REMOTE HIGH PERFORMANCE VISUALIZATION OF BIG DATA FOR IMMERSIVE SCIENCE

Faiz Abidi

Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24060
fabidi89@vt.edu

Nicholas Polys

Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24060
npolys@vt.edu

Srijith Rajamohan

Advanced Research Computing
Virginia Tech
Blacksburg, Virginia 24060
srijithr@vt.edu

Lance Arsenault

Advanced Research Computing
Virginia Tech
Blacksburg, Virginia 24060
lanceman@vt.edu

Ayat Mohammed

Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24060
maaayat@vt.edu

1 ABSTRACT

Remote visualization has emerged as a necessary tool in the analysis of big data. High-performance computing clusters can provide several benefits in scaling to larger data sizes, from parallel file systems to larger RAM profiles to parallel computation among many CPUs and GPUs. For scalable data visualization, remote visualization tools and infrastructure is critical where only pixels and interaction events are sent over the network instead of the data. In this paper, we present our pipeline using VirtualGL, TurboVNC, and ParaView to render over 40 million points using remote HPC clusters and project over 26 million pixels in a CAVE-style system. We benchmark the system by varying the video stream compression parameters supported by TurboVNC and establish some best practices for typical usage scenarios. This work will help research scientists and academicians in scaling their big data visualizations for remote, real-time interaction.

Keywords: Remote rendering, CAVE, HPC, ParaView, big data.

SpringSim-HPC, 2018 April 15-18, Baltimore, MD, USA;

2 INTRODUCTION

Network is one of the biggest bottlenecks of remote rendering and we want to minimize the data sent and received from the server. The most commonly used method to do this is to compress the data. However, compression has its own disadvantages. Compression and decompression consumes CPU time meaning that some extra time will be added to the rendering process. Another disadvantage is that compression can lead to loss of data depending on what type of algorithms are being used (lossy versus lossless algorithms). Apart from compression, there are other factors that can help minimize network bandwidth usage at the cost of reduced render quality like JPEG quality, JPEG chrominance sub-sampling, and the amount of compression level applied. The goal of this work is to identify which of these factors is significant for remote rendering of big data over a dedicated 10 Gbps network.

2.1 Graphics rendering

Graphics rendering pipeline consists of two phases - geometry processing and rasterization. The authors in (Molnar, Cox, Ellsworth, and Fuchs 1994) have described three classes of parallel rendering algorithms; sort-first, sort-middle, and sort- last. A significant advantage of the sort-last algorithm is that the renderers implement the complete rendering pipeline and are independent until pixel merging. In general, all the three rendering algorithms suffer from some common problems like load balancing, high processing and communication costs, and highly pixelated content. In this paper, we used ParaView (Kitware 2017) to run our experiments, which implements sort-last algorithm in its code base. Samanta et al. in (Samanta, Funkhouser, Li, and Singh 2000) proposed a hybrid of sort-first and sort-last parallel polygon rendering algorithm. The hybrid algorithm outperforms sort-first and sort-last algorithms in terms of efficiency. Overall, the hybrid algorithm achieved interactive frame rates with an efficiency of 55% to 70.5% during simulation with 64 PCs. The hybrid algorithm provides a low cost solution to high performance rendering of 3D polygonal models.

Visualizing big data sets has always been challenging because of limited CPU, memory, parallelization, and network requirements and Ahrens et al. talk more about it in (Ahrens, Desai, McCormick, Martin, and Woodring 2007). Ahrens et al. (Ahrens, Brislawn, Martin, Geveci, Law, and Papka 2001) also discuss similar problems with respect to large-scale data visualization and present an architectural approach based on mixed dataset topology parallel data streaming. Moreland et al. in (Moreland, Wylie, and Pavlakos 2001) used sort-last parallel rendering algorithm to render data to large tile displays. Morland et al. in (Moreland and Thompson 2003) describe a new set of parallel rendering components for the Visualization Toolkit (VTK) (Schroeder, Martin, and Lorensen 1996), which is also used by ParaView for data visualization. They introduced components of Chromium (Humphreys, Houston, Ng, Frank, Ahern, Kirchner, and Klosowski 2002) and ICE-T (an implementation of (Moreland, Wylie, and Pavlakos 2001)) into VTK and showed that VTK can be a viable framework for cluster-based interactive applications that require remote display.

Eilemann et al. (Eilemann, Makhinya, and Pajarola 2009) introduced a system called Equalizer, a toolkit for parallel rendering based on OpenGL. Equalizer takes care of distributed execution, synchronization, and final image composting, while the application programmer identifies and encapsulates culling and rendering. This approach is minimally invasive since the proprietary rendering code is retained.

2.2 Remote Scientific Visualization

ParaView (Ahrens, Geveci, and Law 2005)(Kitware 2017) is an open-source toolkit that allows scientists to visualize large datasets. It is based on the visualization toolkit called VTK (Schroeder, Martin, and Lorensen 1996) that provides the data representations for a variety of grid types including structured, unstructured,

polygonal, and image data. ParaView extended VTK to support streaming of all data types and parallel execution on shared and distributed-memory machines.

TurboVNC is a derivative of VNC (Virtual Network Computing) (Richardson, Stafford-Fraser, Wood, and Hopper 1998) that has been specifically developed to provide high performance for 3D and video workloads (TurboVNC 2017). It is an open source project that contains a modern X server code base (X.org 7.7).

VirtualGL (VirtualGL 2017) is an open source software that enables remote display software the ability to run OpenGL applications with full 3D hardware acceleration. It redirects the 3D rendering commands and the data of the OpenGL applications to a graphical processing unit (GPU) installed on the remote server and sends the rendered 3D images to the client.

A CAVE automated virtual environment (Cruz-Neira, Sandin, DeFanti, Kenyon, and Hart 1992) is a virtual reality interface that consists of walls and each wall can be driven by one or more projectors. The first CAVE was developed at the University of Illinois, Chicago Electronic Visualization Laboratory in 1992 for scientific and engineering applications and overcome the limitations of head mounted displays (HMD).

3 DESIGN AND IMPLEMENTATION

3.1 Experimental setup

ParaView consists of a data server, render server, and a client. The data and the render server can be separate machines but if they are on the same host, they are referred to as a pvserver. In our experiments, we had the render and the data server running on the same machine acting as the pvserver. We used NewRiver (Computing 2017) for remote rendering and the Hypercube machine in our lab for running the ParaView client. A vncserver was started on Hypercube and a user could use that vncsession to connect to Hypercube to run the experiments. Ganglia daemon (Sourceforge 2016) running on Hypercube was set at a polling frequency of 3 seconds and used to collect metrics like memory and network usage. By default, Ganglia sets the polling frequency at 15 seconds but we wanted to poll more frequently to collect more accurate data. However, polling every second is also not recommended since that could stress the ganglia server and therefore we decided to poll every 3 seconds instead. Our experimental pipeline consisted of Paraview, TurboVNC, and VirtualGL. Figure 1 shows the architectural overview and Figure 2 shows a flow diagram of the different components talking to each other. Figure 3 shows a 40 million point cloud dataset being analyzed in the CAVE at Virginia Tech.

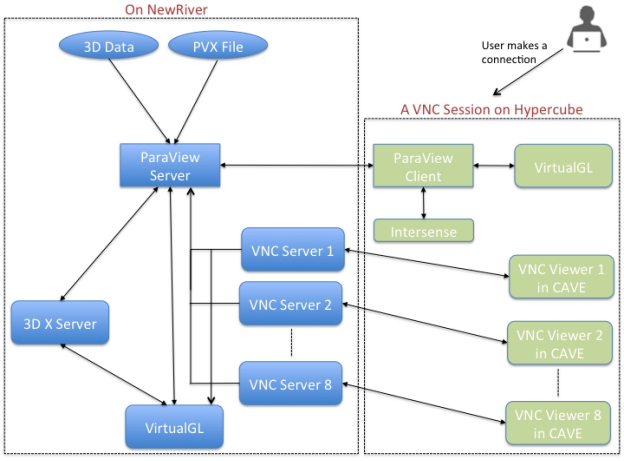

Figure 1: Overview of the experimental setup

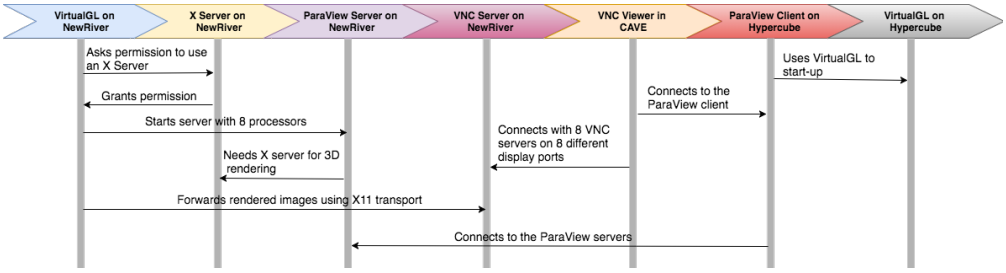

Figure 2: Flowdigram of the experimental setup

Figure 3: 40 million point cloud data set visualized using our experimental pipeline and remote HPC servers

3.2 Hardware specifications and data used

We used the NewRiver HPC cluster at Virginia Tech for remote rendering. Each host had 24 CPUs, 512 GB of memory, and 2 Tesla K80 GPUs. The Hypercube machine used to run the ParaView client had 48 CPUs, 512 GB of memory, and 4 Nvidia Quadro M6000 GPUs. Each projector used in the lab to run the CAVE runs at a frequency of 120 HZ with active stereo and a resolution of 2560x1600. We used a point cloud dataset that had more than 5 billion points in it. It is a LiDAR (Reutebuch, Andersen, and McGaughey 2005) mass scan of the Goodwin hall at Virginia Tech that has motion sensors installed at different locations. The total size of this dataset was 270 GB and it was available in the form of CSV files. This data was cleaned of the noise and converted into unstructured XML VTK format in binary also called VTU (FEM 2016) format using ParaView's D3 filter (Kitware 2017). Converting the CSV files into VTU files was necessary to make sure that the data can be distributed in parallel when running ParaView with MPI.

3.3 Data Interaction and ParaView's timer log

ParaView provides a timer log that monitors the time taken to render the data during different stages. We were interested in observing the still render time and the interactive render time. Still render measures the time from when the render is initiated to the time when the render completes. Interactive render occurs when a user moves and interacts with the 3D data using the ParaView GUI. We needed the interactions to remain consistent for all the experiments run and for this reason we decided to script the mouse movements on the client machine using a Python library called pynout (Palmér 2014). We added 1000 random scripted mouse movements on the client side interacting with the data shown on the ParaView GUI. We used the values of still and interactive renders obtained from the timer log and averaged them to obtain the Framerate Per Second (FPS) reported here.

3.4 Other compression options

ParaView provides several compression parameters that can be set with remote rendering. However, it does not allow to script the parameters and every time a user wants to make a change, they have to do it using the GUI. For this reason, we decided to not use ParaView's compression settings. VirtualGL also lets a user set compression settings but it does not perform any compression when using the X11 transport method in which case it relies on the X proxy to encode and deliver the images to the client. We did not realize this initially when we started running our experiments and were trying to vary the compression parameters supported by VirtualGL. As expected, we did not observe any difference in the results that we got and this led us to explore the compression supported by TurboVNC instead. The compression settings provided by TurboVNC were relevant for our case and we could script them for our experiments.

4 RESULTS

In this section, we discuss the different results obtained by varying the compression parameters supported by TurboVNC. We measured the maximum, minimum, and average network and memory consumed at the client side, the maximum memory consumed at the server side, and the average still and interactive frame rates. Note that we did not measure percentage CPU on the server side because when we run pvservers with MPI, we get 100% CPU utilization even when the pvserver sits idle. This is just how the MPI layer is implemented in the two most common implementations: OpenMPI (MPI 2017) and MPICH (MPICH 2017).

We created nine groups based on the different compression parameters as shown in Table 1 and all the results obtained by varying the number of VTU files and CPUs are shown in Figure 4.

Table 1: Different compression parameters used in the experiments

#	Encoding	JPEG	Quality	JPEG Subsampling	Compression Level
CASE 1	Tight	0	95	1x	0
CASE 2	Tight	0	95	1x	1
CASE 3	Tight	0	95	1x	5
CASE 4	Tight	0	95	1x	6
CASE 5	Tight	1	95	1x	0
CASE 6	Tight	1	95	1x	1
CASE 7	Tight	1	95	1x	5
CASE 8	Tight	1	80	2x	6
CASE 9	Tight	1	30	4x	7

4.1 Frame rates with 8 VTU files and 8 CPUs

For still render FPS, the average obtained as shown in Figure 4 was 6.536 with a standard deviation of 0.118 and a standard error of 0.039. CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) with 6.727 FPS performed the best while CASE 4 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:6) with 6.305 FPS performed the worst. For interactive render FPS, the average obtained as shown in Figure 4 was 52.073 with a standard deviation of 4.393 and a standard error of 1.464. CASE 8 (JPEG:1, Quality:80, JPEG Subsampling:2x, Compression Level:6) with 55.907 FPS preformed the best while CASE 4 with 40.172 FPS performed the worst. Overall, there was not much difference between the different still and interactive render FPS obtained for the different cases except for CASE 4.

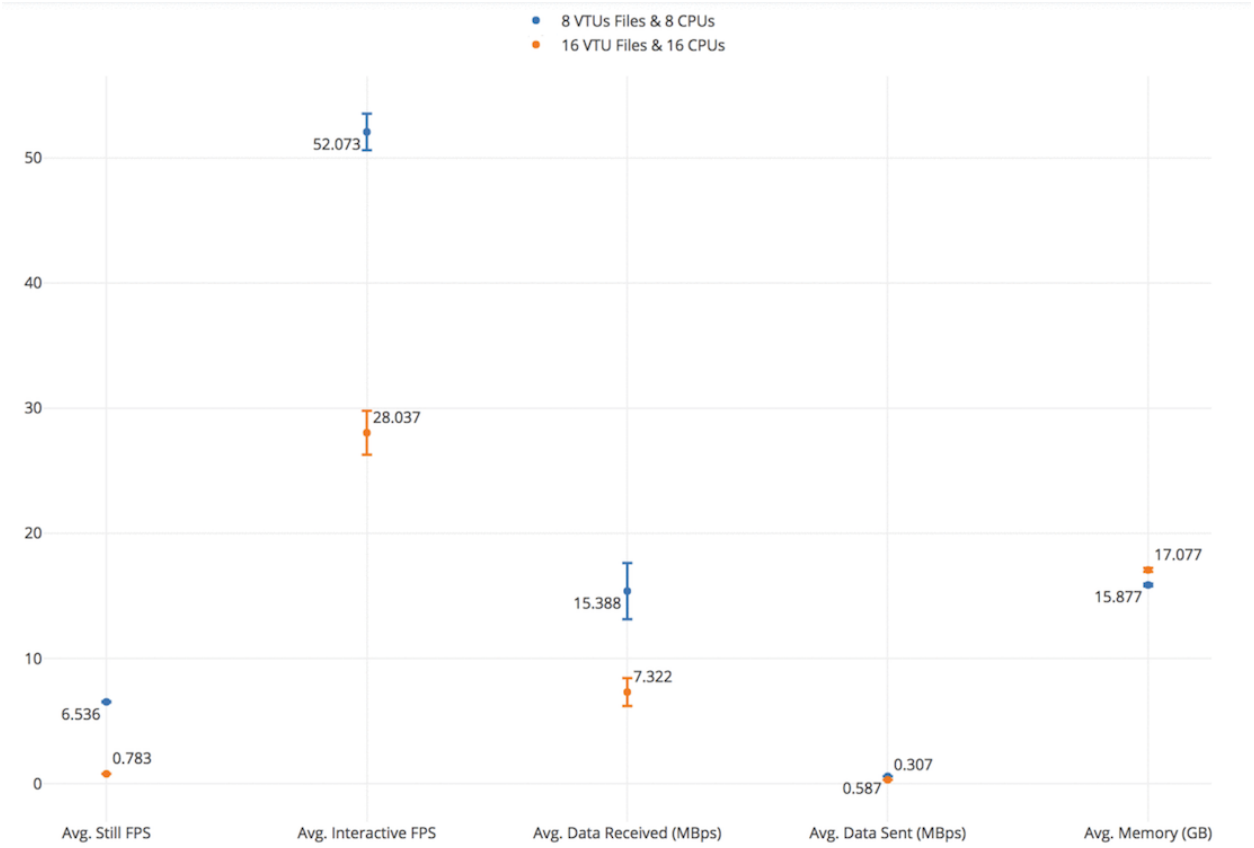

Figure 4: Experimental results obtained using different combinations

4.2 Frame rates with 16 VTU files and 16 CPUs

For still render FPS, the average obtained as shown in Figure 4 was 0.783 with a standard deviation of 0.004 and a standard error of 0.001. CASE 7 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:5) with 0.792 FPS performed a little better than the others while CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) and CASE 2 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:1) performed the worst with 0.778 FPS each. For interactive render FPS, the average obtained as shown in Figure 4 was 28.037 with a standard deviation of 5.255 and a standard error of 1.751. CASE 9 (JPEG:1, Quality:30, JPEG Subsampling:4x, Compression Level:7) with 33.946 FPS preformed the best while CASE 2 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:1) with 18.699 FPS performed the worst. Overall, there was not much difference between the different still and interactive render FPS obtained for the different cases except for CASE 2 and CASE 4 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:6).

4.3 Network usage with 8 VTU files and 8 CPUs

Figure 4 shows the average data received by the client at 15.388 MBps with a standard deviation of 6.723 and a standard error of 2.241 while the average data sent by the client was at 0.587 MBps with a standard deviation of 0.017 and a standard error of 0.005. The maximum data received by the client at any given point in time during the rendering process was almost the same for all cases except CASE 3 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:5) at 37 MB/sec and CASE 4 (JPEG:0, Quality:95, JPEG Sub-

sampling:1x, Compression Level:6) at 40.7 MB/sec. On an average, CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) and CASE 3 received the maximum amount of data at 24.4 MB/sec each while the least average data was received by CASE 9 (JPEG:1, Quality:30, JPEG Subsampling:4x, Compression Level:7) at 6 MB/sec. The average data sent from the client to the server was almost the same for all the cases but it was the least for CASE 7 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:5) at 0.554 MB/sec.

4.4 Network usage with 16 VTU files and 16 CPUs

Figure 4 shows the average data received by the client at 7.322 MBps with a standard deviation of 3.333 and a standard error of 1.111 while the average data sent by the client was at 0.307 MBps with a standard deviation of 0.023 and a standard error of 0.007. The maximum data received by the client at any given point in time during the rendering process was almost the same for all the cases (52.8 MB/sec) except for CASE 7 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:5) at 41.8 MB/sec. On an average, CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) at 12.4 MB/sec received the maximum amount of data while the least average data was received by CASE 9 (JPEG:1, Quality:30, JPEG Subsampling:4x, Compression Level:7) at 2.8 MB/sec. The average data sent from the client to the server was least for CASE 6 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:1) at 0.246 MB/sec and maximum for CASE 5 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:0) at 0.329 MB/sec.

4.5 Memory usage with 8 VTU files and 8 CPUs

Figure 4 shows the average memory consumed at the client side as 15.877 GB with a standard deviation of 0.332 and a standard error of 0.110. The peak memory was consumed by CASE 9 (JPEG:1, Quality:30, JPEG Subsampling:4x, Compression Level:7) at 23 GB while CASE 5 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:0) consumed the least memory at 17.7 GB. On an average, CASE 2 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:1) and CASE 7 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:5) consumed the maximum memory at 16.3 GB each while CASE 3 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:5) consumed the least memory at 15.4 GB. On the server side, there was not much difference in the peak memory usage for any test case with CASE 4 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:6) at 50.85 GB consuming the maximum while CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) at 50.65 GB consuming the least.

4.6 Memory usage with 16 VTU files and 16 CPUs

Figure 4 shows the average memory consumed at the client side as 17.077 GB with a standard deviation of 0.456 and a standard error of 0.152. The peak memory was consumed by CASE 8 (JPEG:1, Quality:80, JPEG Subsampling:2x, Compression Level:6) at 22.4 GB while CASE 1 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:0) and CASE 2 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:1) at 17.7 GB each consumed the least memory. On an average, CASE 3 (JPEG:0, Quality:95, JPEG Subsampling:1x, Compression Level:5) at 18.1 GB consumed the maximum memory while CASE 6 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:1) at 16.3 GB consumed the least memory. On the server side, there was not much difference in the peak memory usage for any test case with CASE 2 and CASE 7 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:5) at 108.5 GB each con-

suming the maximum while CASE 5 (JPEG:1, Quality:95, JPEG Subsampling:1x, Compression Level:0) at 108.39 GB consuming the least.

5 DISCUSSION

5.1 Welch Two Sample t-tests

We had several combination of variables that we varied and got different results. In order to understand which of these results are significant and dependent on the variables, we ran 8 Welch Two Sample t-tests. We created two groups as shown in Table 2. The t-tests were run for 8 variables - a) average data received; b) maximum data received; c) average data sent; d) average memory on client; e) maximum memory on client; f) maximum memory on the server; g) average still render frame rates; and h) average interactive render frame rates.

Table 2: Groups created for Welch Two Sample t-tests

Group	# Processors	# VTU files
I	8	8
II	16	16

Table 3: Significant results obtained from the t-tests

Variable name	p-value
Avg. data received	0.005267
Avg. data sent	1.96E-14
Avg. still render frame rate	4.14E-15
Avg. interactive render frame rate	2.03E-08

Only 4 out of the 8 p-values that we got were significant as shown in Table 3. This means that the other 4 variables (max. data received, avg. memory on the client, max. memory on the client, and max. memory on the server) were not affected by changing the number of processors and VTU files.

5.2 ANOVA Analysis

To further analyze the effects of the different compression parameters on the variables, we created sub-groups as shown in Table 4 and ran 8 ANOVAs for each of the 8 variables. The results we got are shown in Table 6. All the results except for maximum memory on the client and maximum data received were affected by the number of processors and the VTU files. We can see from the results that the number of processors and the VTU files were the main factors affecting the variables and there was no variable that was affected if JPEG was enabled or disabled. Within each of the two main groups (8 and 16 processors and VTU files), we created sub-groups and used 3 predictors: JPEG, Quality, and Chrominance (see Table 5). ANOVA did not show significant results for any variable except for one as shown in Table 7.

Table 4: Groups created for ANOVA for the entire data

Group	# Processors and VTU files	JPEG
I	8	Disabled (0)
II	8	Enabled (1)
III	16	Disabled (0)
IV	16	Enabled (1)

Table 5: Sub-groups created for ANOVA

Group	JPEG Compression	Quality	Chrominance
I	Disabled (0)	95	1
II	Enabled (1)	95	1
III	Enabled (1)	80	2
IV	Enabled (1)	30	4

Table 6: Significant results obtained from ANOVA for all the data

Variable name	Significant factor	p-value
Avg. data received	Processors and VTU files	0.01209
Avg. data sent	Processors and VTU files	3.25E-13
Avg. memory on client	Processors and VTU files	2.68E-05
Max. memory on server	Processors and VTU files	<2e-16
Avg. still render FPS	Processors and VTU files	<2e-16
Avg. interactive render FPS	Processors and VTU files	5.169e-08

Table 7: Significant results obtained from ANOVA of sub-groups

Variable name	Significant factor	p-value
Avg. still render FPS	JPEG	0.04305 for 16 processors and 16 VTU files

5.3 Significant observations

It can be observed from the results obtained above that the average still render frame rate for 8 CPUs and 8 VTU files was significantly better than those for 16 CPUs and 16 VTU files by a factor greater than 8. This is also shown by the t-tests. The interactive render frame rate for 8 CPUs and 8 VTU files was also significantly better than those for 16 CPUs and 16 VTU files by a factor greater than 1.6. Based on the ANOVA analysis, the most significant factor deciding the frame rates was the number of CPUs and VTU files and if JPEG compression was enabled. Quality and JPEG chrominance sub-sampling did not have any significant impact on the dependent variables.

The average data received by the client for 8 CPUs and 8 VTU files was significantly more than those for 16 CPUs and 16 VTU files by a factor almost equal to 2. The peak data received by the client for 8 CPUs and 8 VTU files was almost the same as for 16 CPUs and 16 VTU files.

The average memory consumed by the client for 8 CPUs and 8 VTU files was slightly less than that consumed for 16 CPUs and 16 VTU files by a factor almost equal to 1.11. The peak memory consumed by the client for 8 CPUs and 8 VTU files was almost equal to that consumed for 16 CPUs and 16 VTU files. The peak memory consumed by the server for 8 CPUs and 8 VTU files was significantly less than that consumed for 16 CPUs and 16 VTU files by a factor almost equal to 2.

5.4 Effect of network and stereo

All the results that we got used the VT-RNET 10 Gbps network, which meant that if we tried to upload or download a single file, we could get speeds almost equal to 256 MB/sec (we would get higher speeds with parallel downloads or uploads). However, had we been using a 1 Gbps connection, we would have got upload and download speeds in the range of 5 MB/sec (we would get higher speeds with parallel downloads or uploads). Generally speaking, the network speeds often have a bottleneck with hard disk throughput, bus contention, frame size, packet size distribution among other reasons. The download and upload speeds that we are reporting in this paper is based on experimental data collected in the lab. In our experiments run using ParaView, the peak data transfer speed achieved was 52.7 MB/sec, which means that if we were using the 1 Gbps network to run our experiments, our results would have been affected significantly.

All our experiments were run in a monoscopic mode using 8 processors or 16 processors. As we observed in our results, the frame rates achieved with 8 processors was almost double the frame rates achieved with 16 processors and this is mainly because of how ParaView duplicates the data on each node (see section 6.1 for details). Therefore, we believe that if we were to do stereoscopic data visualization in the CAVE that would mean defining 2x number of DISPLAYs in the pvx file (one for the left eye and one for the right eye) and requesting 2x number of processors for rendering, and because of how ParaView duplicates data on each node, adding more processors should adversely affect performance.

6 LIMITATIONS

6.1 Big data in CAVE

Our initial goal was to visualize all the five billion points in the CAVE and test the different compression parameters. But we encountered an issue with ParaView and the MPI implementation when we tried to load big data sets. More research into this issue showed that there are two MPI routines that are used for sending and receiving data used in ParaView - MPI_Send and MPI_Recv. The return type for both of these routines is an integer meaning that the return value can not be greater than 2^31 - 1, which is equal to 2 GB. This essentially meant that whenever MPI will try to send or receive data greater than 2 GB, it will crash. ParaView in a CAVE mode duplicates data on each node and there is no compositing. This duplication of data does not happen in a non-CAVE mode and hence, bigger datasets can be visualized. The reason given by the ParaView developers for the duplication of data in the CAVE mode is to increase its efficiency. There are at least two solutions that can potentially be applied in the ParaView source code to fix this issue. If an application wants to send, say, 4 x 2 GB to a peer, instead of sending one big single 8 GB message, the application can send 4 messages each of 2 GB in size. Another workaround is to use “MPI_TYPE_CONTIGUOUS" to create a datatype comprised of multiple elements, and then send multiple of these creating a multiplicative effect (Squyres 2014)

6.2 Stereo rendering in CAVE

With Barco F50 projectors that we used in the lab at Virginia Tech, the active stereo happens at the projector level. It sends two separate video channels, one for the left eye and one for the right eye. ParaView in a CAVE mode does not have an option to define a left eye and a right eye and as of writing this paper, ParaView does not have this functionality in their latest version v5.4. It assumes that there is just one channel coming from the projector that has information for both the eyes. The DISPLAYs set in the ParaView’s pvx file are not meant to be different for different eyes and defines a single eye separation value for the whole configuration. This problem can be solved if we enable the stereo option in the Nvidia configuration file in which case ParaView would have given us a left eye on one display and the right eye on the other and this would have been fed to the projector that would again turn it into active stereo. But we kept the stereo option disabled in the X configuration file since when stereo was enabled it interfered with the Nvidia driver-level blending. This is a known issue with Nvidia drivers for Linux (as of this writing, Nvidia has put it on their to-do list to fix it). The other option to solve this problem as suggested by the ParaView community is to define a left eye and a right eye in the ParaView source code, which would lead to passive stereo (this feature does not exist in the latest ParaView version v5.4).

7 CONCLUSION AND FUTURE WORK

In this work, we remotely rendered a point cloud dataset consisting of more than 40 million points in a CAVE-style system. Our pipeline consisted of ParaView, VirtualGL, and TurboVNC. We varied 6 parame-

ters including the number of processors, number of VTU files, JPEG compression, Quality, JPEG chrominance sub-sampling, and compression level. We measured the data flowing to and from the client to the servers, the memory foot print on the client and the server, and the frame rates received at the client side. We used NewRiver HPC machines at Virginia Tech for remote rendering. Our results show that beyond a point, adding more CPUs and dividing the data into more number of VTU files does not help in speeding the rendering process and visualizing the data in a CAVE-style system. The frame rates achieved with 8 processors and 8 VTU files was significantly better than the frame rates achieved with 16 processors and 16 VTU files.

The work we did in this paper leaves room for some future. It would be interesting to see how other combinations perform like 8 CPUs and 16 VTU files, 16 CPUs and 8 VTU files, etc. Currently, there is a limitation to how much data can be visualized using ParaView in the CAVE but if some more work is done on enhancing ParaView's capability to support bigger data sets by incorporating the changes suggested in section 6.1, it would be interesting to see how ParaView performs and how it affects the network and frame rates. In terms of network speed, we used a 10G network for all our tests, which could easily accommodate the maximum data flowing between the client and the server. This could be one of the reasons why compression parameters like quality and JPEG chrominance sub-sampling did not have any significant impact on the frame rates achieved. It would be interesting to see how the same experiments will behave while using a 1G network. Adding active stereo support to ParaView in addition to the mono mode that it currently supports is also future work. We believe that using stereo will adversely affect frame rates compared to the frame rates achieved in a monoscopic mode

REFERENCES

Ahrens, J., K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka. 2001. "Large-scale data visualization using parallel data streaming". *IEEE Computer graphics and Applications* vol. 21 (4), pp. 34–41.

Ahrens, J and Geveci, B and Law, C 2005. "ParaView: An End-User Tool for Large Data Visualization. Number ISBN-13: 978-0123875822. Visualization Handbook".

Ahrens, J. P., N. Desai, P. S. McCormick, K. Martin, and J. Woodring. 2007. "A modular extensible visualization system architecture for culled prioritized data streaming". In *Visualization and Data Analysis 2007*, Volume 6495, pp. 64950I. International Society for Optics and Photonics.

Advanced Research Computing 2017. "Newriver Cluster for HPC". https://secure.hosting.vt.edu/www.arc.vt.edu/computing/newriver/. [Online; accessed 11-April-2017].

Cruz-Neira, C., D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. 1992. "The CAVE: audio visual experience automatic virtual environment". *Communications of the ACM* vol. 35 (6), pp. 64–72.

Eilemann, S., M. Makhinya, and R. Pajarola. 2009. "Equalizer: A scalable parallel rendering framework". *IEEE transactions on visualization and computer graphics* vol. 15 (3), pp. 436–452.

Elmer FEM 2016. "VTU file format and ParaView for visualization". http://www.elmerfem.org/blog/uncategorized/vtu-file-format-and-paraview-for-visualization. [Online; accessed 17-April-2017].

Humphreys, G., M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. 2002. "Chromium: a stream-processing framework for interactive rendering on clusters". *ACM transactions on graphics (TOG)* vol. 21 (3), pp. 693–702.

Kitware 2017. "ParaView - Parallel Visualization Application". http://www.paraview.org/. [Online; accessed 12-April-2017].

Molnar, S., M. Cox, D. Ellsworth, and H. Fuchs. 1994. "A sorting classification of parallel rendering". *IEEE computer graphics and applications* vol. 14 (4), pp. 23–32.

Moreland, K., and D. Thompson. 2003. "From cluster to wall with VTK". In *Parallel and Large-Data Visualization and Graphics, 2003. PVG 2003. IEEE Symposium on*, pp. 25–31. IEEE.

Moreland, K., B. Wylie, and C. Pavlakos. 2001. "Sort-last parallel rendering for viewing extremely large data sets on tile displays". In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pp. 85–92. IEEE Press.

Open MPI 2017. "Open MPI: Open Source High Performance Computing". https://www.open-mpi.org. [Online; accessed 10-April-2017].

MPICH 2017. "MPICH". https://www.mpich.org. [Online; accessed 10-April-2017].

Moses Palmér 2014. "pynput Package Documentation". https://pynput.readthedocs.io/en/latest/. [Online; accessed 17-April-2017].

Reutebuch, S. E., H.-E. Andersen, and R. J. McGaughey. 2005. "Light detection and ranging (LIDAR): an emerging tool for multiple resource inventory". *Journal of Forestry* vol. 103 (6), pp. 286–292.

Richardson, T., Q. Stafford-Fraser, K. R. Wood, and A. Hopper. 1998. "Virtual network computing". *IEEE Internet Computing* vol. 2 (1), pp. 33–38.

Samanta, R., T. Funkhouser, K. Li, and J. P. Singh. 2000. "Hybrid sort-first and sort-last parallel rendering with a cluster of PCs". In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp. 97–108. ACM.

Schroeder, W. J., K. M. Martin, and W. E. Lorensen. 1996. "The design and implementation of an object-oriented toolkit for 3D graphics and visualization". In *Proceedings of the 7th conference on Visualization'96*, pp. 93–ff. IEEE Computer Society Press.

Sourceforge 2016. "What is Ganglia?". http://ganglia.sourceforge.net. [Online; accessed 14-April-2017].

Jeff Squyres 2014. "Can I MPI_SEND (and MPI_RECV) with a count larger than 2 billion?". http://blogs.cisco.com/performance/can-i-mpi_send-and-mpi_recv-with-a-count-larger-than-2-billion. [Online; accessed 17-April-2017].

TurboVNC 2017. "A Brief Introduction to TurboVNC". http://www.turbovnc.org/About/Introduction. [Online; accessed 10-April-2017].

VirtualGL 2017. "A Brief Introduction to VirtualGL". http://www.virtualgl.org/About/Introduction. [Online; accessed 12-April-2017].

AUTHOR BIOGRAPHIES

FAIZ ABIDI graduated with a Master's in Computer Science from Virginia Tech in 2017.

NICHOLAS POLYS is the Director of Visual Computing at Virginia Tech Advanced Research Computing and an Affiliate Professor in the School of Engineering and the Department of Computer Science.

SRIJITH RAJAMOHAN is a Computational Scientist at ARC at Virginia Tech.

LANCE ARSENAULT is a visualization and virtual reality specialist at ARC at Virginia Tech.

AYAT MOHAMMED is a postdoctoral fellow at TACC's Scalable Visualization Technologies group.