Parameter Space Visualization for Large-scale Datasets Using Parallel Coordinate Plots

Kurtis Glendenning; Wright State University, 3640 Colonel Glenn Hwy, Dayton, Ohio, USA
Thomas Wischgoll; Wright State University, 3640 Colonel Glenn Hwy, Dayton, Ohio, USA
Jack Harris; Wright Patterson Air Force Base; Dayton, Ohio, USA
Rhonda Vickery; Engility Corporation; Dayton, Ohio, USA
Leslie Blaha; Wright Patterson Air Force Base; Dayton, Ohio, USA

Abstract

Visualization is an important task in data analytics, as it allows researchers to view patterns within the data instead of reading through extensive raw data. Allowing the ability to interact with the visualizations is an essential aspect since it provide the ability to intuitively explore data to find meaning and patterns more efficiently. Interactivity, however, becomes progressively more difficult as the size of the dataset increases. This project begins by leveraging existing web-based data visualization technologies and extends their functionality through the use of parallel processing. This methodology utilizes state-of-the-art techniques, such as Node.js, to split the visualization rendering and user interactivity controls between a client-server infrastructure without having to rebuild the visualization technologies. The approach minimizes data transfer by performing the rendering step on the server while allowing for the use of HPC systems to render the visualizations more quickly. In order to improve the scaling of the system with larger datasets, parallel processing and visualization optimization techniques are used. This work will use parameter space data generated from mindmodeling.org to showcase our methodology for handling large-scale datasets while retaining interactivity and user friendliness.

Keywords: Parameter space visualization, parallel coordinate plot, parallel processing, D3, JavaScript, web browser visualization

Introduction

The ability to do rapid visual assessments of parameter spaces has the potential to change the workflow for both model simulation and model fitting/parameter recovery. It enables the rapid identification of input parameters that result in similar output data or model behaviors. This allows researchers to eliminate redundant input parameters for more efficient use of modeling and simulation computational resources. For example, should two parameters exhibit a strong correlation, one might be held constant while the other varied in order to capture all the unique model behaviors. Further, early visual assessment of the parameter space means that ineffective or incorrect models may be rapidly identified and eliminated from study. This again results in effective use of both experimenter and computational time. Finally, parameter space visualizations can reveal unexpected relationships between the parameters and model behavior. If the behavior is incorrect, errors in model design or in model may be more easily found. If the behavior is novel, parameter space visualization will have resulted in new hypotheses or expanded research findings.

Web-based visualizations are of interest in this application area as they can be directly integrated into the high-performance computing (HPC) environment. At the same time, they can make the implementation and use of parameter space visualization easy for any level of visualization programmer. The potential for interacting with the data and feeding any resulting visually-identified parameter constraints directly into the modeling and simulation process would further improve the modeling workflow.

The approach described in this paper targets this area: web-based visualization directly integrated with the HPC job scheduling environment, optimized for a fast and interactive user experience. It is based on existing visualization tools, such as Data-Driven Documents (D3), combined with Node.js to devise a parallel implementation for maximal performance. Specifically, parallel coordinate plots worked well in the past for identifying correlations between variables so it was chosen as the first prototype visualization algorithm for this framework[16]. While standard tools, such as D3 and Plotly already provide common visualization algorithms, such as parallel coordinate plots, the amount of data these tools can handle is typically limited. In our experiments, datasets that exceed 500,000 data points can no longer be handled by these tools. Hence, an approach is needed that is capable of handling datasets beyond that limit. There are multiple bottlenecks that need to be overcome for this to be accomplished. Specifically, the amount of memory available on the system needs to be considered as well as the computational resource available. At the same time, transferring the dataset from the server to client can take a considerable amount of time. By utilizing a parallel server-side approach, all these bottlenecks can be avoided. The parallel approach only processes parts of the data at a time thereby reducing the memory footprint. Using the HPC resource directly provides more computational resources to the visualization algorithm. At the same time this approach avoids the transfer of the entire dataset and instead only requires a significantly smaller amount of data to be transferred from the server to the client. Overall, this enables the approach to process significantly larger datasets in a shorter period of time.

Related Work

Big data has been a booming topic for several years. Visualization is one key aspect of solving today's big data challenges as it is necessary to understand and analyze data efficiently. Traditional visualization techniques do not necessarily suffice for big data. They are often not equipped to handle large sample sizes and generally do not account for data being too large to fit into main memory. The process of visualization needs to be revamped to accommodate the ongoing growth of data.

Using web-based techniques for visualization tools can help target a broader audience. Bostock et al. proposed a JavaScript library, called Data-Driven Documents (D3), which serves as a flexible infrastructure for many types of visualizations[2]. Our system will provide tools built on the D3 library.

An upcoming visualization service, known as Plotly, provides a number of web based visualizations with many customization tools available. This system uses a client-server model to produce highly interactive visualizations for data [10]. While the generalization of various visualizations provided a flexible and efficient way to view data in multiple types of plots, the rendering process was not capable of handling large datasets. When testing a 500,000 point dataset with Plotly, it failed to respond after several minutes of processing. The proposed methodology in this paper successfully rendered the largest test case that we were able to provide, consisting of 246 million points.

Data can come in many different forms and sizes. High-dimensional data, being one variation, is a bit more difficult to visualize due to the inability to physically see more than three dimensions. Parallel coordinates[14] have proven to be a very sufficient visualization technique for this task. The basic idea is to present N dimensional data of the attribute space by mapping N equidistant parallel axes to the two-dimensional space. The axis of each dimension represents a property. The axis of the corresponding attribute values range from minimum to maximum uniform distribution. Thus, each data item can be used in accordance with its property values in a line segment on N parallel axes.

When large datasets are visualized using parallel coordinates, it can cause confusion due to a large number of overlapping lines. For this Peng has presented the concept of clutter-based dimension reordering. This concept allows the algorithm to reduce the clutter of parallel coordinate plots without sacrificing information in the visualization[9]. Siirtola has introduced two browser based techniques for manipulating parallel coordinate plots[11]. The first technique uses polyline averaging to summarize a set of polylines. The second provides a visualization for correlation coefficients between polyline subsets in order to help the user discover new information. Zhao et al. proposed a technique of rearranging variables to better identify patterns of interest[15]. This work contained a query tool that enabled the user to describe a specific target pattern to be displayed. Johansson et al. introduced a method to simultaneously examine the relationship of a single dimension to many dimensions. To allow the user to quickly view different combinations of dimensions, the single dimension being used can interactively be swapped with another[7]. Hauser et al. have demonstrated and expanded on some of the intuitive features of parallel coordinate plots[5]. Some of the features presented in these works are used in the proposed system to showcase that our methods can accommodate more advanced visualizations.

One of the many difficulties of visualizing big data is that traditional visualization techniques require all of the data to be held in memory. Ahrens et al. have developed a methodology for handling datasets that are too large to fit into memory. They accomplish this by streaming data to the visualization, eliminating the size limitation and gaining some efficiency from running visualizations in a small memory space resulting in higher cache hits[1]. Streaming data is used in the work described in this paper alongside data chunking in order to compare efficiency. These two methods of breaking down data have resulted almost identical in speed. Out–of-core techniques use memory only as their secondary storage medium. All data is maintained on the hard drive and the main memory serves as cache for that data. As such these techniques allow the visualization algorithm to be able to process datasets that exceed the main memory[13].

In some cases, big data visualizations are created by either rendering subsets of data or by mining features of data and rendering those results. Goecks et al. have developed Trackster, which is a tool that couples analysis and visualization to allow interactive visualizations for large datasets[4]. While this can efficiently produce a visualization, it does not actually render a large amount of data. In fields of study that are still in their early stages, such analysis tools for mining and subsetting may not exist, therefore making this technique ineffective. Instead these researchers are trying to view the entire parameter space in order to develop the generalizations of their data.

Pretorius et al. have created a system for exploring parameter spaces for image analysis. In this work, the paradigm of parameter sampling is changed in order to incorporate large parameter sweeps in a more efficient way[11]. The proposed system extends this methodology by doing parameter sweeps using multiple models and comparing their outputs through a parallel coordinates visualization.

Zhou et al. developed several web-based visualization frameworks combined with pre-processing tools to provide a way for domain specialists to interpret their data[16]. The framework contained parallel coordinate plots and heat maps that could be used to present identification-confusion matrix data.

Methodologies

The software portion of this system was developed from an open source parallel coordinate plot library built on top of Data-Driven Documents (D3). The baseline implementation will be overviewed, followed by the details for the performance improvements including client-server modelling, parallel rendering and line binning.

Baseline

The baseline of this implementation uses Parcoords, an open source D3 library specifically designed for building parallel coordinate plots. Parcoords is a client-side JavaScript library that internally manages the creation of HTML tags, data manipulation, and rendering[8]. Many of the basic tools associated with parallel coordinate plots can be used with a simple flag on instantiation.

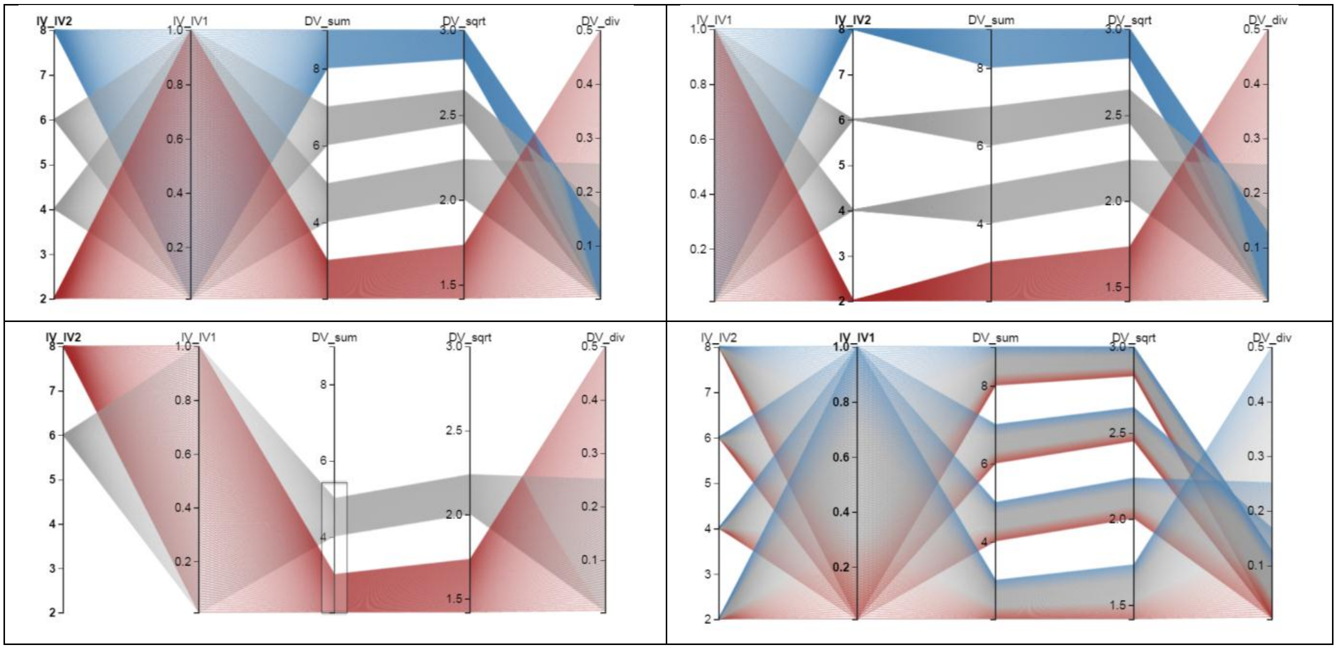

Figure 1. Photos of the resulting visual after making changes to the original plot, which is represented in the top left. The top right has reordered IV_IV1 and IV_IV2. Bottom left has brushed a section of DV_sum. Bottom right has changed the statistical coloring to interpolate IV_IV1.

Some of these features include reordering, removing, brushing, and statistical coloring of axes. Figure 1 illustrates these interaction features and their results on the visualization that are described in more detail below as well. Using any of these features will automatically refresh the visualization with the new parameters.

Reordering is a simple feature that allows the users to organize the axes to their liking. To do this, the user can click and drag an axis to a point in between two other axes. The plot will then be refreshed with the new arrangement. When two dimensions are not direct neighbors, it is sometimes hard to see their relationship. This feature provides a way to select which axes are adjacent to provide the most useful insight. Reordering axes can also help to better organize the plot for a cleaner visualization.

Some axes may prove to be of less importance to the user. Removing these will both minimize the clutter in the visualization and improve the refresh speed by reducing the amount of the data being rendered. Parcoords provides an API method for removing axes, which is connected to a list of existing axes. The user can toggle each of the axes individually to remove or reintroduce them.

Brushing is a very powerful tool for parallel coordinate plots. It allows the user to select a portion of an axis, or multiple axes. This will then translate to upper and lower bounds based on the scale of the axis being brushed. These bounds are used to limit the number of data points being rendered to the plot. By using this tool, a user can select a range of values on one axis to more clearly see where they fall on the other axes. While rendering with active brushes, the system will skip any data point with a value outside of these bounds, drastically reducing both the clutter of the plot and the rendering time.

Lastly, another powerful tool is statistical coloring. This feature provides a color scheme for the lines representing each tuple in the dataset. The color scheme is based on a single axis specified by the user via clicking the name label at the top. Then by mapping the value of a tuple for the specified axis to a color range, the color of each line will reflect the tuple's position on that axis. As mentioned before, it is hard to see the relationship of two dimensions when they are not directly neighboring. Statistical coloring provides a way to compare one specified axis with all other dimensions in the plot.

Altogether Parcoords provides an easy to use JavaScript API for creating parallel coordinate plots with some commonly used features. The infrastructure of using this library out of the box is sufficient for very small datasets, but presents several major problems when moving into larger datasets. Since this is a client-side JavaScript library, the users will need to download the data they wish to plot from a database server. This becomes unreasonable even when talking about data as small as one hundred megabytes, while most times large-scale datasets start at gigabytes or terabytes. Even if the user manages to wait for such a large dataset to download, most client machines are not equipped to handle data manipulation and rendering at this scale. The following sections will discuss the new infrastructure model and performance improvements used to allow the system to handle larger datasets.

Client-Server Model

The first problem addressed is the overwhelming data transfer. To reduce the network load this system moves the rendering step to the server where the data is stored. By utilizing a state-of-the-art engine like Node.js, the same JavaScript files used on the front-end can be run on the back-end. Another benefit of using Node.js is that it provides a module, jsdom, which simulates a web browser environment. With these technologies the implementation of the rendering methods does not need to be altered because it thinks that it is rendering to an ordinary webpage.

In order to retain the controls on the user interface of the webpage, some modifications were made to the visualization library. Parcoords is built for the client-side to manage both the HTML tags and image rendering. In order to separate the controls from the data management and visualization refresh, this library was split into two files, one for the server and one for the client. Parameters are generated from user input of reordering, removing, brushing, or statistically coloring axes and are given to the server as input in order to generate the refreshed image.

Parcoords generates two components in the rendering step. These are the visualization and the dimension axes. The visualization is rendered to an HTML canvas, which temporarily stores the image in memory. The axes are scalable vector graphic (svg) elements, which is required for enabling the control features. The svg tags can directly translate to a string to be sent to the client, but the canvas must be converted to a static image format. Another Node.js module, canvas, is used to convert the temporary canvas image into a string format. These two strings then become the output that the server sends to the client. The client script can reconstruct the visualization on the webpage by simply pasting the DOM string into a designated div element and drawing the image string to an underlying canvas. Initially, the visualization library generates an image and then waits for input to refresh. Since the redesigned workflow only generates one image per run, the server script will terminate immediately after the first render and return the output results to the client.

Parallel Rendering

In most visualizations, rendering one part of the image is independent of rendering another part. In the case of parallel coordinate plots, each line and even each individual line segment can be considered independent of one another. By exploiting this

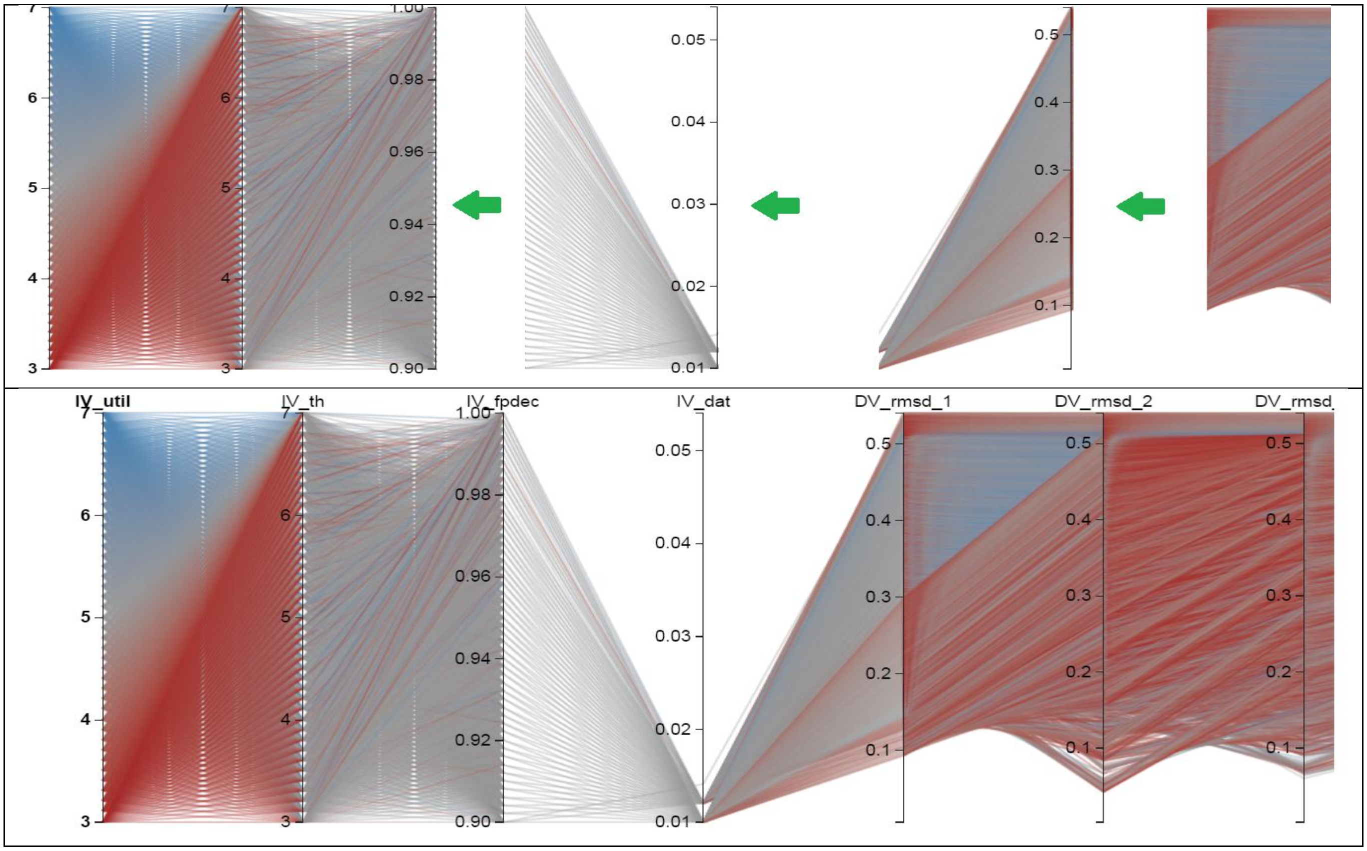

Figure 2. Demonstration of how the database chunking is performed to allow parallel rendering. By dividing by columns, the processes do not overlap in the visualization space, making the merge simple and efficient.

fact one can distribute the work among parallel rendering processes for dramatically faster rendering speeds.

Node.js is run on a single core, so threading will not produce parallel processing. Instead a node-module, child process, was used to fork new Node.js instances on a separate processing core. This required some modification of the server workflow. First the server rendering script was reinstated as the child process, while a master script was developed to fork sub processes and manage the distribution of work. Inter-process communication can create substantial overhead if a large amount of data is transferred. In order to keep this at a minimum, each process queries for its own data instead of the master process distributing the data to the children.

As it was stated before, each line rendered is independent of another allowing work distribution to be flexible. Two general approaches to distributing work would be to group by number of rows or number of columns. To divide by number of rows, the master script would have to render a full sized image in each process since the position on the axes where the lines will fall cannot be predetermined. If divided by columns, the length of the image rendered in each process can be shortened, creating a smaller amount of overhead for communicating results. For this reason, distributing by columns is primary, although distributing by rows can still occur if necessary. This would occur in a case where number of rows is large but number of columns is small. Figure 2 shows a graphical representation of how the task gets subdivided among the processes. Hence, dividing the image to be created allows the master script to divide the rendering task between the cores in a relative straight forward fashion. Sub processes are forked to create a new rendering task to generate the sub-image between two axes of the parallel coordinate plot. Only in cases where this would create an imbalance on the load of the involved processes, which typically only occurs toward the end of the rendering step with larger number of columns, the rendering task may get broken up by rows to ensure an even distribution of the load.

In the master script, a global parameter is set to specify the number of processing cores to use. Using this value a controller manages when to start new processes and handle output. Experiments were conducted to identify optimal strategies for load balancing. The growth rate of data points to render time is linear, inferring that many small workloads will not yield better results than fewer large processes. Creating more processes than the number of allocated processing cores will be slower since some processes will have to wait for an open core to run on. To further improve the efficiency, the controller evenly spreads the work between the processing cores so that none are substantially slower than the rest. Basically, there are different approaches of balancing the load among the processors. On the one hand, one could create one process per processor and task that process with generating as many subsections of the image as necessary. On the other hand, one process could be generated for each subsection of the image. In either case, the master script needs to monitor the processors involved to either create a new process or task an existing process with a new subsection. Due to the fact that typically the number of axes is significantly larger than the number of available processors in our test scenario, balancing the load this way works very effectively and only toward the end of rendering the image the rendering tasks have to be broken up by row. Both parallelization approaches mentioned above were implemented and tested. As can be seen from the results it turns out that creating less number of individual process improves the rendering speed considerably as it avoids the overhead of creating and removing a larger number of processes.

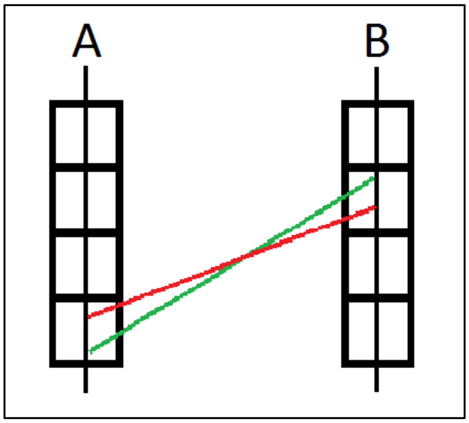

Figure 3. Precision of rendering can at best draw a line from a pixel in axis A to a pixel in axis B. Although the actual point values in the red and green line are slightly different, they will be drawn in the exact same place in the visualization.

To further improve the overall performance of the approach, different access schemes were implemented to fully support larger datasets. When there are many columns and many rows, it may occur that one process does not have enough memory to fit an entire column of data. In this case the system will stream data from the database, handling one data point at a time. This reduces the amount of memory that a process needs to store and negates the idle time in transferring data from the database to the process as a data structure. In practice this methodology has proved to be almost equivalent in speed and scale as taking the whole dataset as a chunk, but it allows for dataset sizes to be larger than the process memory.

A side effect of taking the streaming route is that the scaling for the axes must be done outside of the Parcoords library since it will not have all of the data available at once. Scaling for each axis is performed by mapping the range between the max and min values to the height of the image being produced. For most cases this is a fast query, but in testing with an SQL database (in this case MySQL version 14.14, distribution 5.6.12) it was found that unindexed columns in a database are extremely slow at finding max and min. To account for datasets that have this many dimensions, the max and min are saved on the client side so that this lookup only occurs once as a preprocessing step before the first render.

Line Binning

Constructing visualizations is a very slow process for a computer when comparing to simple math on a processor. Often times improving rendering speed consists of identifying criteria to find shortcuts around drawing every component. In the case of parallel coordinate plots, one can recognize that there are a limited number of lines between any two given axes for a given image resolution.

This provides a way to reduce the number of lines drawn by avoiding rendering the same line twice.

Parallel coordinate plots have a static height value which translates into a number of pixels on the screen. Each line segment in this visualization is drawn from one axis to the next or from a pixel in axis A to a pixel in axis B as shown in Figure 3.When scaling a value to the axis in the visualization, it often occurs that two or more lines will fall in the same location, i.e. connect the exact same pixels on the respective axes. As a result, the user is not able to distinguish one from the other. Thus we can infer that the maximum number of unique lines that can exist is equal to the product of the height of two axes, measured in pixels. Our system draws a height of four hundred pixels making the maximum number of unique lines 400 by 400, or 160,000. This of course is only the worst case scenario. On average the algorithm would have to draw much fewer lines depending on the variety of the data.

In order to identify the fact that there is multiplicity for a given line, the algorithm sets up a table based on the height values on each of the two involved axes. In this specific implementation the height values can range from 0 to 400 so that the overall number of possible combinations is 160,000. A hash-function is used to translate the height values of the two end points of a line to the index within this table to identify whether this line was encountered before or not. Since this test is merely based on the end points of the line, it is fairly efficient, especially when compared to the rendering process for the entire line using, for example, the Bresenham algorithm. Due to the fact that additional lines can effectively be skipped entirely by this simple look-up, the computational savings can be quite significant.

When using statistical coloring, it does not suffice to draw the first line and skip the rest. In order to retain an accurate color scheme, the system must accumulate the average color value of each existing line. The hexadecimal color value of each line can be converted into an integer and used to efficiently calculate an average. Once finished iterating through the data, each existing line is drawn once using the averaged color value. This improvement has a very strong effect on large datasets as it changes the growth rate to match that of integer addition instead of canvas rendering. Whether the dataset contains 200k, 1,000k or even 1,000,000k points, the maximum number of lines drawn will be 160k. It is important to note that the lines that are skipped are lines that would only be drawn multiple times with the exact same start and end point, i.e. the exact same line. By accumulating the color values of all these identical lines, the final image is in no way different from the one that would have been obtained by drawing all lines individually; it only saves on computational time.

Interactive Components

When visualizing data with parallel coordinate plots, the interactive features that allow one to analyze the data further by specific interactive features, such as eliminating columns, brushing along one axis to select sub-sections of the data, or applying color coding. These features are key to the ability to successfully investigate the dataset at hand. It is important to note that this parallel rendering approach retains those abilities. Axes of the parallel coordinate plots are listed in a separate section right below the visualization where the user can enable and disable them specifically.

The rendering of the user interface elements is decoupled from the rendering of the actual parallel coordinate plots. The user interface elements are drawn using svg elements, whereas the visualization is added as an image. This decoupling allows the web interface to retain the interactive features. Hence, the user can still apply color coding or brush along an axis in exactly the same way as one would with the original, sequential implementation of the parallel coordinates plot. As such the user still has the capabilities one would expect for filtering to reduce the clutter that can occur in a parallel coordinate plot.

Results

The implementation outlined resulted in a fully functional web based visualization tool, connected to mindmodeling.org. Users can initiate this visualization tool by visiting the results section of the desired job and simply clicking the Refresh button. Immediately after opening the results tab, some options are available to the user in order to make specifications for the first rendering. These options include selecting active columns and a number of data points to display. After the first visual is created, other interactive options will be available on the parallel coordinate plot such as brushing and statistical coloring. To allow the user to make several modifications before updating the image, no refresh request will be sent until the user clicks the Refresh button again. Based on our experimental test runs, a render time estimator is shown below the plot that updates are every interaction with the controls.

The algorithms were tested on an Apple MacBook Pro with Intel Core i5 and 4GB of memory for the client. The server consists of 16 Intel Xeon processors running at 2.27 GHz with 32 GB of RAM. For the experiments, only 8 CPUs were used at a time to not completely block the server from performing any other tasks. The algorithm was tested with 8 different data sizes. Each test run was performed five times. The running time for each test run was mostly identical for each dataset size with almost non-existing variance.

The test data for this system resulted from large scale modeling and simulation of two computational cognitive models (Adaptive Control of Thought-Rational and the Linear Ballistic Accumulator). The goal of the study was thorough model comparison, so the simulations entailed wide sampling of the parameter spaces. This sample of data consisted of 246 million points on 155 dimensions, totaling 1.9 gigabytes of data. More details about the simulations can be found in Fisher et al.[3]. A number of dataset sizes were tested at each stage of development and recorded for discussion. While the number of data points is listed, it is important to note that there were 155 dimensions in the tested dataset. A high dimensional dataset is handled differently than a low dimensional one, although for this system the rendering speeds are relatively similar. The first stage recorded was using the standard visualization tools on the server side. Initially this only uses one core, so the results should be comparable to running on the client side without the network data transfer. After parallelizing the rendering process, two scenarios were considered. Minimizing the data distribution size and creating many processes versus maximizing the distribution size and creating few processes. The results of both are shown in Table 1, represented by min col and max col respectively. Lastly, line binning has been added to each of these to further compare rendering speeds.

The single core rendering speed has a very large growth rate and eventually breaks due to lack of memory. The baseline visualization

Table 1. Rendering times in seconds recorded for various stages of development. Red entries represent test cases where a linear growth rate was not followed.

Data Points	1-Core	8-Core, Min Col	8-Core, Max Col	8-Core, Min Col, Binning	8-Core, Max Col, Binning
1.55E+06 (12 MB)	29.1	100.7	8.8	99.3	6.4
3.88E+06 (30 MB)	74.4	105	15.8	99.8	8
7.75E+06 (60 MB)	148.6	118.3	30.4	103.1	12
1.55E+07 (120 GB)	288.6	153.2	53.8	114.2	18.5
2.33E+07 (180 GB)	437.6	194.7	82	121.3	23.7
3.10E+07 (239 MB)	3086	242.7	106.9	132.7	28.2
1.55E+08 (1.2 GB)	N/A	984.8	593.8	334.4	112.2
2.46E+08 (1.9 GB)	N/A	1581.6	958.1	498.2	171.5

tool (*1-core* in Table 1) can render 23 million points in roughly 437 seconds. The performance numbers for these single core runs are effectively identical to the rendering times that can be expected from the original parcords implementation as it is using the exact same code. These numbers were used as the baseline instead of the ones obtained via running parcoords on the client to ensure the same type of computing environment is used. When using parallelization, the system becomes capable of rendering any size of data and can render small data sizes quickly. The downside is that the growth rate of speed is still quite large. This stage (*8-core, Max Col* Column in Table 1) can render the same 23 million point dataset in 82 seconds, an improvement of 4 times. *8-core, Min Col* represents parallelization using many processes of minimal size. It is obvious that the overhead from creating more processes harshly affects the rendering speed since rendering 23 million points takes over 120 seconds as opposed to 82 using *8-core, Max Col*. Line binning slightly reduces the small dataset speeds, but greatly reduces the growth rate. This result backs the methodology discussed and is capable of rendering 23 million data points in only 23.7 seconds when combined with maximizing data distribution, an improvement of 20 times over the baseline system. The largest dataset tested on the system contained 246 million points and successfully rendered in 171 seconds. Figure 4 shows least-square-fitted lines for a plot of the numbers shown in table 1. This graph shows nicely how much faster the parallel rendering performs as the dataset size grows. It should be noted that the performance of the Max Col approach performs too similarly no matter whether the data base is accessed using the streaming method or not for the dataset sizes tested so far to be distinguishable in the plot. This is why only one of the lines is visible in the graph.

The improvement in performance of the visualization algorithm makes it easier to use for our collaborative partners thanks to the increased interactive capabilities and ability to process the larger datasets that were impossible to visualize using existing approaches. By using the current implementation of the described algorithm, our collaborators were already able to identify characteristics within the data which they were not able to do before. Due to the fact that it is directly integrated with the web interface that the users of the mindmodelling.org system use to track the progress of their computations, the visualization is ready to use within that same interface. As a result it is very easy to use and ready to deploy by a relatively large user base.

While this system is demonstrated with parallel coordinate plots for parameter space data, the general concept can also be applicable to other types of visualizations. The distribution of work in the parallelization process will generally be specific to the type of visualization but the infrastructure can be applied very broadly.

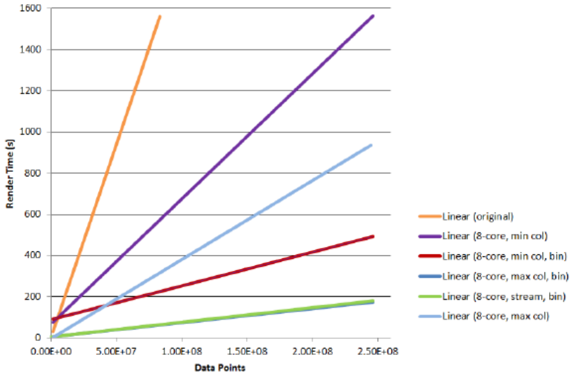

Figure 4. Rendering times in seconds versus data set size for the different algorithms tested.

Future Work

In the future we will extend our framework to include additional visualization algorithms thereby expanding the capabilities as well as providing further functionality to our user base. Moreover, the framework will be scaled up so that it is able to take advantage of more computational resources. We have an in-house high-performance cluster available to this project that consists of 2048 parallel cores. We expect an improvement in performance by fully utilizing this computational platform. Further potential future work lies in the utilization of GPUs as highly parallel computational resource that can be used to enhance the performance of the framework.

Conclusion

Visualization is a task that, like many, becomes increasingly difficult when moving into large-scale datasets. This work has demonstrated our methodology for transforming a typical web based visualization library into a client-server model. By leveraging HPC resources, we were able to parallelize the rendering process to effectively handle large datasets. Our experiments have shown that using only eight parallel cores, we were able to render a plot 20 times faster than the baseline implementation originally took. The largest test case for this system, containing over 246 million data points, successfully rendered in 171 seconds on eight cores. By moving the visualization step to the server end, network transfer has been reduced to the size of a typical image per refresh. Lastly, by utilizing a state-of-the-art technology, Node.js, we were able perform this task using an existing browser based visualization library. Overall, this approach was able to preserve the interaction paradigms provided by the original algorithms with the added capability of being able to handle significantly larger datasets while providing better rendering performance at the same time.

Acknowledgements

This research was supported by the DoD HPC Modernization Program under award number GS04T09DBC0017. Views and opinions of, and endorsements by the author(s) do not reflect those of the US Army, Air Force, or the Department of Defense. The authors would like to thank the collaborators at the Air Force Research Laboratory at Wright Patterson Air Force Base. The authors also wish to thank Wright State University and the CECS department for their support and facilities.

References

[1] Ahrens, J., K. Brislawn, K. Martin, B. Geveci, C.c. Law, and M. Papka. "Large-scale Data Visualization Using Parallel Data Streaming." IEEE Computer Graphics and Applications IEEE Comput. Grap. Appl. 21.4 (2001): 34-41. Web.

[2] Bostock, M.; Ogievetsky, V.; Heer, J., “D³ Data-Driven Documents”, IEEE Transactions on Visualization and Computer Graphics, vol.17, no.12, pp.2301,2309 (2011).

[3] Fisher, C. R., Walsh, M., Blaha, L. M., &Gunzelmann, G. (2015, July). ACT-R and LBA model mimicry reveals similarity across levels of analysis. 37th Annual Conference of the Cognitive Science Society, Pasadena, California.

[4] Goecks, J, Coraor N, The Galaxy Team, Anton Nekrutenko, and James Taylor. "NGS Analyses by Visualization with Trackster." Nat Biotechnol Nature Biotechnology 30.11 (2012): 1036-039. Web.

[5] Hauser, H., F. Ledermann, and H. Doleisch. "Angular Brushing of Extended Parallel Coordinates." IEEE Symposium on Information Visualization, 2002. INFOVIS 2002. (2002): n. pag. Web.

[6] Inselberg, A., andDimsdale, B. “Parallel coordinates for visualizing multi-dimensional geometry” Springer Japan, 25-44 (1987).

[7] Johansson, J., Cooper, M., andJern, M. “3-dimensional display for clustered multi-relational parallel coordinates”. In Information Visualisation, 2005. Proceedings. Ninth International Conference, 188-193 (2005).

[8] Parallel-Coordinates, “Parcoords,” https://syntagmatic.github.io/parallel-coordinates/ (2014).

[9] Peng, W., Ward, M. O., &Rundensteiner, E. A. “Clutter reduction in multi-dimensional data visualization using dimension reordering.” In Information Visualization, 2004. INFOVIS 2004. IEEE, 89-96 (2004).

[10] Plotly, “Plotly.js,” https://plot.ly/ (2013).

[11] Pretorius, A. J., Mark-Anthony P. Bray, A. E. Carpenter, and R. A. Ruddle. "Visualization of Parameter Space for Image Analysis." IEEE Transactions on Visualization and Computer Graphics IEEE Trans. Visual. Comput. Graphics 17.12 (2011): 2402-411. Web.

[12] Siirtola, H. “Direct manipulation of parallel coordinates”. In Information Visualization, 2000. Proceedings.IEEE International Conference, 373-378 (2000).

[13] Silva, C., Chiang, Y., Corrêa,W., El-sana, J., Lindstrom, P., “Out-of-core Algorithms for Scientific Visualization and Computer Graphics.” Visualization ’02 Course Notes.(2002).

[14] Ward, M. “XmdvTool: Integrating multiple methods for visualizing multivariate data.” In Proceedings of the Conference on Visualization ’94, pp.326-333, IEEE Computer Society Press, 1994.

[15] Zhao, K., Liu, B., Tirpak, T. M., & Schaller, A. “Detecting patterns of change using enhanced parallel coordinates visualization.” In Data Mining, 2003. ICDM 2003. Third IEEE International Conference, 747-750 (2003).

[16] Zhou, Y., Wischgoll, T., Blaha, L. M., Smith, R., and Vickery, R. J.. "Visualizing Confusion Matrices for Multidimensional Signal Detection Correlational Methods." Visualization and Data Analysis 2014 (2013): n. pag. Web.

Author Biography

Kurtis Glendenning received his BS in Computer Science from Wright State University (2013) and is currently a MS student in Computer Science at Wright State University. Kurtis has been involved in various research projects in Wright State’s Computer Science department including topics such as Visualization and Data Analysis.

Dr. Thomas Wischgoll received his PhD from the University of Kaiserslautern in 2002. He was working as a post-doctoral researcher at the University of California, Irvine until 2005 and is currently an associate professor and the Director of Visualization Research at Wright State University. His research interests include large-scale visualization, flow and scientific visualization, as well as biomedical imaging. His research work in the field scientific visualization and analysis resulted in over fifty peer-reviewed publications.

Dr. Jack Harris is the Director of the MindModeling.org project and works for the Warfighter Research Readiness Division, 711th Human Performance Wing, Air Force Research Laboratory. He graduated from Indiana University in 2011 with a joint Ph.D. in Computer Science and Cognitive Science, specializing in automated methodologies for cognitive model evaluation.

Dr. Rhonda Vickery is a visualization specialist with Engility Corporation. She provides leadership on the DoD High Performance Computing Modernization Program for the User Productivity Enhancement, Technology Transfer and Training Program at the Air Force Research Laboratory DoD Supercomputing Research Center in Dayton, OH. She has over 20 years experience in the area of visualization and has over 12 years experience working with DoD high performance computing environments.

Dr. Leslie Blaha is an Engineering Research Psychologist in the Battlespace Visualization Branch, 711th Human Performance Wing, Air Force Research Laboratory. She graduated from Indiana University in 2010 with a joint Ph.D. in psychology and cognitive science, specializing in mathematical psychology.