations between each metric and execution time. These correlations are primarily linear with some plateau characteristics. The resources plots contain very few observations toward the higher end of each metric. For instance, the GPU utilisation is only sparsely represented past 85 percent utilisation and memory utilisation is cut off shortly after 50 percent utilisation. A structured testing environment could provide further insight into the upper bounds performance of the system.

GPU Unit Performance Comparison
Section 4.6 analysed the execution time performance of the GPUs by series. Three of the four series appeared to form pronounced clusters above and below the median execution time, indicating a strong bimodal distribution of tasks. In a random task allocation process, it would be expected to observe a more single-moded distribution. This is perhaps a result of distributed computing by Azure between geographically diverse datacentres. Consultation with Microsoft could possibly illuminate the cause of these findings. However, without a structured test environment to ensure uniform task difficulty, any conclusions drawn would not be robust.

Conclusion

This analysis has demonstrated the intense computational requirements of rendering images to the terapixel level. The deployment of 1024 GPUs across the Azure cloud provided a flexible and economical approach to this computation. Findings from the analysis of this process will inform future deployments and provide a solid analytical basis from which to design a benchmarking methodology.

References

1. Torsten Hoefler and Roberto Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15). ACM, New York, NY, USA, Article 73, 12 pages. DOI: https://doi.org/10.1145/2807591.2807644

2. Nick Holliman. 2018. Automating Data Visualization. Presentation. Newcastle University, Newcastel upon Tyne, UK