Browser Based Visualization for Parameter Spaces of Big Data Using Client-Server Model

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

by

Kurtis M. Glendenning
B.S.C.S., Wright State University, 2013

2015
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Kurtis M. Glendenning ENTITLED Browser Based Visualization for Parameter Spaces of Big Data Using Client-Server Model BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Thomas Wischgoll, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Thomas Wischgoll, Ph.D.

Michael Raymer, Ph.D.

Derek Doran, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT


Visualization is an important task in data analytics, as it allows researchers to view abstract 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 provides 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. 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.
## Contents

1 Introduction .................................................. 1
   1.1 Background .............................................. 1
   1.2 Related Work ........................................... 2
   1.3 Purpose ................................................ 5

2 Underlying Concepts ........................................ 8
   2.1 Parallel Coordinate Plots ............................... 8
   2.2 Client-Server Model .................................. 10
      2.2.1 NodeJs ............................................... 10
   2.3 High Performance Computing .......................... 11

3 Implementation .............................................. 14
   3.1 Baseline ................................................ 14
   3.2 Client-Sever Infrastructure ............................ 16
   3.3 Parallel Rendering ...................................... 19
   3.4 Line Binning ............................................ 24

4 Discussion .................................................. 26
   4.1 Results ................................................ 26
   4.2 Future Research ......................................... 30
      4.2.1 Additional Visualizations ......................... 30
   4.2.2 Hardware .............................................. 30
   4.2.3 Software .............................................. 31

5 Conclusion ................................................ 32

Bibliography .................................................. 33

A An Example Appendix ......................................... 36
   A.1 Parallel Coordinate Plot Features ...................... 36
## List of Figures

1.1 Parameter space of a typical cognitive modeling job. .......................... 6
2.1 Parallel coordinate plot for car data ................................................. 9
2.2 Client-server model diagram .......................................................... 10
2.3 Growth of performance in supercomputers ........................................ 12
3.1 Screenshots of the resulting visual after making changes to the original plot 15
3.2 Standard infrastructure of D3 library ................................................. 17
3.3 Client-server infrastructure with Nodejs ............................................. 18
3.4 Full user interface ........................................................................... 19
3.5 Parallel rendering infrastructure ....................................................... 20
3.6 Demonstration of how the database chunking is performed to allow parallel rendering ................................................................. 21
3.7 Keeping all cores active will reduce overall rendering time ............... 23
3.8 Precision of rendering ..................................................................... 25
4.1 Growth rate of rendering speeds ....................................................... 28
A.1 Original ......................................................................................... 36
A.2 Reordered ...................................................................................... 37
A.3 Brushed ......................................................................................... 37
A.4 Color interpolation .......................................................................... 38
List of Tables

4.1 Render times recorded for various stages of development. Red entries represent test cases where linear growth rate was not followed. . . . . . . . . . 27
Acknowledgment

I would like to take this opportunity to extend my thanks to my Advisor, Dr. Thomas Wischgoll, for providing me with this opportunity and for his consistent support and guidance. Dr. Michael Raymer and Dr. Derek Doran for their input and for serving on my thesis committee. Thank you Dr. Mateen Rizki and the rest of the Computer Science and Engineering department faculty for enabling this opportunity at Wright State University.
Dedicated to

Alina Dieli, the love of my life

my parents, Sarah, Jeff and Michelle.
Introduction

This section introduces the project by providing contextual information, reviewing related works, and exploring the importance of the work. The background and related works will discuss the technical challenges involved as well as existing work that approaches these problems. The purpose subsection will then highlight the significance of the solution to these problems.

1.1 Background

The ability to do rapid visual assessments of parameter spaces has the potential to change the workflow for both model simulation and model fitting. Model fitting, also known as parameter recovery, is the process of finding which parameter combinations produce a desired output. This allows researchers to eliminate redundant input parameters for more efficient use of modeling and simulation computational resources. For example, should two parameters exhibit low correlation, one might be held constant while the other varied in order to reduce computation without losing 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 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.

A ongoing HPC resource portal project at the Air Force Research Laboratory (AFRL), called MindModeling, provides a fully web-based interface for scheduling jobs. This enables the use of HPC systems for researchers who need such resources to improve their workflow but lack the technical training to create and manage their own jobs. MindModeling was initially built to target users in cognitive and network modeling, but now is broadening its toolset to target a larger user base. To accompany these goals, this project seeks to implement a general infrastructure that also has the capability to broaden its features to support a wide variety of users.

1.2 Related Work

Big data has been a booming topic for several years. Visualization is one key aspect of this area that is necessary to understand and analyze data efficiently. Traditional visualization techniques do not suffice for big data. They are not equipped to efficiently render large datasets 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 [15]. 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 have proven to be a sufficient visualization technique for this task. The basic idea is to present N dimensions of the attribute space by mapping N equidistant parallel axes to the two-dimensional space. Each axis represents a property in the dataset. The corresponding attribute values range from minimum to maximum in uniform distribution. Each data tuple is then mapped across all axes intersecting each dimension at a position according to its property value.

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 [14]. Siirtola has introduced two browser based techniques for manipulating parallel coordinate plots [18]. 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 [20]. 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 [9]. Hauser et al. have demonstrated and expanded on some of the intuitive features of parallel coordinate plots [7]. 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, rendering only one piece of data at a time. This eliminates the size limitation and gains 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 [19].

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 [6]. 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 [16]. The work presented here 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 framework combined with pre-processing tools to provide a way for domain specialists to interpret their data [21]. The framework contained parallel coordinate plots and heat maps that could be used to present identification-confusion matrix data.

1.3 Purpose

The approach described in this paper targets this area: web-based visualization directly integrated with a high-performance computing (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. Being integrated into an HPC environment allows for the inherent use of the HPC resources and direct access to the data produced by the jobs. In the past, parallel coordinate plots worked well for identifying correlations between variables so it was chosen as the first prototype visualization algorithm for this framework [21]. 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, data sets that exceed 500,000 data points can no longer be handled by these tools. Hence, an approach is needed that is capable of handling data sets 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 resources 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.

This project is built into the MindModeling website in order to inherently target a large user base. Since MindModeling users are primarily involved in cognitive and network modeling, parallel coordinate plots are the visualization type chosen to test the infrastructure.

Cognitive modeling research entails creating, testing, and comparing models that attempt to simulate human perception and interaction. The type of experiments that are run involve sweeping a large set of parameters into several models to get a comprehensive understanding of the models’ behaviors. Figure 1.1 shows a common job submitted in MindModeling.

This dataset has five input parameters and simulates six different models, giving it a total of eleven dimensions. The user will often want to compare and contrast model behaviors with respect to their input parameters. In order to do this it is necessary to view all of the data in...
this job together. Most visualization types allow the user to view up to three dimensions, but lack a method for displaying a fourth (or greater) dimension. Parallel coordinate plots work especially well as a visual analysis tool for this type of job, as it provides a means of viewing a parameter space with more than three dimensions.
Underlying Concepts

This section will introduce the key concepts involved in this project. Parallel coordinate plot is the primary visualization used in this project. Client-server model, with the use of Node.js, is the infrastructure that is leveraged to make this solution possible. High performance computing systems are utilized in the implementation of this system.

2.1 Parallel Coordinate Plots

Parallel coordinate plot is a technique for visualizing multi-dimensional numerical datasets [11]. Most graphs and models can project two or three dimensions that human eyes can perceive. Some examples include scatter plots, bar charts, and heat maps. These however lack a way of representing a fourth dimension. Parallel coordinate plots are not limited by the number of dimensions, so long as the screen space or resolution is available. The trade off being made to achieve this is that the pairwise relationship of dimensions is partially limited to only adjacent axes in the plot. However, some features have been developed to account for this limitation.

The design of a parallel coordinate plot is straightforward. Each dimension is represented by one vertical axis; the scale of each axis is determined by the range of its values. Every tuple is then drawn horizontally, passing each axis at a point relative to the value of that dimension. As the size of data gets larger, this can turn into a jumble of lines. Many features have been discovered in order to account for this; several of these features will be
discussed in a later section. Parallel coordinate plots may seem overwhelming at first, but after some experience it is possible to recognize patterns and strength of correlations in a dataset very quickly [11].

In Figure 2.1, a sample parallel coordinate plot is given for a collection of car data [11]. The dimensions in this data consist of specifications that all cars have such as MPG (miles per gallon), number of cylinders and horsepower. This is necessary so that each tuple makes a complete span horizontally. Also notice that there are two labels for each axis; the top representing the maximum numerical value, and the bottom being the minimum value. All axes must have these in order to scale correctly. Before adding any visual features, one could easily see some correlation. For example, by evaluating the lines between the "Weight" and "Year" axes, it can be inferred that the weight of cars decreased between 1970 and 1982.
2.2 Client-Server Model

Client-server model is the most commonly used infrastructure for the web. It provides an easy way for service requests to be requested and fulfilled through a wide-area network (WAN). The client machine (desktop, laptop, phone, tablet, etc.) connects to a server to make a request. The server then processes the request and returns the data. The connection is then terminated. This infrastructure allows a server to provide shared data to many clients simultaneously. The diagram in Figure 2.2 shows this relationship [10].

![Client-server model diagram](image-url)

Figure 2.2: Client-server model diagram

2.2.1 NodeJs

JavaScript, a very powerful and very popular scripting language has been in the mix for web developers for many years. It is primarily used for programmatically modifying the document object model (DOM), which is the cross-platform structure for managing documents in html. JavaScript later evolved into being the entire framework for the client side of a web browser. Its popularity partly came from its simple and flexible nature, making it easy for novice programmers to adapt to [4]. Another attractive feature of JavaScript is its ability to handle graphical elements especially well. Many graphics libraries have been implemented in JavaScript. One in particular, data driven documents (D3), has simplified
visualization to a beginner’s level.

While JavaScript won its popularity through client-side scripting, new engines are coming out that compile JavaScript on the server-side. NodeJs is one of the most popular of these. It is a packaged compilation of Google’s V8 JavaScript engine, and provides a non-blocking, event driven I/O paradigm of programming [3]. Because this engine can parse JavaScript very similarly to the web browser, it provides an easy means of transferring a client-side script to the server-side.

Nodejs also comes with many helpful modules that keep it up to par with a web environment. One of these modules, jsdom, is leveraged in this project to simulate the html DOM framework so that libraries like D3 can be utilized. Since scripts can now be run on the server-side, Nodejs also provides modules for accessing popular databases, like SQL, directly. As a result, the script can bypass the previously needed middle-man script. Altogether, Nodejs provides a state-of-the-art and robust platform for building client-server applications.

2.3 High Performance Computing

One of the key benefits of using a client-server model is that an application may take advantage of the processing power of the server machine, which often has better performance than the client machine. Another advantage of servicing requests with a server machine is that it becomes much more cost effective to upgrade hardware. Upgrading a single server machine is far cheaper than upgrading many client machines. This ideology is the basis of high-performance computing (HPC) systems. HPC is the use of parallel processing for running applications more quickly and efficiently. This is sometimes used as a synonym for super-computing, although super-computing refers to the highest operational rate for the current computing state [17].

Computing power is often measured in floating point operations per second (Flops).
Figure 2.3: Growth of performance in supercomputers
A typical desktop computer, which often is the most powerful type of client machine can achieve upwards of 50 GFlops, or fifty billion operations per second. Today’s supercomputers can perform over 10 PFlops, which equates to ten quadrillion operations per second [12]. This is an incredibly large output, about 200,000 times faster than a client machine, turning days of computation into seconds. That fact is one reason that HPC is becoming more popular. Another cause of this can be seen in Figure 2.3 [8]. There are three lines on this chart. The red line (middle) represents the largest system available at the time. Orange (bottom) represents the 500th best computer, which would be more of an average HPC system. Finally, blue (top) represents the total computer power of all available systems. The primary detail to notice here is that the growth over the last twenty years is pretty constant. This is a very attractive broadcast for researchers [8]. The argument could be made that client machines are also experiencing a similar growth rate. The benefit of using HPC systems is that it is more cost effective to upgrade a single server machine than to upgrade all client machines.

The targeted user base for this project involves those that use HPC systems to produce their data. If the size of the job requires HPC resources then it is often imperative that the visualization would require these resources as well. The proposed system is built into the MindModeling to give users not just a convenient web based application, but also the benefit of the processing power of an HPC server.
Implementation

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.

3.1 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 [13]. Many of the basic tools associated with parallel coordinate plots can be used with a simple flag on initialization. Some of these features include reordering, removing, brushing, and statistical coloring of axes. Figure 3.1 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. Parcoods 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 tuples 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 tuple 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 tuples 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.

### 3.2 Client-Server Infrastructure

The first problem addressed is the overwhelming data transfer. The initial infrastructure, as shown in Figure 3.2, needs to request the data across the wire. For large datasets, this is impractical. 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. The redesigned infrastructure can be seen in Figure 3.3. The rendering process is now completely moved to the server. The transaction between the client and server is modified as well. The client now asks for a refresh, and includes a set of parameters which describes the active features to apply. The server script then uses these parameters to build refresh the visualization in the requested state.

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 as seen in Figure 3.3. 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.

Since the visualization library needed to be divided into client and server parts, some of the controls had to be modified as well. These changes are mostly for aesthetic reasons, as opposed to functional. An example of this was the change of how to remove axes from the visualization. The original tool would allow the user to drag on axis to the far left, causing it to remove the axis and automatically readjust to the remaining dimensions. Since the new system avoids automatic refreshes, allowing the opportunity to perform multiple changes before a new rendering, this had to be modified. A new control grid was developed for this feature, allowing the user to select or deselect any dimensions that they do not want to display. Figure 3.4 shows a screenshot of the full user interface of the application,
including the dimension grid at the bottom. By using this new control method, the user will not tamper the image currently being shown and can make and reverse multiple changes before a refresh.

### 3.3 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 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. Figure 3.5
shows a diagram for the final infrastructure, including the parallel distribution of rendering. 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. The results of each child process is combined by the mastery script and exported as a single png as in the previous infrastructure. This allows the input and output with the client machine to remain the same for simplicity.

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 either 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 primarily chosen, 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 3.6 shows a graphical representation of how the task gets subdivided among the processes. By dividing by columns, the processes do not overlap in the visualization space, making the merge simple and efficient.

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 performed to determine how much work to give to each child process. 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.

Another scenario was considered for dataset shape. 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 contains two options. The first is to stream data from the database, handling one tuple 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. The other option is to create multiple sets of processes; a set refers to the number of allocated cores. When the memory in one set of cores is filled up, the work is distributed across two (or more) sets to keep all cores active during the rendering process. If one extra process was created instead of a full set, then after the first set finishes, one core will begin a new process and the rest will be inactive. This ideology is shown in Figure 3.7. Both methods use 25 seconds of processing time, although splitting the work across two sets of cores takes only 7 seconds of wall time versus 10 seconds. Furthermore, this allows the system to split the work evenly across two sets of cores, avoiding any efficiency reduction other than the overhead of a new process. In practice these two methodologies have proved to be almost equivalent in speed and scale, however the streaming technique reduces the amount of memory used.

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 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
Figure 3.7: Keeping all cores active will reduce overall rendering time

(a) Time without filling a second process set

(b) Time using two equal sets of processes
step before the first render.

3.4 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 3.8. Although the actual point values in the red and green lines are slightly different, they will be drawn in the exact same place in the visualization. As a result, the user is not able to distinguish one from the other. This happens more frequently as the size of the dataset increases, due to the higher ratio of tuples to unique lines. Since the height of the axes are constant, 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. An average the algorithm would have to draw much fewer lines depending on the variety of the data.

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. 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.
Discussion

In this section the results of this project and the potential future directions are discussed. The performance improvements achieved by the redesigned infrastructure are presented and compared to satisfy the purpose of this project. Several directions of future work involving this project are also presented.

4.1 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 tuples 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 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
Table 4.1: Render times recorded for various stages of development. Red entries represent test cases where linear growth rate was not followed.

<table>
<thead>
<tr>
<th>Data Points</th>
<th>1-Core</th>
<th>8-core, Min Col</th>
<th>8-core, Max Col</th>
<th>8-core, Min Col, Binning</th>
<th>8-core, Max Col, Binning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.55E+06</td>
<td>29.1</td>
<td>100.7</td>
<td>8.8</td>
<td>99.3</td>
<td>6.4</td>
</tr>
<tr>
<td>3.88E+06</td>
<td>74.4</td>
<td>105</td>
<td>15.8</td>
<td>99.8</td>
<td>8</td>
</tr>
<tr>
<td>7.75E+06</td>
<td>148.6</td>
<td>118.3</td>
<td>30.4</td>
<td>103.1</td>
<td>12</td>
</tr>
<tr>
<td>1.55E+07</td>
<td>288.6</td>
<td>153.2</td>
<td>53.8</td>
<td>114.2</td>
<td>18.5</td>
</tr>
<tr>
<td>2.33E+07</td>
<td>437.6</td>
<td>194.7</td>
<td>82</td>
<td>121.3</td>
<td>23.7</td>
</tr>
<tr>
<td>3.10E+07</td>
<td>3086</td>
<td>242.7</td>
<td>106.9</td>
<td>132.7</td>
<td>28.2</td>
</tr>
<tr>
<td>1.55E+08</td>
<td>N/A</td>
<td>984.8</td>
<td>593.8</td>
<td>334.4</td>
<td>112.2</td>
</tr>
<tr>
<td>2.46E+08</td>
<td>N/A</td>
<td>1581.6</td>
<td>958.1</td>
<td>498.2</td>
<td>171.5</td>
</tr>
</tbody>
</table>

Simulations entailed wide sampling of the parameter spaces. This sample of data consisted of 246 points on 155 dimensions, totaling 1.9 gigabytes of data. More details about the simulations can be found in Fisher et al. [5]. 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. Table 4.1 shows the results (measured wall time in seconds) for each test performed, while Figure 4.1 shows the growth rates. 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 4.1, represented by min col and max col respectively. Lastly, line binning has been
added to each of these to further compare render speeds.

The single core render speed has a very large growth rate and eventually breaks due to lack of memory. The baseline visualization tool (1-core in Table 4.1) can render 23 million points in roughly 437 seconds. Once the baseline method attempts to render a significantly larger dataset, it is forced to do some storage swapping to avoid filling memory; this is demonstrated by the substantial increase in time at 31 million points. When testing 155 million points, the system refused to automatically handle swapping and aborted the operation. There are existing methods for manually caching data to perform out-of-memory visualization, although these are general much slower even when good caching strategies are used. When using the developed parallelization algorithm, the system becomes capable of rendering any size of data without the need of memory swapping 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 4.1) can render the same 23 million point dataset in 82

![Figure 4.1: Growth rate of rendering speeds](image)
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, as seen in Figure 4.1. This result backs the methodology discussed and is capable of rendering 23 million 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 4.1 also contains the results of the streaming tests. This line is almost directly stacked with the "8-core, max col, binning" line. As previously discussed, these performed nearly identical in speed.

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 data sets that were not efficiently handled previously. 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 to the web interface that the users of the mindmodelling.org system utilize to track the progress of their computations, the visualization is available 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.
4.2 Future Research

There is great potential for future work with this project. Various forms of improvement can be applied including expanding visualization forms, scaling up of the hardware, and advancement of the software features. The following sections will further discuss the possible advancements.

4.2.1 Additional Visualizations

The system was built into mindmodeling.org which primarily aids research in cognitive modeling. For this reason parallel coordinate plots were the primary focus for the type of visualization. As mindmodeling.org adapts to support more research areas, this project too has the potential to expand to other forms of visualization. The discussed methodologies can easily become the base infrastructure of other visual techniques.

4.2.2 Hardware

The system that this project was developed on contain sixteen processing cores. In the HPC world this is a very small cluster. Scaling the implementation up to a larger system could immediately see improvement by a significant factor. Wright State University has received an in-house high-performance cluster that consists of 2048 parallel cores. By fully utilizing this computational platform, a very large improvement is expected.

Other hardware improvements could lie in the utilization of graphical processing units (GPU). Not all HPC systems have these units available, but those that do could yield another great improvement in speed. GPUs are highly parallel computational resources that could greatly enhance the effect of parallel computation at low cost.
4.2.3 Software

Currently this infrastructure takes advantage of multiple processing cores on a single HPC system. This can be expanded to utilize resources from multiple systems as well. By using a network of systems to divide the work, jobs can be run with more resources without the need to centralize the data on a larger system. Some difficult questions still lie in this methodology relating to data location and transfer bottlenecks, but some potential research certainly exists.

While this project is primarily focused on the implementation of a high performance visualization infrastructure, there is also potential future work in the features of the visualization. When collaborating with some of the target users, a lot of interest was raised for potential feature exploitation that implicitly comes with visualizing large datasets. An example of this is the density of a cluster in the parallel coordinate plot. When multiple tuples with similar color values are drawn over a single area, the visualization space becomes slightly darker. This detail would allow the user to infer density of the cluster based on how dark the cluster’s color is.


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 to 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.
Bibliography


Appendix A

A.1 Parallel Coordinate Plot Features

Figure A.1: Original
Figure A.2: Reordered

Figure A.3: Brushed
Figure A.4: Color interpolation