2015-12-01

An Automated Tool for High Resolution Visualization Applied to Transient Watershed Models

Noah Robert Taylor
Brigham Young University - Provo

An Automated Tool for High Resolution Visualization

Applied to Transient Watershed Models

Noah Robert Taylor

A prospectus submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Norman L. Jones, Chair
E. James Nelson
Daniel P. Ames

Department of Civil and Environmental Engineering

Brigham Young University

December 2015

ABSTRACT

An Automated Tool for High Resolution Visualization
Applied to Transient Watershed Models

Noah Robert Taylor
Department of Civil and Environmental Engineering, BYU
Master of Science

Numeric hydrologic models can aid in water resource management by providing predictive simulations of water behavior. As computers become more advanced, the models developed also become more complex using more data to represent larger areas for forecasting hydrologic behavior. Unfortunately, as the simulations use more data, the output often becomes difficult to manage and share without investing time and effort into setting up server environments or decreasing the quality of the output to compensate for an efficient and effective user experience. A proposed solution to facilitate the accessibility of massive hydrologic model output is through the web-based visualization tool developed at Carnegie Mellon University called Time Machine. For a more efficient and automated workflow, a Python tool named TMAPS was developed from this research for rendering hydrologic model results, geoprocessing the rendered output, and generating Time Machines seamlessly. The tool can be installed from the CI-WATER GitHub repository and allows the user to 1) select the output parameters and visualization settings desired to be rendered, 2) run the code on a local or HPC setup, and 3) use a web browser interface to view the tiled transient results seamlessly while maintaining high quality. Currently, the only hydrologic model supported is ADHydro - a large-scale high-resolution multi-physics watershed simulation. In an effort to facilitate organizing the library of Time Machine products, an app was created through Tethys - a server-based Django application designed to aid in the development and sharing of water resource engineering apps.

Keywords: hydrologic model, Time Machine, web-based visualization, tiled video, TMAPS

ACKNOWLEDGEMENTS

I would like to thank the Department of Civil Engineering at Brigham Young University and all the faculty and staff who have contributed to my education and understanding of the engineering principles that I hope to use to go forth and serve. My family also deserves my appreciation as they motivate me to work hard and for their endless support.

I would like to thank CI-WATER for funding the research that I was involved with and for enabling me to build a professional relationship with great researchers at the University of Wyoming including Dr. Fred Ogden and Robert Steinke. My committee members including Dr. Norman L. Jones, Dr. E. James Nelson, and Dr. Daniel P. Ames also are much appreciated for their patience and encouragement throughout my research.

This research is based upon work supported by the National Science Foundation under Grant No. 1135483.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Water is in limited supply in many parts of the world. As populations grow and the demand for water increases, it has become much more important for water managers to forecast water budgets(Water 2009). Hydrologic models are used to aid in the management of water budgets in regions of interest. These models represent the hydrologic cycle and can be used to better understand where the water goes. The usefulness of these models can increase significantly as the resolution or number of elements representing the model region increases. Recent advances in watershed modeling development and computer hardware capabilities are allowing for the production of much larger computational models which can have a great impact on the way decisions are made for managing water resources (Rouholahnejad, Abbaspour et al. 2012).

Higher resolution models require a significant amount of computational power and memory to perform the necessary computations and for subsequent viewing of the results. These large-scale models are normally run on high-performance computing (HPC) networks or supercomputers. HPC's dedicated to research are typically shared between end users that have different needs and therefore computational tasks or jobs are submitted in a queue to accomplish the tasks most efficiently (Rouholahnejad, Abbaspour et al. 2012). In cases such as this, it may be feasible to run a large model on an HPC but viewing the results may not be feasible if

computational time is restricted. Access to HPC networks is also restricted so as not to expose the resources to non-intended end-users (Blanc and Lalande 2013).

There are many potential end users that could benefit from facilitating the sharing of high-resolution simulation results to provide an efficient and effective viewer experience. Water resource managers working with high-resolution models need ways to share their results with clients without investing much time and effort into advanced concepts of map servers. Also, being able to readily share model results in a more accessible way is desirable when access is restricted on most HPC environments.

1.1 Background

Visual exploration of geospatial data information is an important component to model post-processing and interpretation. It is desirable for the visualization interpreters to be able to view high quality large-scale raster geospatial data in a web environment so as to facilitate broad accessibility to model results. There are challenging technical issues as a result of large data transfer between servers and client applications that need addressed for improved performance. If data transfers are not managed efficiently, significant lag or overloading systems can occur (Zhang 2010).

Tiled map techniques are used to ease the query performance strain between server and client applications as users evaluate regions of interest. The Open Geospatial Consortium (OGC) is an international industry consortium that maintains encoding standards for sharing and requesting geographic information between diverse data stores (Open Geospatial Consortium 2015). One of the more popular protocols used from the many maintained is the Web Map Service (WMS) protocol. The WMS protocol is the standard protocol for serving georeferenced

map images over the internet that have been generated by a map server (Michaelis and Ames 2008). Another standard protocol is the Web Map Tile Service (WMTS) which performs the same function as the WMS mentioned previously but takes into tiles for the layers for more efficient transfers. Companies and government agencies such as Google Maps and Earth, Microsoft Bing Map, ArcGIS, the National Aeronautics and Space Administration (NASA) and the National Oceanic and Atmospheric Administration (NOAA) take advantage of OGC standards for map tiling techniques to conveniently serve static tiles of Terabytes of imagery (Zhang and You 2010).

When working with data sets from transient models, the data is typically processed interactively based on user preferences. There are two main approaches for processing the visualization interactively over the internet: client-side and server-side. Client-side means that the machine of the end-user does the majority of the processing for visualization and server-side means that the server handles most of the processing. There are a variety of tools that are capable of processing visualization interactively. If rendered server-side, the layers generated can be served using standard OGC protocols to the client-side application (Huang 2003).

To avoid heavy loads on the server- or client-side machines due to dense data sets, rendering of model data can also be handled before it is hosted on the server. Transient model visualization can be converted into video tiles on the platform where performance will be most effective prior to serving the data sets (Feng, Croft et al. 2011). This method sacrifices interactive viewing options and memory storage for more efficient data transfer that does not suffer from the potential lag of server- or client-side processing.

1.2 Tools for Viewing Model Results over the Internet

Many options are available for visualizing transient model output over the internet via an efficient and effective viewing experience. Each method has a unique set of benefits and drawbacks. A selected number of these methods will be briefly discussed in the following sections.

1.2.1 Google Earth

Researchers working on projects that do not require ultra-high spatial resolution can often use the capabilities that Google Earth and Google Maps offer to visualize the data in a GIS environment. Both tools handle raster animation of transient data through the support of the Keyhole Markup Language (KML). Users can install Google Earth or access the Google Maps API for free to view the data locally by passing copies of the KML layers of interest to users and overlaying vector data for further interpretation (Compieta, Di Martino et al. 2007). For large regional-scale data with high resolution, Google Earth and Google Maps are not equipped to handle the memory constraints that are inherent in the KML workflow. The size and complexity restrictions for KMLs in Google Maps include maximum fetched file sizes of 3 megabyte (MB) and maximum uncompressed KML file size of 10 MB (Google).

1.2.2 Google Maps Engine

Google also provides a cloud service called Google Maps Engine to serve data on interactive web maps. The benefits to using Google Maps Engine is that users can upload customized imagery for map hosting. If raster imagery is uploaded, the layers are tiled and converted into online map compatible layers by Google Maps Engine. This makes it easy to generate map layers that can be published and shared online. The drawbacks to using Google

Maps Engine include price as Google charges for the space used to host layers which can be substantial in size (Blower 2010). Google Maps Engine was not capable of producing transient animated maps without the use of KML layers when this research was conducted. It is also noted that Google Maps Engine and its API will be permanently deprecated in January 2016 (Google).

1.2.3 GeoServer

GeoServer is an open-source java-based server software that allows the sharing and editing of geospatial data. It follows the standards of OGC and can be implemented to deploy WMS layers and create map layers using a variety of formats. Implementation consists of setting up a host server and deploying the layers to local end-users through internet browsers running mapping libraries. OpenLayers is one of the more popular mapping libraries that GeoServer users implement since it is integrated into the service. The benefits of using GeoServer are that it is a free software and has a large following of users and developers that contribute to its growth. Users can deploy both raster and vector data layers with interactive selection for visualization. One drawback is that there can be significant lag when serving a massive amount of data that is not pre-rendered or cached server-side. It can put major strain depending on the size and number of time steps being served, server hardware capabilities, and the number of users accessing the server (Neelakantan, Mueller et al.).

1.2.4 ncWMS

Another method that is similar to the GeoServer is ncWMS. ncWMS is a java-based specification for multi-dimensional gridded environmental data that can read in a larger number of common scientific data formats including NetCDF and generate map layer imagery in various coordinate reference systems. The WMS specification from OGC is used to call the data and it

has the capability of using animation as well. It is developed and maintained by the Reading e-Science Centre at the University of Reading, UK. The convention relies heavily on the Java NetCDF from Unidata. Currently, there is no single well-defined way to serve the gridded data through a WMS interface but the workflow of the ncWMS specification can be seen below in Figure 1-1 (Blower 2010).

1.2.5 Time Machine

A team of researchers at the CREATE Lab in Carnegie Mellon University working in collaboration with a company named GigaPan Systems developed a tool called Time Machine for viewing of images from a gigapixel camera that was used to capture high resolution detail over time. Time Machines are interactive web browser viewers for video tiles of separate zoom levels and areas of focus. The video tiles are generated from high-definition imagery being cut into tiles of manageable resolutions then stacked and converted into video based on the same geospatial location. The video tiles are stored server-side in a folder that contains all the necessary code to view and serve the results. The result of this system is an efficient and effective viewer experience of a potentially enormous set of rendered data. Time Machine Creator is a software that was generated to facilitate the generation of Time Machines. It can be used through a graphical user interface or from the command line using installed Ruby scripts. (Sargent, Bartley et al. 2010).

Time Machine was initially developed for the purpose of viewing real-time high definition photography in a shareable format. After it was developed, a research team interested in cosmological models with rendering definition on the terapixel-scale, frames composed of one trillion pixels each, used supercomputers to render and process the Time Machine with numerous

time steps. The results suggest very responsive viewer experience that maintain the highest definition (Feng, Croft et al. 2011).

The benefit of video-based tiling is that the user can take advantage of the HPC used to produce the model results and run this post-processing tool so that their results are more readily available. Another benefit is users will not have to invest time and money into server systems to render such large results as the product is easily shareable once placed on an accessible server.

1.2.6 Tethys Platform

In related efforts, a team of researchers from universities in Utah and Wyoming under the group name CI-WATER focused on the development of HPC- and web-based resources for improving water management in the Intermountain West. One product developed from the CI-WATER initiative is a free and open-source server-side application called Tethys Platform. Tethys was designed for hosting water resource web applications for use by engineers, decision-makers, and water resource managers (Swain, Christensen et al. 2015).

Tethys is built on software that gives it capabilities for visualization, geoprocessing, and distributed computing as shown in Figure 1-1. The Tethys visualization tools include a built-in component of GeoServer and OpenLayers map libraries for GIS. It provides a simplified web-interface for developed applications that have controlled user access for specifying which end-users are able to log in and view app-specific content (Jones, Nelson et al. 2014).

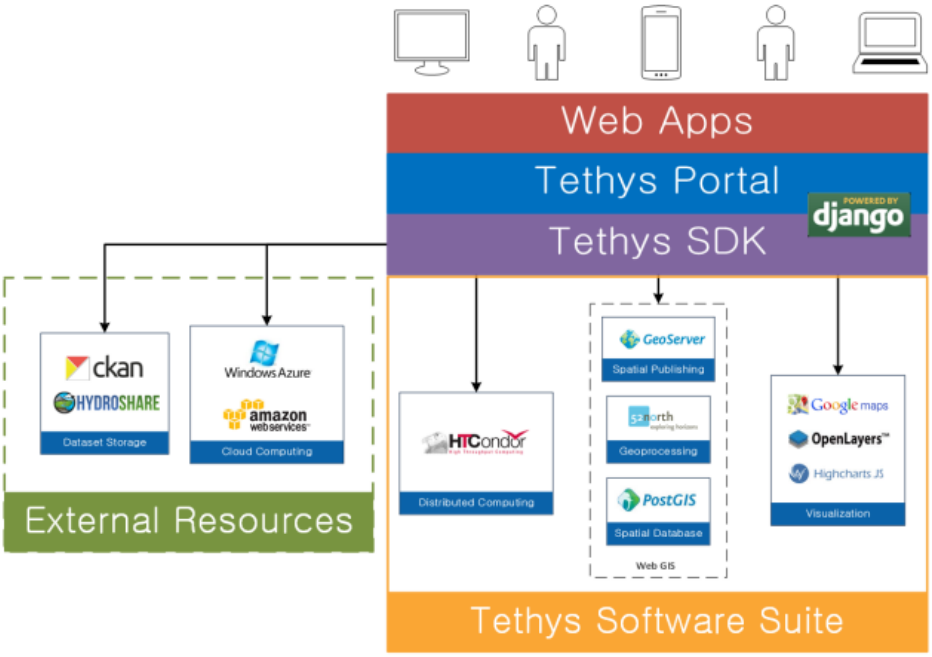

Figure 1-1 Tethys Platform Workflow (Jones, Nelson et al. 2014).

1.3 ADHydro Hydrologic Model

As mentioned previously, the enhancements in computer systems are allowing for larger models that are more complex using more data to represent larger areas. Though smaller models for rapid modeling and analysis are needed to address emergency situations and is useful for teaching and applying hydrologic principles, larger-scale models are becoming more prevalent among professionals. Along with the increase in model sizes arose the need for more efficient model development practices (Rouholahnejad, Abbaspour et al. 2012).

A new hydrologic model for improving water management in the Intermountain West entitled ADHydro was also developed by the CI-WATER team at the University of Wyoming. It is a physics-based finite-volume watershed model that is designed for water managers to simulate rainfall, infiltration, snowfall and snowmelt. The model is designed to run on HPC

systems to take advantage of highly parallel computations. The ADHydro model domain sizes are also designed to be able to handle massive regions with high definition.

A representable example of a very large model being built using ADHydro is the Green River basin. The entire Upper Colorado River, shown below in Figure 1-2, is included in the model domain which has a basin area of approximately 288,000 square kilometers and a population of more than 30 million depending on the water source. The input parameters implemented in the model are based on the water management rules for major reservoirs and irrigation districts in the region as monitored by the Bureau of Reclamation (Steinke, Ogden et al. 2014).

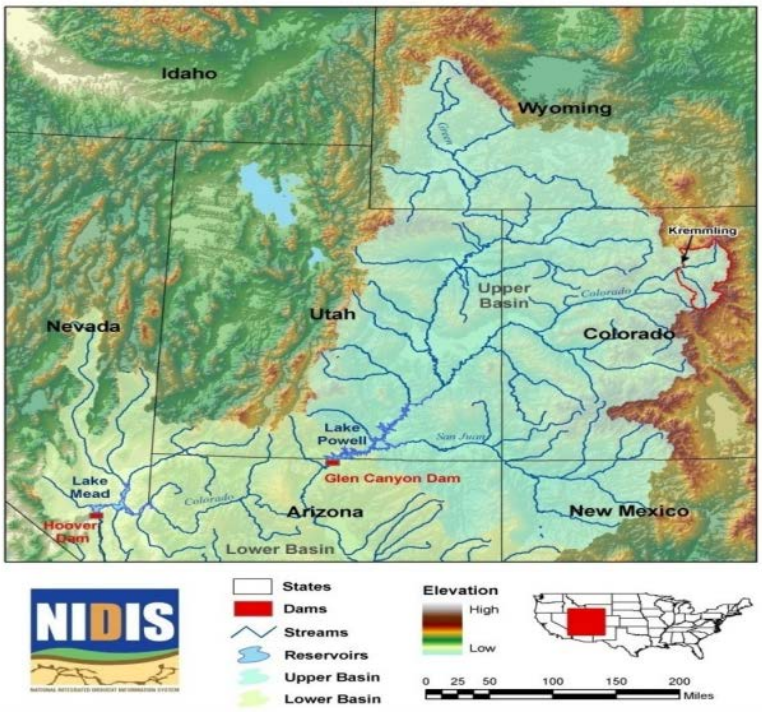

Figure 1-2 The Upper Colorado River Basin to be Modeled (Steinke, Ogden et al. 2014).

1.4 Objective

In this research, I propose a method of facilitating the accessibility of high-resolution model visualization by processing the model output using the same computational resources on the HPC that are used to run the model. The processing includes generating imagery of each time step output of a specified parameter. To make the processed output even more manageable, the imagery is then tiled or cut into smaller resolutions so that they can be managed according to standard conventions for sending and receiving geospatial data over the internet. To produce a more fluent animation between time steps, the tiles are stacked and converted into video tiles. The product can be hosted on a third-party server so as not to disrupt the resources on the HPC. The results allow water managers and the community at large to have more accessibility to high-resolution model results for better decision-making.

More specifically, the objective of this research is to develop a method for efficiently sharing potentially massive hydrologic model output visualization being developed on secure HPC systems to a nontechnical audience. To accomplish this objective, the following tasks were undertaken:

1. Evaluate Time Machine as a visualization tool for massive hydrologic models.
2. Develop automated workflows using Python.
3. Validate automated code with ADHydro case study.
4. Discover limitations of process including feasibility.
5. Explore the possibility of using Tethys as a tool combined with Time Machine.
6. Recommend setup and use for system implementation.

Time Machine was chosen as the primary tool for facilitating the distribution of large watershed models over the previously mentioned systems due to its demonstrated capacity of

massive data distribution. ADHydro is the hydrologic model being used as researchers at the University of Wyoming are working in collaboration with this research under the CI-WATER project. Tethys is used as a tool to deploy the product to a non-technical audience.

In the following sections, I will describe the workflow of the automated Time Machine code that was developed in this research. It is a combination of Ruby, CSS, JQuery, and JavaScript, wrapped in Python. The resulting code is called Time Machine Automated Python Scripts or TMAPS. After describing the workflow, I will present a case study using ADHydro on a UNIX supercomputer environment and using Amazon Web Services. I will also discuss usefulness of Tethys in conjunction with TMAPS. Finally, I will summarize the process, outline the benefits and limitations, and propose areas of further research and development.

2 METHODS

TMAPS was designed so that one can take the output of a hydrologic model and seamlessly generate an interactive Time Machine web-viewer that can be used to share output parameter results for each time step of potentially massive high-resolution simulations. Currently, the ADHydro model output is supported in TMAPS, but it could be applied to other models with some adjustments. To arrive at the Time Machine viewer from the binary output of the model there are a number of processes that need to take place. I will briefly explain these processes in the next section and give more in-depth descriptions of the code in later sections.

2.1 Overview of TMAPS Design

The TMAPS workflow is illustrated in Figure 2-1 showing how it is intended to be used on an HPC environments. TMAPS includes three major workflow processes including rendering output, frame processing, and generating the Time Machine.

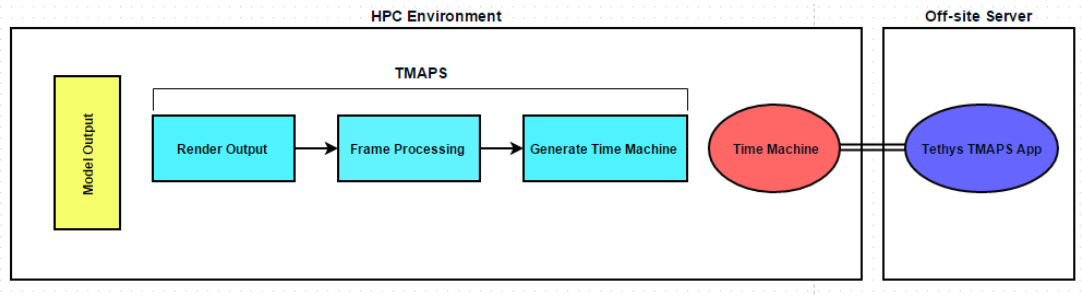

Figure 2-1 Overview of the TMAPS Workflow.

2.1.1 Render Output

The first step involves rendering the output from ADHydro. The solution parameters that ADHydro outputs to an unstructured grid include surface water depth, groundwater head, and snowmelt. The solution can have numerous time step outputs for each parameter and is exported into a binary format called NetCDF. Although binary is a more efficient file storage format than the standard ASCII format, the size of the files can still be hundreds of gigabytes for larger models and thus too large to efficiently move to end-user machines.

Using an open-source software package developed by Kitware called ParaView, the binary data is transformed into high-definition raster data. ParaView is a graphical user interface that is built on top of the visualization toolkit (VTK) for 3D computer graphics, image processing and visualization. ParaView has the ability to provide visualization by accessing computational components of multiple machines as needed by larger data renderings. It also has a Python application programming interface (API) where renderings can be performed and exported programmatically for each time step associated with a data set (Quammen 2015).

2.1.2 Frame Processing

Once each time step is rendered into an image using the ParaView API, the images are then processed as noted in the Frame Processing stage, shown in Figure 2-1. The frame processing step is necessary since there are limited image output options and no basemap support in the ParaView API. Using Python modules, the images are trimmed and, if desired, the users call a basemap to be put under the mesh for geographical context.

2.1.3 Generate Time Machine

After each frame is setup, the final process in the TMAPS workflow is generating a Time Machine of the processed frames. Time Machine Creator is the software that was developed by the CREATE Lab at Carnegie Mellon University to facilitate making Time Machines based on a set of frames for input. The Time Machine Creator installation comes with Ruby scripts that can be used programmatically to generate Time Machines. The Time Machine product that TMAPS generates is a folder that contains the necessary components to serve and visualize the specified ADHydro output data. Once Time Machines are generated, they can be moved to a remote server for sharing over the internet. The Tethys TMAPS application can be used to organize view generated Time Machines and limit access to specific end-users.

2.2 Software Organization

TMAPS is built using the Python 2.7 programming language. The Tethys TMAPS App is based on the Django web-development language. Both codes for TMAPS and the Tethys TMAPS App can be found in separate repositories on GitHub under the CI-WATER group. The software dependencies and structure are presented in Figure 2-2; the two products generated from the research TMAPS and the TMAPS App are outlined as well.

The TMAPS code includes separate Python modules for rendering ADHydro, frame processing and generating Time Machines in Python. I developed the code this way so that it is flexible enough for future development of additional models by allowing developers to use the methods that apply to their model output. For example, future developers may only need to develop a rendering module for their model and utilize the tools already developed in the other two modules to generate a Time Machine.

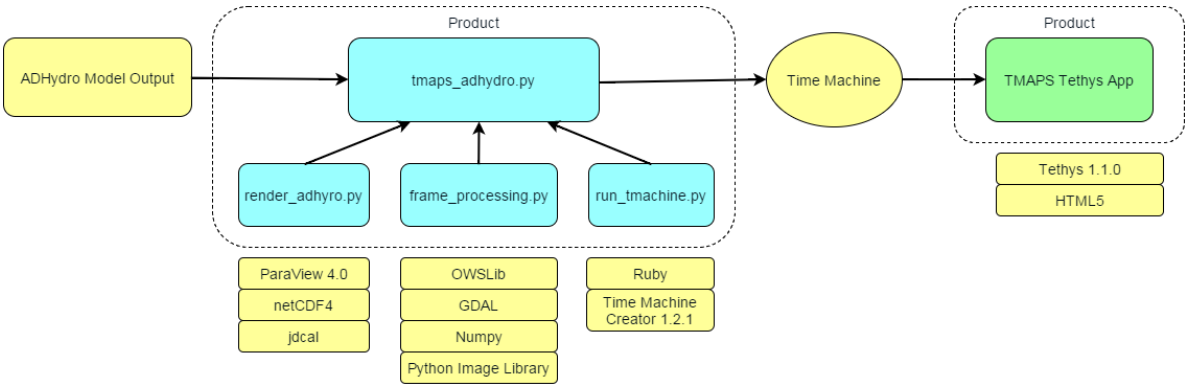

Figure 2-2 Software Organization of TMAPS and TMAPS App.

The software in the TMAPS GitHub repository contains all the necessary tools to develop a Time Machine. The three main modules developed in TMAPS are called 'render_adhydro.py', 'frame_processing.py', and 'run_tmachine.py'. The 'render_adhydro.py' contains methods specific for ADHydro that will contour and render out an image of each time step. It also contains a method to return specific information about a parameter and time step information. The 'frame_processing.py' contains general tools to process the images including trimming images, reprojecting bounds of image when working in different coordinate systems, downloading and underlaying basemaps, imposing opacity onto the model output image when being overlayed onto basemaps, The 'run_tmachine.py' contains methods for setting Time Machine Creator input, running Time Machine Creator, and updating the Time Machine viewer styles.

Included in the TMAPS GitHub repository is a script entitled 'tmaps_adhydro.py' that is already setup to use the methods developed from each TMAPS module to generate a Time Machine from ADHydro with little effort. The script is setup to use the multiprocessing Python module. The use of multiprocessing allows for methods to be distributed over shared CPUs so that processing can be more efficient. Each of these modules are presented in Figure 2-2 below.

2.3 TMAPS Implementation

Using the TMAPS code requires that users install prerequisites as defined in the Readme.md document in the TMAPS repository download. The following sections define a standard installation and implementation of TMAPS on the Debian-based Linux operating system Ubuntu.

The Tethys TMAPS App requires Tethys Platform to be installed before the app can be installed. For complete installation instructions for Tethys Platform, see http://www.tethysplatform.org/

Once Tethys Platform is installed on the server operating system, the TMAPS Tethys App can be cloned from the CI-WATER GitHub repository. The app can then be installed by running the 'setup.py' in the app download. For complete Tethys app installation instructions, see the Tethys documentation found on the Tethys website mentioned above.

2.3.1 Specify ParaView Render Mechanics

After the prerequisites are installed, the system can be setup to use CPU or GPU rendering depending on the hardware that is available for the jobs. It is preferable to use a graphics card or GPU to handle the rendering of each output as it can be many times faster. For this reason, when the instructions are followed for standard installation of ParaView, it will have settings to look for a local GPU to do any rendering.

In some HPC environments, there is not a GPU available to assign the rendering jobs. If there is not a GPU readily available, users will then need to recompile the ParaView installation to use a CPU rendering library such as classic OSMesa or Gallium Llvm. Instructions on doing this are outlined on the ParaView website (Burlen 2013).

A benchmark study using the OSMesa Gallium Llvmpipe flavor of CPU rendering was performed by ParaView developers to evaluate how many cores allowed for the most efficient CPU rendering without exhausting resources unnecessarily. Figure 2-3 is referenced from the ParaView Wiki at www.paraview.org and denotes the results of the study performed using 16 cores on a very dense data set with Message Passing Interface (MPI) enabled on the installation. The MPI component means that parallel processing is enabled and that jobs or processes can share CPUs and spawn across multiple CPUs in ranks. As shown in Figure 2-3, as more ranks are used, the less time it takes for the CPUs to render the shared task. As more than 10-12 ranks of MPI are used in the benchmark test below, the rate of render time does not decrease significantly enough to justify additional CPU usage. It is an important note that if MPI is not enabled on the ParaView compiled code then that compile is limited to one CPU per task.

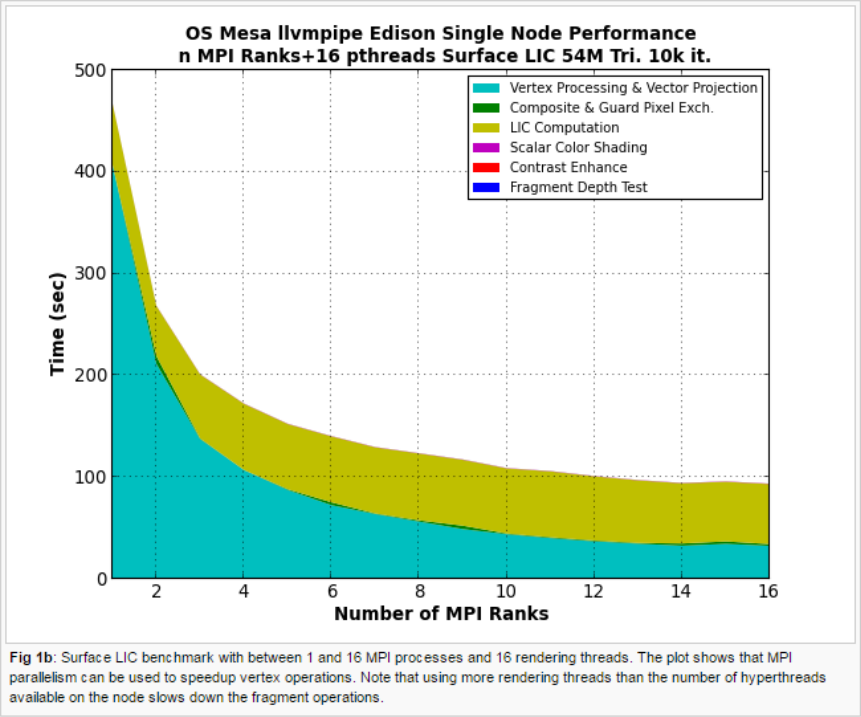

Figure 2-3 Benchmark Study by ParaView Developers of CPU Rendering (Burlen 2013).

2.3.2 ADHydro Model Output to XDMF

Before the ParaView libraries can be called to render each output, the ADHydro output needs to be interpreted from the NetCDF binaries to a format that ParaView can read. The file reader that translates the NetCDF for ParaView to interpret is called the XDMF Reader (http://www.xdmf.org/). The XDMF Reader requires XDMF files that use the '.xmf' extensions to be generated to read the NetCDF files; a brief example of the syntax that the '.xmf' file follows is presented in Figure 2-4 below. Upon request, the developers of ADHydro at the University of Wyoming have a code that can be run to generate the necessary XDMF files so that ParaView can read the binary NetCDF files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd">
<Xdmf Version="2.0" xmlns:xi="http://www.w3.org/2001/XInclude">
  <Domain>
    <Grid GridType="Collection" CollectionType="Temporal" Name="MeshState">
      <Grid GridType="Uniform">
        <Time Value="0.000000"/>
        <Topology NumberOfElements="36452" Type="Triangle">
          <DataItem ItemType="HyperSlab" Dimensions="36452 3" Type="HyperSlab">
            <DataItem Dimensions="3 3" Format="XML">0 0 0 1 1 1 1 36452 3</DataItem>
            <DataItem DataType="Int" Dimensions="1 36452 3" Format="HDF">geometry.nc:/meshElementVertices</DataItem>
          </DataItem>
        </Topology>
```

Figure 2-4 An Example of the XDMF File Syntax.

Once the '.xmf' reader file has be generated for the display.nc and state.nc files, ParaView can then read the data for rendering. A list of ADHydro output files that TMAPS needs for reference are shown in Table 2-1.

Table 2-1 List of Standard ADHydro Output Files

File	Description
mesh_state.xmf	XDMF File for parsing state results
mesh_display.xmf	XDMF File for parsing display results
parameter.nc	NetCDF File for input parameter specifics
geometry.nc	NetCDFFile for geometries of model domain
state.nc	NetCDF File for output associated to state run
display.nc	NetCDF File for output associated to display run

2.3.3 Render ADHydro Methods

To arrive at the frames needed to produce a Time Machine, the first step is to render each output that a user specifies. There is a class in the 'render_adhydro.py' called 'ADHydro_Render' which contains necessary methods to export each output in the form of a portable network graphics (PNG) image. The Python code necessary to run the rendering is shown in Figure 2-5 and requires the following input:

- adhydro_output_dir – the location of the ADHydro NetCDF and XMDF files
- user_parameter – parameter that the user would like to visualize
- user_contour – currently there are 6 options for contour schemes
- user_opacitiy – the level of opacity if using a basemap
- start_frame – the output timestep that will be the first show (zero-based counting)
- end_frame – the ending frame or the last consecutive output to be rendered

```
import render_adhydro

runtmaps = render_adhydro.Adhydro_Render(output_dir, user_parameter,user_contour,user_opacity, start_frame, end_frame)
runtmaps.renderpv()
```

Figure 2-5 Code Sample of Rendering Specified Output.

The product of this first phase is a folder called ‘images’ that contains \ each PNG image with a specified background color; the last image generated is the legend used for overlay on the Time Machine. An example of the rendered output using the ParaView tools is shown in Figure 2-6. The ParaView Python API does not currently provide a way to impose transparency using an alpha channel. Each image is exported and the next phase incorporates image processing.

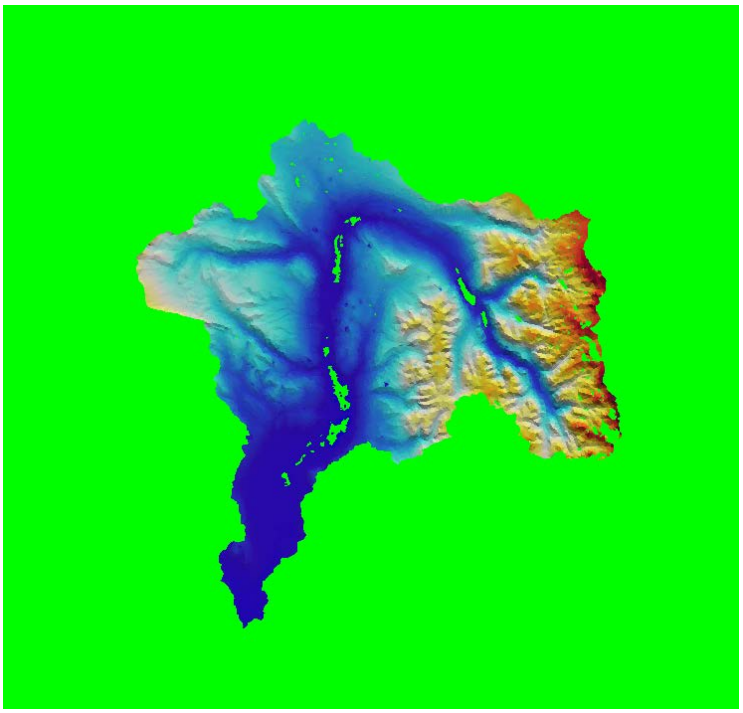

Figure 2-6 Example of Exported Output PNG using ‘render_adhydro.py’.

Another result of generating an ‘ADHydro_Render’ object is that the maximum and minimum bounds of the x and y coordinates are calculated from the ‘geometry.nc’. These are referenced to call a basemap and incorporate other view options explain in the next section. Each time step is also written out to a text file entitled ‘tstepstotext.txt’ to be used with the Time Machine viewer referencing.

2.3.4 Frame Processing Methods

The next phase in the TMAPS workflow requires the use of the methods in 'frame_processing.py'. Before the methods in 'frame_processing.py' can be called, there should be a sequential number of images outputted in the 'images' folder located in same current working directory after using the 'ADHydro_Render' class methods.

Using the trim methods defined in 'frame_processing.py', each image can be trimmed to the exact extents of the mesh; this is so that the maximum and minimum x and y coordinates of the mesh correspond to the bounds of the image. The background color is also processed according to the user's desire for a basemap. One reason that background processing is done in the 'frame_processing.py' phase and not originally in the previous rendering phase is due to the limited image export options such as transparent backgrounds in ParaView. If a user specifies a basemap by utilizing the 'getbasemap' method and the 'underlaymap' method, the background color is assigned to be fully transparent by imposing an alpha channel on its pixel color, and the mesh image is superimposed onto the basemap image through the use of the Python Image Library (PIL) combined with the capabilities of the Numpy package. When a basemap is not specified, the background color defaults to be black. An example of a sample code used to call the necessary methods including a basemap is shown in Figure 2-7 and the processed output of the combined methods is shown in Figure 2-8. Each image or frame is processed and passed into a subdirectory within the Time Machine Creator input entitled '0100-original-image'. The input parameters for the frame processing methods are described as follows:

- path – the directory that contains the rendered output from the model.
- project_proj – the projection the model was developed in using +proj4 format.
- xmin – the minimum x-coordinate of the unstructured grid.

- xmax – the maximum x-coordinate of the unstructured grid.
- ymin – the minimum y-coordinate of the unstructured grid.
- ymax – the maximum y-coordinate of the unstructured grid.
- opacity – the transparency factor for the mesh being placed on the basemap.
- fgnewsizexy – the coordinate extents of the basemap
- overlayx – the x-coordinates of where to place unstructured grid on basemap.
- overlayy – the y-coordinates of where to place unstructured grid on basemap.

```
import frame_processing

# Path to images rendered
path = "./images/"
project_proj = '+proj=sinu +datum=WGS84 +lon_0=-109 +x_0=20000000 +y_0=10000000'

# Download basemap based on specified proj4 projection and unstructured grid extents
getbasemap_return = frame_processing.getbasemap(project_proj, xmin, xmax, ymin, ymax)
fgnewsizexy = getbasemap_return[0]
overlayx = getbasemap_return[1]
overlayy = getbasemap_return[2]

# Trim and impose transparencies on the rendered imagery
combinations = [(os.path.join(path,fname), opacity) for fname in os.listdir(path)]
frame_processing.runtrim(combinations)

# fgnewsizexy, overlayx, overlayy are generated from the frame_processing.getbasemap method
# Overlay the images rendered from the model results onto basemap
underlaycombo = [(hname, fgnewsizexy, overlayx, overlayy,tmcname,path) for hname in os.listdir(path)]
frame_processing.underlaymap(underlaycombo)
```

Figure 2-7 Sample Code for using Frame Processing Methods.

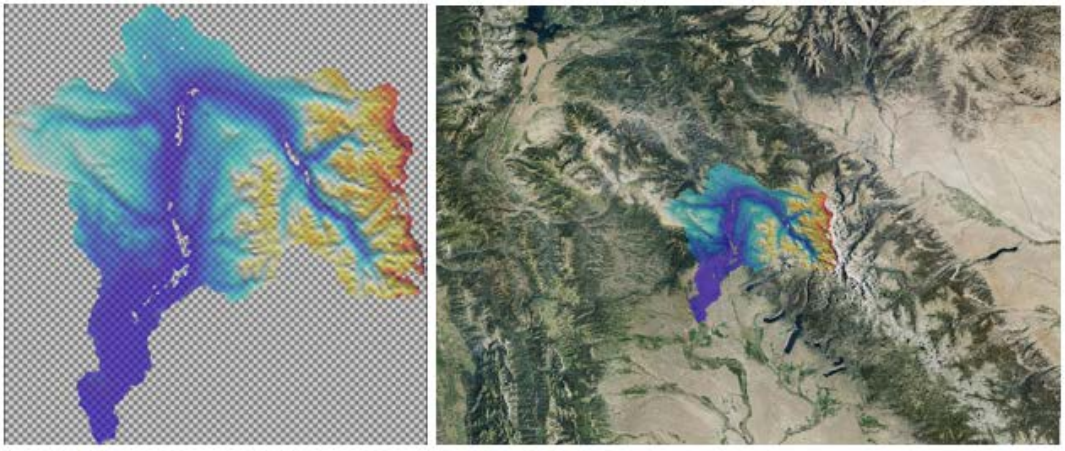

Figure 2-8 Sample Frame Output Produced using the 'frame_processing.py'.

2.3.5 Run Time Machine Methods

After each frame is deposited into the '0100-original-images' folder using the 'frame_processing.py' methods, the functions from 'run_tmachine.py' are called to setup the necessary Time Machine settings. The settings that control the Time Machine tile size, video size, frames per second rate and other settings are written out to the 'definition.tmc' and placed in the same directory as the '0100-original-images' folder.

Once both the folder that contains the frames and the 'definition.tmc' inputs are specified, the 'run_tmachine.py' methods use the Time Machine Creator Ruby script entitled 'ct.rb' to handle the Time Machine generation. This code takes each frame and cuts them into tiles based on the resolution of the image and the tile size specified. The script then utilizes a program called FFMPEG to create video of each tile stack. FFMPEG is a cross-platform solution for generating video and has a command-line interface that Time Machine Creator utilizes. After the 'ct.rb' script is finished, a new folder entitled '*user_specified_name*.timemachine' is generated. This is the Time Machine as it contains all the video tiles, JavaScript, HTML, CSS, and other needed code to view and serve the product. Before it can be viewed with the ADHydro solution results, additional updates are performed by the 'run_tmachine.py' to update the styles so that the output times associated to each frame are located on the viewer. The legend is also placed on the viewer. One final component on the viewer that is updated by the 'run_tmachine.py' is the Small Google Map widget so that the user can be given more geospatial context as to where the project is located. An example of the method called to initialize the Time Machine Creator methods is presented in Figure 2-9, the input includes the following:

- cores – the number of cores to be used in parallel.
- tmcname – the name of the Time Machine to be generated.

- tmc_ct_dir – directory where the Time Machine Creator scripts can be located.
- mapxmin – the frame extent for the minimum x-coordinate.
- mapxmax – the frame extent for the maximum x-coordinate.
- mapymin – the frame extent for the minimum y-coordinate.
- mapymax – the frame extent for the maximum y-coordinate.

```
import run_tmachine

run_tmachine.runtm(cores, tmcname, tmc_ct_dir, mapxmin, mapxmax, mapymin, mapymax)
```

Figure 2-9 Sample Code to Initialize Time Machine Creator from the 'runtmachine.py'.

After the ‘run_tmachine.py’ method have been executed, the final product can be viewed by opening the ‘view.html’ in a web browser. The file is located in the folder of the user-specified project name followed by ‘.timemachine’. When migrating the Time Machine to other locations, users will need to move the entire folder of ‘*.timemachine’ as it contains all the necessary code to view and serve the product.

2.3.6 Running the TMAPS ADHydro Code

To make things simpler, users can follow the example of incorporating each of the methods necessary to generate a Time Machine using the ‘tmaps_adhydro.py’. The necessary inputs can be modified and the script can be ran in a Python interpreter that has the prerequisites installed.

2.4 TMAPS Tethys App Setup

Once the Tethys TMAPS App is installed, administrators can transfer Time Machines to the directory that is located in the './public/time_machines/' – relative to where the app is installed. For a Time Machine to be served correctly through the TMAPS App interface, it will need to include the '*.timemachine' folder produced from a successful TMAPS run.The folder should include a 'tmaps_app_info.txt' metadata file that is generated by the TMAPS code; this is a unique file for each Time Machine that the Tethys TMAPS App needs to list each Time Machine.

2.5 Migration of Time Machine to TMAPS Tethys App

Generating many Time Machines is best handled when users categorize and manage the output. A Tethys application was developed called the TMAPS App that will allow users to store Time Machines and view them interactively through a simple web interface. The Tethys application can also allow administrators to limit access to project results to a specified audience since Tethys inherently uses a login capability and group-access protocols.

For users to incorporate their Time Machines into the app, they need to place the generated Time Machine in a directory specified in the app instructions. The standard method for getting them from the HPC to the host server is to use a Secure Shell (SSH) protocol. Once the user has a SSH connection, they can use a Secure Copy protocol (SCP) to copy the files into the specified directory. This process can be scripted or automated if the location where the Time Machine is being generated is secure enough to store the host server credentials.

2.6 TMAPS Tethys App

When the TMAPS code is ran successfully, there is a file generated with each Time Machine that contains unique metadata about the Time Machine. The TMAPS Tethys application looks for this file in each of the migrated Time Machines. The application loops through all the Time Machines placed in the directory and lists the metadata for each in a dropdown menu for the user to select which Time Machine they would like to view as shown in Figure 2-10. Upon selection, the Time Machine is shown and the user can interactively view the product.

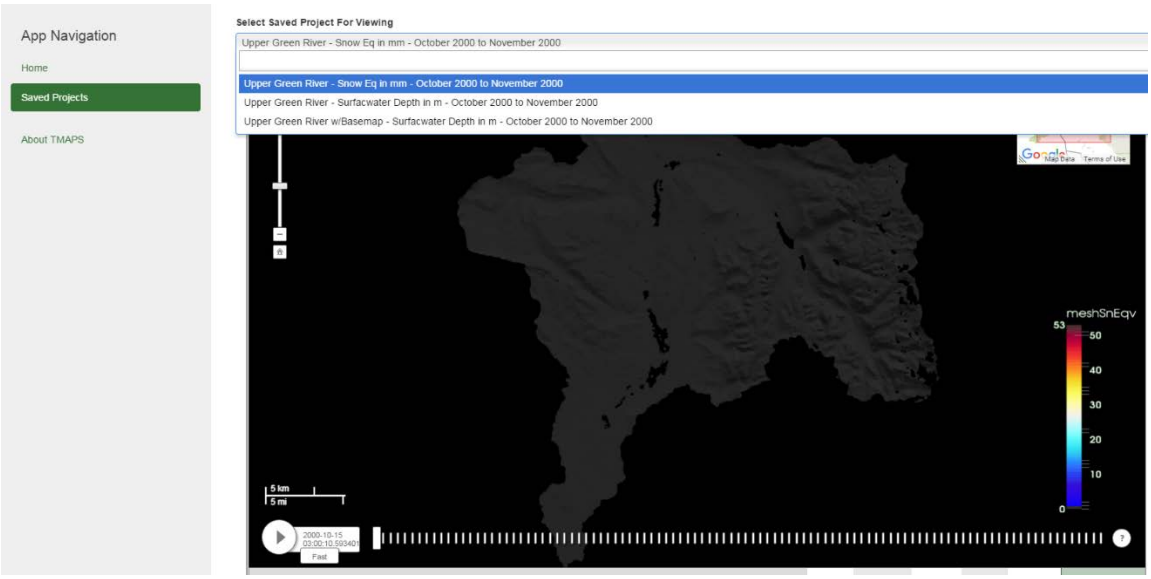

Figure 2-10 TMAPS App View for Selecting a Time Machine.

Furthermore, administrators of the installed app can go into the Site Admin and modify the groups and users that belong to the groups to specify if they want restricted access to what content is available as shown in Figure 2-11. This is beneficial to protect sensitive projects.

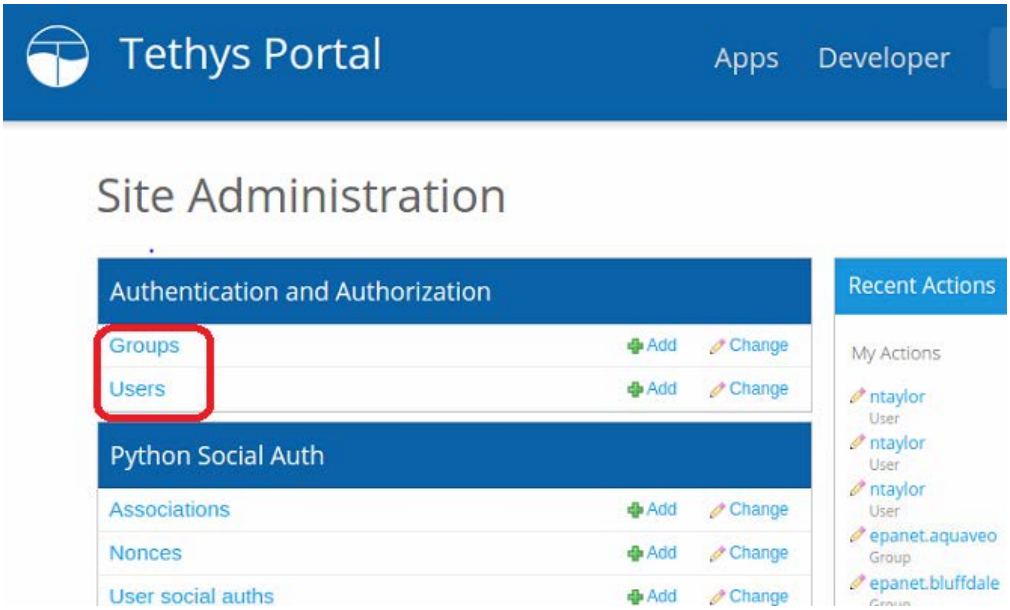

Figure 2-11 Site Admin Settings that Allow Restricted Access.

2.7 ADHydro Test Case

A representative case study was used to evaluate how the TMAPS code would be used on multiple environments where different hardware is available. The case study uses results of a large ADHydro model covering the Green River area of the Upper Colorado River Basin.

2.7.1 Introduction to ADHydro Test Case

The CI-Water research team at the University of Wyoming has a working model for a portion of the Upper Colorado River Basin. The model domain is relatively small with 36,452 elements since it is a model the developers are using to validate the ADHydro code. ADHydro models will be much larger than this as more models are built. The unstructured grid elements and different watershed capture zones are presented in Figure 2-12.

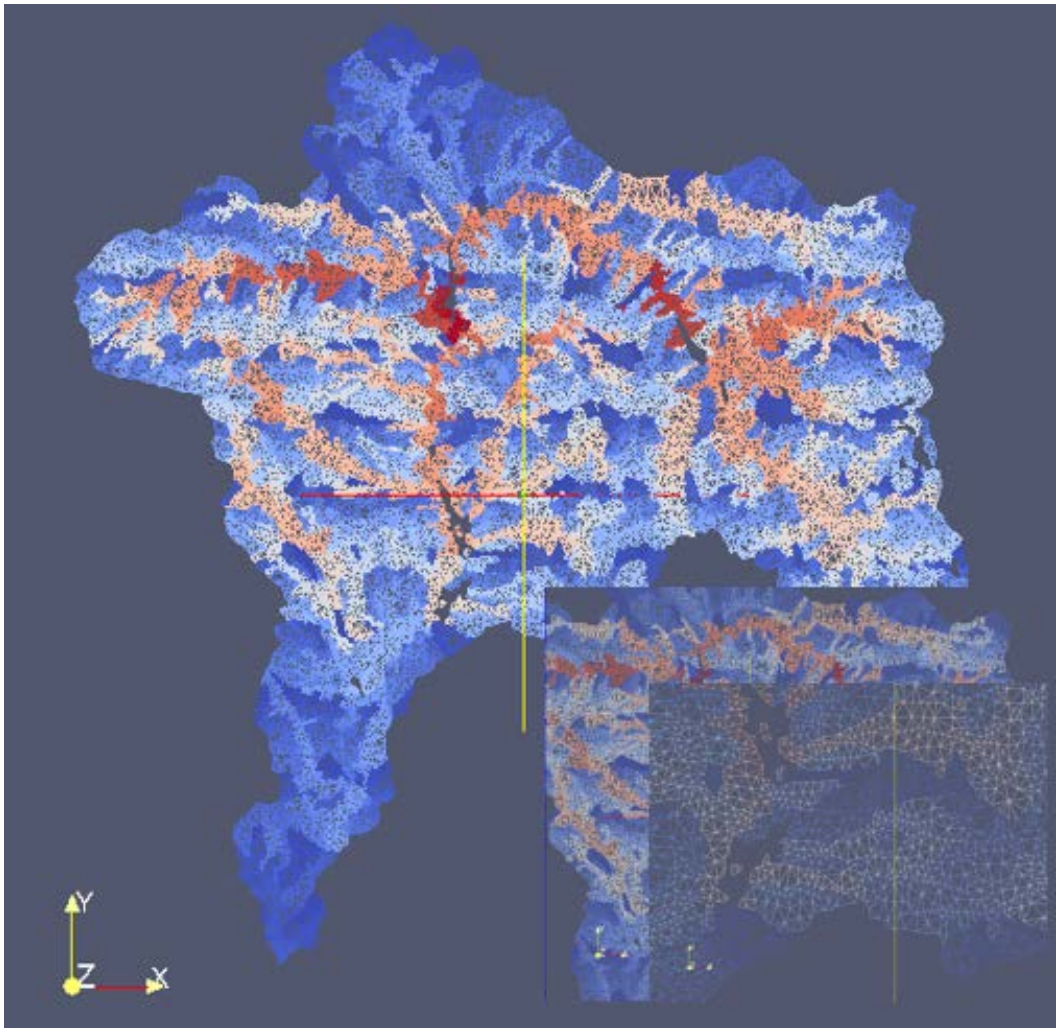

Figure 2-12 ADHydro Element Sizes of Upper Colorado River Basin.

The inputs to the ADHydro model are based off assumptions of water management rules for major reservoirs and irrigation districts in the region as monitored by the Bureau of Reclamation. Some of the more influential input parameters are presented in Figure 2-13. Additional parameters include precipitation, evaporation, and transpiration.

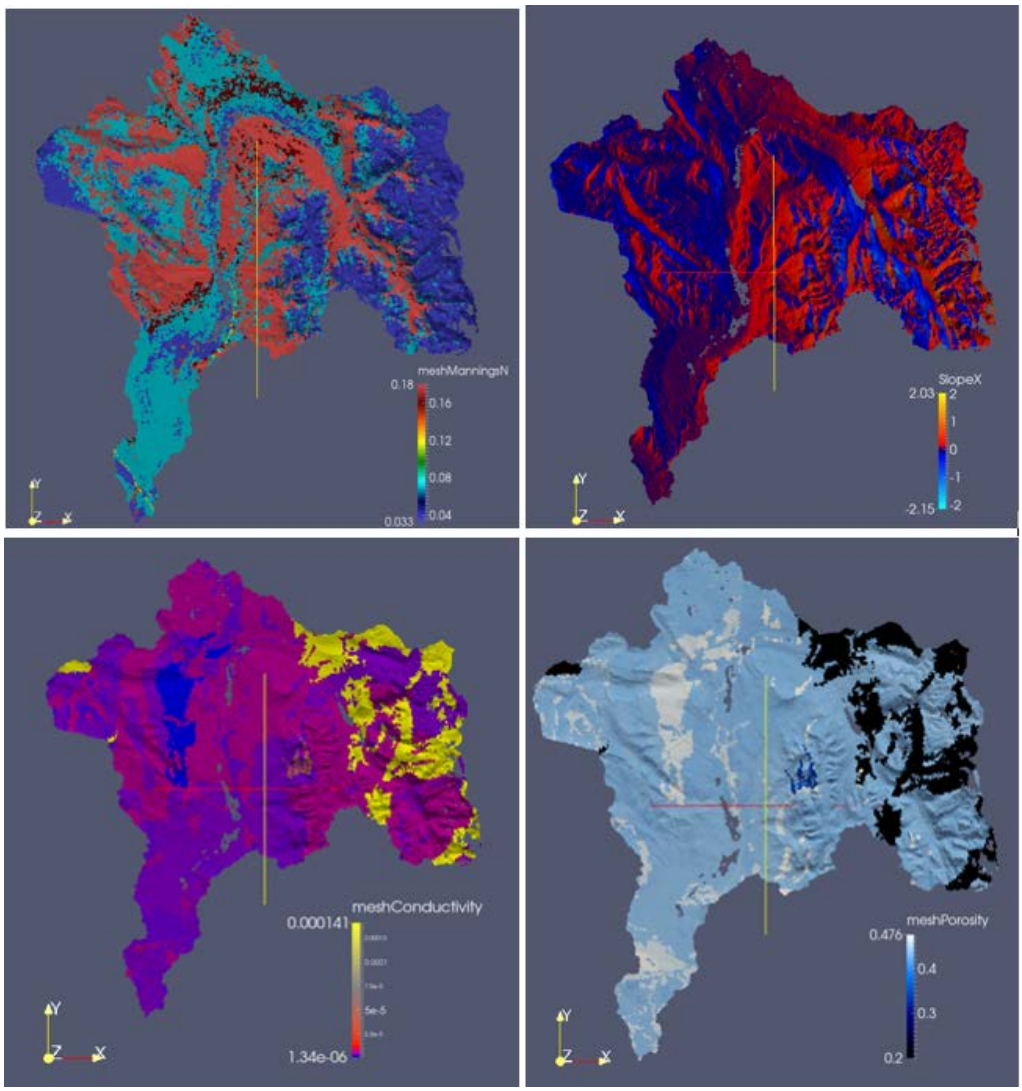

Figure 2-13 Input Parameters used for Upper Green River Model.

The Time Machine images generated by TMAPS code for the case study share the same unstructured grid as shown in Figure 2-12, and the symbology is adjusted appropiately for selected output parameters as specified by the end-user. The solution file generated is approximately 200GB in size and contains over 100 time steps of output data for each parameter. The output parameters evaluated in the test cases include groundwater head, surfacewater depth, and snow depth.

2.7.2 Test Case Analysis

There are two different environments that were used to evaluate TMAPS – the first is a supercomputer called Mount Moran. It is located at the University of Wyoming maintained by the Advanced Research Computing Center (ARCC) and is the location where the Upper Colorado River Basin model results are being generated. The other environment is a cluster that uses the Amazon Elastic Compute Cloud (EC2).

The cases evaluate different rendering and processing times based on hardware availability. File sizes and flexibility for deploying TMAPS is also evaluated. The resulting Time Machines for both scenarios are sent to the TMAPS Tethys App being hosted on a demo server located at Brigham Young University where the results are viewed.

2.7.3 Mount Moran: Test Case 1

For users to access HPC environments that are nonlocal to the end-user, it is necessary for the users to use the SSH protocol to submit commands and work within the HPC environment. Due to the security requirements of the Mount Moran supercomputer, it is necessary to first apply for an account on the HPC. Once validated, users need to generate a Virtual Private Network that securely connects the remote computer to the HPC. For this case study on Mount Moran, the Juniper Network Connect (JNC) was used to securely tunnel to the Mount Moran before the remote user can SSH through their terminal.

Mount Moran has limited access to GPU processors that would allow for accelerated rendering. To compile the CPU version of ParaView it requires users to use the CMake compiler and build from source. In this case, the ParaView version compiled includes the OSMesa Gallium Llvmpipe, a flavor of OSMesa CPU rendering library that has shown improved CPU

rendering times compared to OSMesa Classic. Unfortunately, the version can only take advantage of one core due to complications enabling the MPI component upon the build; thus, workarounds are explored as noted in the following sections.

Mount Moran is a shared supercomputer environment that uses job queueing to handle job submittals. Mount Moran uses an open-source workload manager called Simple Linux Utility for Resource Management (Slurm). Slurm requires users to setup environments that contain installed software needed to run their code. Once the end-user has installed the dependencies of their code, users generate Slurm scripts that specify the amount of cores that will be used on the run alongside a time out limit that will automatically kill the job if the time needed to finish the script is exceeded (http://slurm.schedmd.com). The results of the Slurm run are returned from where the Slurm job script was submitted with the logistics file of the run included.

Using surface water depth as the output parameter of interest, TMAPS was broken down in three different phases with 8, 12, and 16 CPU's to evaluate rendering and processing times. The first phase is rendering and 100 time steps were rendered and exported to PNGs using the ParaView OsMesa Llvmpipe option. The second phase used the frame processing components to setup the frames for Time Machine input. Both options of including and not including a basemap were ran on the previous 100 frames exported. The final phase was running the Time Machine setup methods and running Time Machine with the different number of cores to evaluate processing times.

2.7.4 Amazon Web Services: Test Case 2

Amazon Web Services offers a commercial cloud service called Amazon Elastic Compute Cloud (Amazon EC2) that is designed to be a scalable HPC for cloud-computing. After setting

up an account with Amazon Web Services, users can spin up instances of nodes with specified operating systems and hardware options as needed for their jobs.

For this case study, a Ubuntu 14.04 operating system combined with a GPU node instance was selected. A node image with the prerequisites to run TMAPS was generated following the installation instructions of the Readme.md from the TMAPS GitHub repository was saved to the AWS Management Console where it could be deployed to an instance when needed. Additional steps were performed for connecting the GPU to the rendering port of the ParaView installation (ParaViewWeb (2013)).

The same ADHydro results used in the first test case were used to evaluate the same output parameter of surface water depth. The same amount of 8, 12, and 16 cores were used on the three different phases of rendering, frame processing, and generating the Time Machine for 100 time step outputs.

3 RESULTS

The results of Test Case 1 and Test Case 2 involving the Green River area of the Upper Colorado River Basin are presented in the following sections.

3.1 Results of Test Case 1: Mount Moran

The first phase of rendering the surface water depth for the first 100 time steps of the Green River model took an average of 27.231 seconds per time step. Due to the ParaView component not being enabled with MPI, only one CPU is accessed for rendering with Llvmpipe.

The results of the second phase of frame processing the previously rendered frames are presented for average individual processing times with basemaps and not using basemaps in Table 3-1

Table 3-1 Frame Process Times Performed on Mount Moran

Mount Moran # of CPUs	Frame Process Time (seconds)	
	Without Basemap	With Basemap
8	1.817	1.995
12	1.118	1.418
16	0.910	1.114

The third and final phase results of generating video tiles and processing the Time Machine for 100 processed frames is presented in Table 3-2 below.

Table 3-2 Time Machines Process Time Performed on Mount Moran

Mount Moran # of CPUs	Time Machine Process Time (minutes)	
	Without Basemap	With Basemap
8	8.387	12.979
12	6.046	9.630
16	4.703	7.871

The resulting Time Machine sizes were 1.3GB without using a basemap and 2.5GB when a basemap was used. The Time Machine settings were 3 frames per second and set on tile sizes of 512 pixels.

3.2 Results of Test Case 2: Amazon EC2

The first phase of rendering 100 time steps using ParaView compiled to run with a GPU took on average 0.603 seconds per time step. The second phase of frame processing the previously rendered 100 frames are presented for average individual processing times with basemaps and not using basemaps in Table 3-3.

Table 3-3 Frame Process Times Performed on Amazon EC2

Amazon EC2 # of CPUs	Frame Process Time (seconds)	
	Without Basemap	With Basemap
8	2.021	2.258
12	1.327	1.631
16	1.063	1.129

The final phase results of processing the Time Machine for 100 processed frames is presented in Table 3-4 below.

Table 3-4 Time Machines Process Time Performed on Amazon EC2

Amazon EC2 # of CPUs	Time Machine Process Time (minutes)	
	Without Basemap	With Basemap
8	8.956	14.778
12	6.295	10.141
16	4.889	8.284

The size of the finished Time Machines are 1.3GB without a basemap and 2.5GB with a basemap. The Time Machine settings were 3 frames per second and set on tile sizes of 512 pixels.

3.3 Overview of Results

The rendering results on average were significantly faster on Amazon EC2 than the single core Mount Moran CPU render. On average, the GPU rendering time was 47 times faster. This is to be expected since GPUs are designed to specially handle dedicated rendering.

The frame processing results of both case studies with and without using basemaps as shown in Figure 3-1 are well correlated. The most efficient frame processing time was using16 cores without using a basemap on Mount Moran. The least efficient was processing with a basemap on Amazon EC2. The frame processing times show that there is much more efficiency achieved when jumping from 8 cores to 12 cores than from 12 cores to 16 cores. This suggests that with more computational distribution there is a point where no more efficiency will be

achieved. This point is relative to the number of frames being processed – these cases are based on 100 frames.

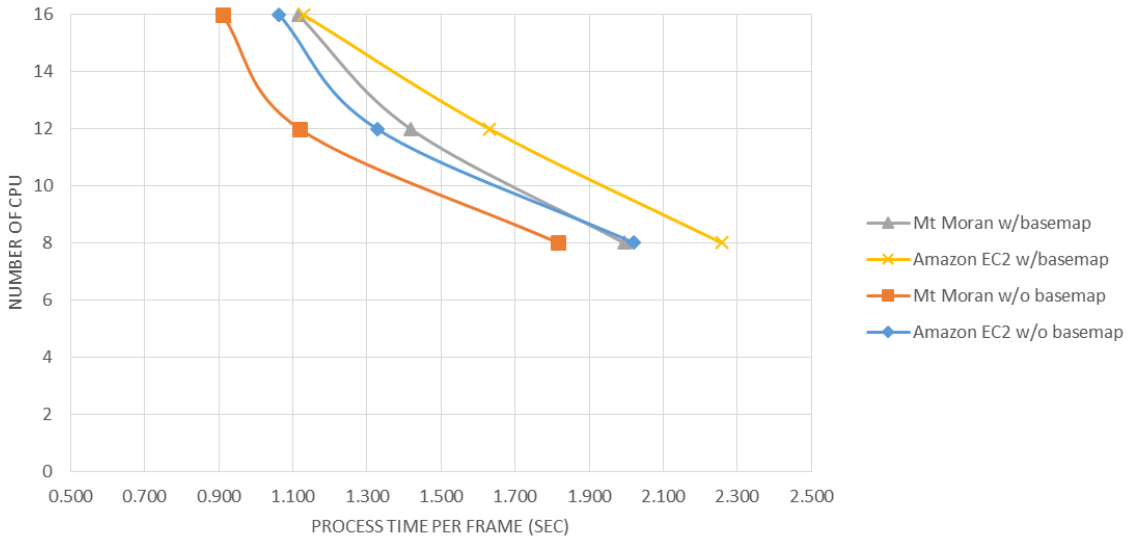

Figure 3-1 Case Study Frame Processing Times Compared

The Time Machine processing times are summarized in Figure 3-4. The Time Machines with basemaps take a little more time to process than the Time Machines that just use the model domain. This is likely due the heavier frames that are processed by the Time Machines with basemaps. The Time Machines that process the fastest are the ones without a basemap and run at the highest core number allotted. The same trend in lower efficiencies as computational distribution is increased can be interpreted in Figure 3-4.

The size of the Time Machines produced in both case studies without basemaps are 1.3 GB while those with basemaps are almost double the size at 2.5 GB. The higher definition the frames the more video tiles will be produced and the Time Machine size will increase as a result. Also, the higher number of frames being exported will have a significant effect on the size of the Time Machine.

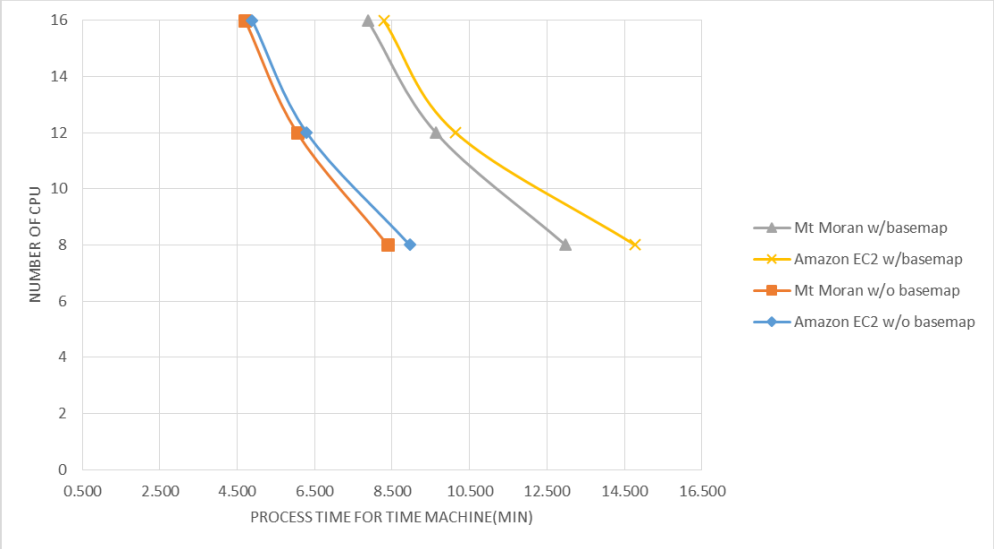

Figure 3-2 Case study Time Machine Processing Times Compared

The resulting Time Machine of surface water depth was exported into the Tethys TMAPS app that is located on a third party server. The Time Machine results of without a basemap and with a basemap were evaluated with respect to responsiveness and quality; both of the samples were found to be highly responsive and easy to navigate. Figures 3-3 and 3-4 below were the samples evaluated.

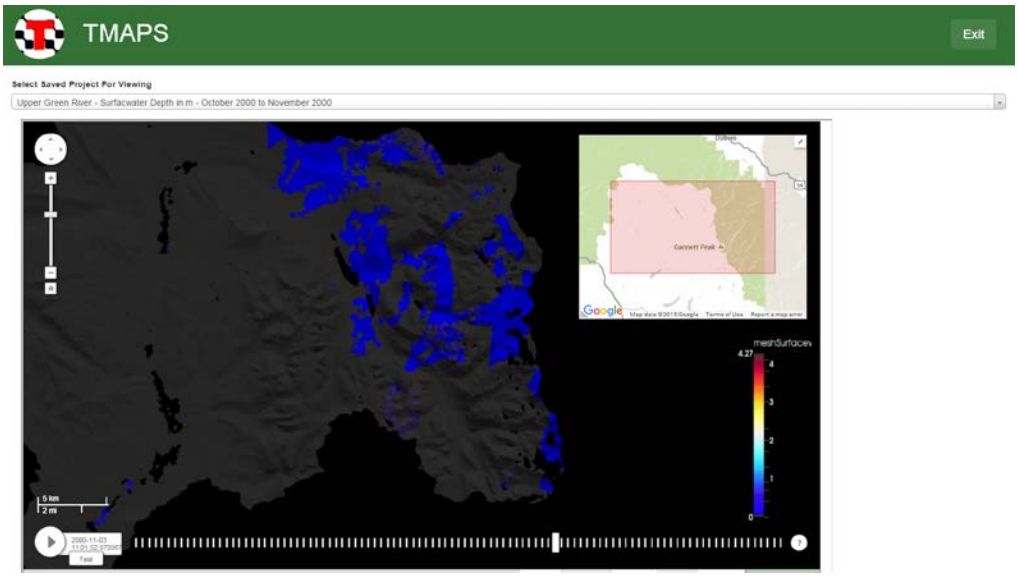

Figure 3-3 Time Machine of Surface Water Depth without a Basemap.

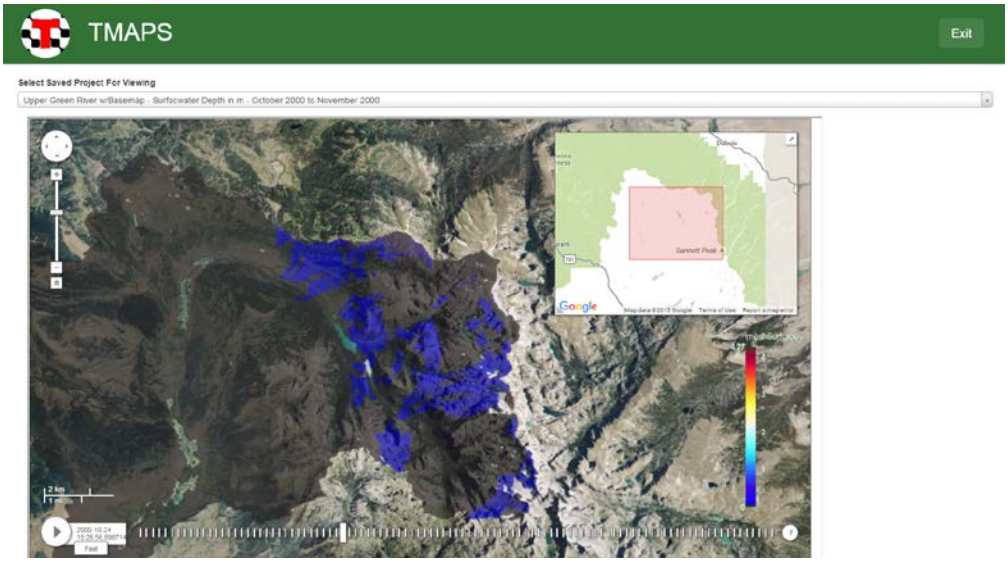

Figure 3-4 Time Machine of Surface Water Depth with a Basemap.

4 CONCLUSIONS

TMAPS is suite of Python modules that provide modelers with an option for generating web-based visualization of ultra-high-resolution simulation results for a nontechnical audience. Currently, it only supports ADHydro output. The Tethys TMAPS app provides support for the Time Machines generated from TMAPS by organizing them on a single webpage and controlling specified-user access.

4.1 Research Objectives

The conclusions of this study are presented in terms of the research objectives that were outlined in the introduction.

4.1.1 Explore Time Machine as a Visualization Tool

The first objective was to evaluate Time Machine to see if it could be used on model output to generate interactive viewers for large hydrologic models. This was accomplished by using the results provided by the ADHydro development team and producing a Time Machine of surface water depth. Although the mesh was relatively small compared to the potential domain sizes of large hydrologic models, the product simplified the over 200 GB of ADHydro output into a manageable size that could be more easily be transferred to a server. The product allowed for high-resolution results of the model to be shared efficiently and interactively over the internet. Examples of cosmological models with terapixels of output that use Time Machine

suggest there is not a ceiling other than memory constraints to prevent Time Machine being used on even larger hydrological models. Based on these results, Time Machine is a viable tool for high-res water resource visualization.

4.1.2 Develop Automated Workflows

The second objective was to facilitate the generation of Time Machines based off output of large models such as ADHydro. This required the automation to be written in a language that could be ran on multiple platforms. Python was chosen as the language as many modules and APIs are available to take advantage of rendering, image processing and wrapping foreign functions in Python to incorporate them into the code design.

Rendering the output from the binary ADHydro output was accomplished using the ParaView Python API which allows for looping through a data set and exporting an image for each time step with specified contour themes. Frame processing to trim, impose transparencies based on image type, and underlay basemaps was accomplished using various Python modules that prepared each frame to be used by the Time Machine code. The Ruby-based Time Machine code was wrapped in Python and incorporated into the automated design. The viewer style and settings are written out to their corresponding data sets using Python as well. The overall automated product called TMAPS fulfills the objective of developing an automated workflow.

4.1.3 Validate Code with ADHydro Case Study

The third objective was accomplished through the use of the Upper Colorado River Basin ADHydro model. The developers of ADHydro provided the results of a lengthy run that had over

200 GB of output data. Two case studies were presented in the results above that included two distinguishing environments.

The first environment at Mount Moran is the native environment where the ADHydro results were produced. This environment lacks readily available GPU rendering. The results of this environment suggest that when rendering with ParaView CPU rendering, it is desirable to have ParaView compiled with OsMesa Llvmpipe and the MPI option enabled. The overall process was successful and the results were reasonable when multiple nodes were used to resolve the slow rendering times and multiple CPU's are implemented in the frame processing and Time Machine run.

The second environment that contributed to accomplishing objective three was validating TMAPS on an Amazon Web Service EC2 node. Using the Ubuntu 14.04 image on a GPU node with TMAPS installed allowed for the rendering much faster and the processing on multiple CPUs to also be efficient. The results of the ADHydro model were transferred via SCP and took some time so it validated that the TMAPS code is most conveniently ran on the same setup as where the ADHydro results are being generated.

4.1.4 Discover Limitations of TMAPS Process

The fourth objective is to outline the limitations and drawbacks that come from the overall process. This objective was accomplished through the use of multiple environments where TMAPS was ran.

Limitations on the Mount Moran supercomputer include CPU rendering on a single core. It is possible to split the job on multiple nodes to distribute the time on the number of time steps being rendered but it comes at an additional step of processing. This limitation has an obvious

workaround of compiling ParaView OsMesa Llvmpipe to have MPI enabled. Due to complications and permissions restrictions on Mount Moran, it was not accomplished during this project time. Another limitation found on Mount Moran includes the WMS call to retrieve a basemap for underlay is blocked due to security restrictions. The workaround for this was to download the basemap outside of Mount Moran and send it to the supercomputer separately.

One notable drawback of performing the TMAPS on the Amazon EC2 is transferring the large files to the active node. This can take long periods of time – for this reason, it is suggested to perform TMAPS on the same environment where the output is being generated.

One final drawback presented is the larger file sizes that Time Machines with basemaps pose compared to the ones that do not use basemaps. This is due to the redundant rendering of the base map on each frame. Potential solution for avoiding wasted memory due to this problem is presented in the following opportunities for continued development.

4.1.5 Explore the Possibility of Using Tethys as a Graphical User Interface

The fifth objective consisted of evaluating Tethys as a potential GUI for generating TMAPS instead of manipulating the code directly. This objective was accomplished through the development of the Tethys TMAPS app. During development of the app, attempts were made to develop it where files could be uploaded to remote Amazon EC2 nodes that would be spun up upon file submission. Soon after development, it became clear that the size of the files made it unfeasible for such a workflow. It also became apparent that workflows would be more feasible if the TMAPS user would generate the Time Machines where the ADHydro results are being ran and then migrate the product to the TMAPS app. The TMAPS app evolved to be a convenient

organizer for viewing the Time Machines generated and also serves the purpose of limiting model results to a specified audience through the Tethys login functionality in its current version.

4.1.6 Recommendations for Setup and Use of TMAPS

The last objective was to give general recommendations for the setup and use of TMAPS. In general, TMAPS is a convenient and powerful method of generating robust visualization that can be shared efficiently and effectively over the web. Currently, only the ADHydro model results are supported for generating the automated Time Machines. For additional model support, it would require users to develop a separate render script that can handle specified model output. Users can use the tools developed for frame processing and running Time Machine with the additional render scripts as they are developed to generate automated Time Machines.

Environments that would produce Time Machines most efficiently are those that have a quality GPU available and have sufficient CPUs for multiprocessing. For Time Machines that are relatively smaller in size, users should avoid basemaps.

4.2 Opportunities for Continued Development

There are a number of potential opportunities for the continued development of TMAPS and the Tethys TMAPS app based on the limitations discovered in the fourth and fifth objectives.

Addressing the opportunities for the TMAPS code, further development could be progressed towards including a variety of model output supported for rendering. Water resource managers may find using Time Machines as methods for sharing their model results as convenient and effective instead of working in more complex server environments. As additional models are supported, more of these managers could be drawn to what TMAPS offers.

One of the major drawbacks of Time Machines is when basemaps are used for geospatial context; these relatively larger video tiles could drastically reduce memory usage if they were used as an overlay onto a static basemap instead of including the basemap in the video tile. For example, OpenLayers 3 recently came out in support of overlaying video or even tiled video onto their basemaps. The opportunity for further development would consist of evaluating if the tiled video produced by Time Machine could have a transparent background around its model elements. Continued development would also consist of programmatically overlaying the video onto OpenLayers and using existing JQuery and JavaScript controls to control the video animation. If these developments were successful, it would directly affect the Tethys TMAPS app.

The Tethys TMAPS app has opportunities for continued development based on the previously mentioned OpenLayers tiled video incorporation. Tethys takes advantage of GeoServer to serve WMS, WMTS, and other layers formats to the OpenLayers maps that Tethys app developers already incorporate into their respective apps. The development would consist of determining if GeoServer could serve video tiles – if so, the tiles could be stored on the Tethys GeoServer. If not, the tiles still could potentially be stored and called as they are setup in Time Machine protocols. If successful in incorporating video tiles onto OpenLayers in the Tethys TMAPS app, then the vector components of models such as polylines featuring channels and points featuring drains or other hydraulic structures with representative vector geometry could be placed on top of the video tiles as additional layers allowing for added dimensions to project visualization and evaluation. Eventually, Time Machines could become a Tethys tool where video tiles would be directly incorporated into any Tethys app as an additional map layer.

4.3 Software Availability

The source code for TMAPS is available on GitHub and can be viewed and downloaded at https://github.com/CI-WATER/tmaps. The source code for the Tethys TMAPS app is also available for viewing and download at https://github.com/CI-WATER/tethysapp-tmaps_app. A live demo of the TMAPS app with sample Time Machines is available at http://demo.tethysplatform.org/apps/tmaps-app.

REFERENCES

(2013). "Setting up a graphics environment on an EC2 instance." Retrieved 11 Nov, 2015, from http://www.paraview.org/ParaView3/Doc/Nightly/www/js-doc/index.html#!/guide/graphics_on_ec2_g2.

Blanc, M. and J.-F. Lalande (2013). "Improving mandatory access control for HPC clusters." Future Generation Computer Systems **29**(3): 876-885.

Blower, J. D. (2010). GIS in the cloud: implementing a Web Map Service on Google App Engine. Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, ACM.

Burlen (2013, 18 Nov 2014). "ParaView/ParaView And Mesa 3D." Retrieved Sept 10, 2015, from http://www.paraview.org/Wiki/ParaView/ParaView_And_Mesa_3D.

Compieta, P., et al. (2007). "Exploratory spatio-temporal data mining and visualization." Journal of Visual Languages & Computing **18**(3): 255-279.

Feng, Y., et al. (2011). "Terapixel imaging of cosmological simulations." The Astrophysical Journal Supplement Series **197**(2): 18.

Google (18 Feb 2015). "Google Maps Engine API (Deprecated)." Retrieved 16 Oct, 2015, from https://developers.google.com/maps-engine/?hl=en.

Google (14 Nov 2014). "KML Support in Google Maps." Retrieved 16 Oct, 2015, from https://developers.google.com/kml/documentation/mapsSupport?csw=1.

Huang, B. (2003). "Web-based dynamic and interactive environmental visualization." Computers, Environment and Urban Systems **27**(6): 623-636.

Jones, N., et al. (2014). "Tethys: A Software Framework for Web-Based Modeling and Decision Support Applications."

Michaelis, C. D. and D. P. Ames (2008). Web Feature Service (WFS) and Web Map Service (WMS). Encyclopedia of GIS, Springer**:** 1259-1261.

Neelakantan, S., et al. "Open Source Web Map Application Development: Opportunities for Extension."

Open Geospatial Consortium (2015). from http://www.opengeospatial.org/standards/wmts.

Quammen, C. (2015). "Scientific Data Analysis and Visualization with Python, VTK, and ParaView."

Rouholahnejad, E., et al. (2012). "A parallelization framework for calibration of hydrological models." Environmental Modelling & Software **31**: 28-36.

Sargent, R., et al. (2010). Timelapse GigaPan: Capturing, sharing, and exploring timelapse gigapixel imagery. Fine International Conference on Gigapixel Imaging for Science, Citeseer.

Steinke, R., et al. (2014). ADHydro: A Parallel Implementation of a Large-scale High-Resolution Multi-Physics Distributed Water Resources Model Using the Charm++ Run Time System. AGU Fall Meeting Abstracts.

Swain, N., et al. (2015). "Tethys Platform: A Platform for Water Resources Modeling and Decision Support Web Apps."

Water, U. (2009). The United Nations World Water Development Report 3–Water in a Changing World, UNESCO Publishing/Earthscan.

Zhang, J. and S. You (2010). Dynamic tiled map services: Supporting query-based visualization of large-scale raster geospatial data. Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, ACM.