|epcc|

4D Visualisations of Climate Data with ParaView

Georgia Klontza

August 24, 2012

MSc in High Performance Computing

The University of Edinburgh

Year of Presentation: 2012

Abstract

In weather sciences and meteorology visualizations provide a powerful tool for a qualitative analysis of the collected climate data. They can offer essential understanding of the described phenomena especially since the data sets are extremely large.

The aim of this project was to manipulate and visualise scientific climate data and produce real time animations. The software used for visualisations was ParaView. The project focused on exploring and analysing the target architecture capabilities as well as providing a parallel implementation of ParaView software that would take full advantage of the target system. The main reason was to enable ParaView to visualise climate data in full resolution and produce real time animations.

At the end of the project a working and scalable parallel implementation of ParaView was built on the target system. Climate data were visualised in full resolution and users interaction with the data was performed in real time. Finally animations of small size data files were also produced.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my two supervisors Dr. Chris Johnson and Alastair McKinstry for their guidance and advice throughout the project.

I would like to express my gratitude to all ICHEC staff for their friendly and welcoming attitude which made the time working with them a special experience. In addition I would like to give special thanks to Nick Geoghegan for his support and valuable help throughout my project.

Furthermore I would like to give my very special thanks to Dr. Judy Hardy and Chrystal Lei for their moral support and understanding.

Last but not least, I would like to thank my mother for her support that made my studies possible.

Chapter 1

Introduction

1.1 Project Overview

In all scientific areas a qualitative analysis of the collected data and the produced results is essential for understanding the given problem. Visualisations can provide a powerful tool of analysing scientific data and results especially when the data sets are large. In earth sciences and meteorology, visualisations are extremely helpful since the nature of data is mainly observatory and the data sets are particularly large. In order to manipulate climate data creating images is necessary for better comprehension and in depth analysis. Since climate and weather are natural phenomena the most effective approach is to visualise the climate data in three dimensions as well. This will not only give the feeling of natural representation but also provide valuable information about variables that change in amplitude.

The aim of this project was to manipulate and visualise scientific climate data and produce real time animations. The project was carried out in the Irish Centre for High-End Computing, ICHEC, located in Dublin, Ireland. It included both static visualisations and real time animations. All the images were produced in three dimensions. They could be displayed on a two-dimensional screen, such as a computer screen, as well as in stereoscopic three dimensional view in the special stereoscopic screen in ICHEC visualisation room. Stereoscopy is a way of displaying different images to the left and right eye, creating the illusion of depth. The image is viewed with special glasses.

The visualisation software used in this project was ParaView [1]. The climate data were provided from ICHEC weather database. They were stored according to CF Convention standard [2] in netCDF format [3]. Due to the large size of the data sets a parallel approach was necessary for this project. A small visualisation cluster was dedicated to this cause. The data were passed to ParaView, processed and rendered in parallel in the visualisation cluster. Then the final image was assembled and was sent as an output to the local machine. The main challenge of the project was testing the available platform and delivering the most efficient implementation of ParaView that took full advantage

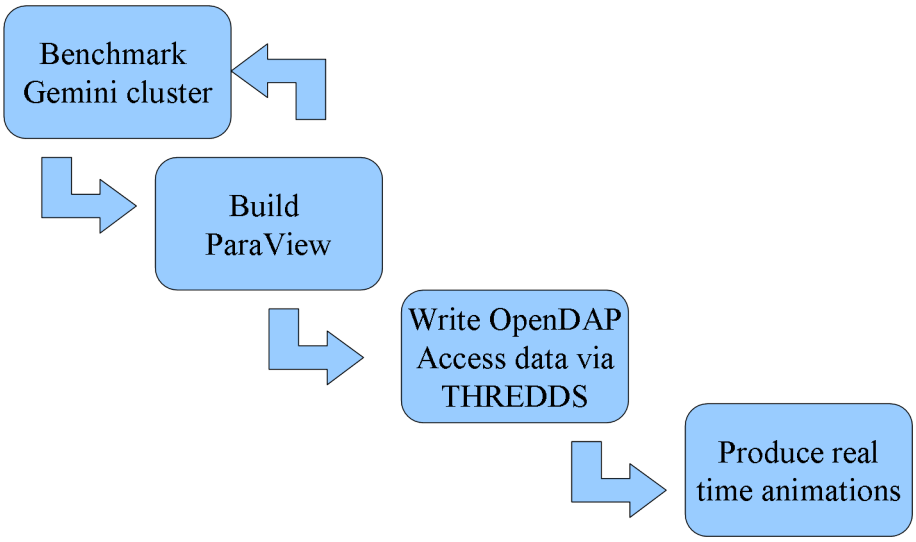

Figure 1.1: *Project work flow diagram*

of the specific platform and also to identify performance or memory inefficiencies and propose solutions to overcome them.

This work was based on the previous work of Rudhai Dervan that was carried out in ICHEC during the summer of 2011 [4]. His project involved converting the existing weather data files into netCDF format and creating visualisations with ParaView. Due to hardware inefficiencies at the time it was not possible to create proper animations with the built-in animation tool in ParaView. Also the ParaView implementation used was a serial application with poor performance. In this project a new architecture was introduced and ParaView was built as a parallel application.

1.2 Project Work Flow

The structure and work flow of the project are presented on figure 1.1.

The first part of the project was dedicated to exploring the provided hardware capabilities. Through this part several benchmarking techniques were used to evaluate the performance of the cluster. This process was essential to identify possible performance bottlenecks and hardware inefficiencies.

The second part included building different versions of ParaView. These versions are illustrated in figure 1.2. In order to take full advantage of the Gemini cluster the different versions were compared and tested. The first version was a simple serial implementation. The second version was a parallel implementation using both CPUs and GPUs. Finally the third version was a parallel implementation with CPUs only. As shown in figure 1.1 the benchmarking process continued throughout building and testing ParaView.

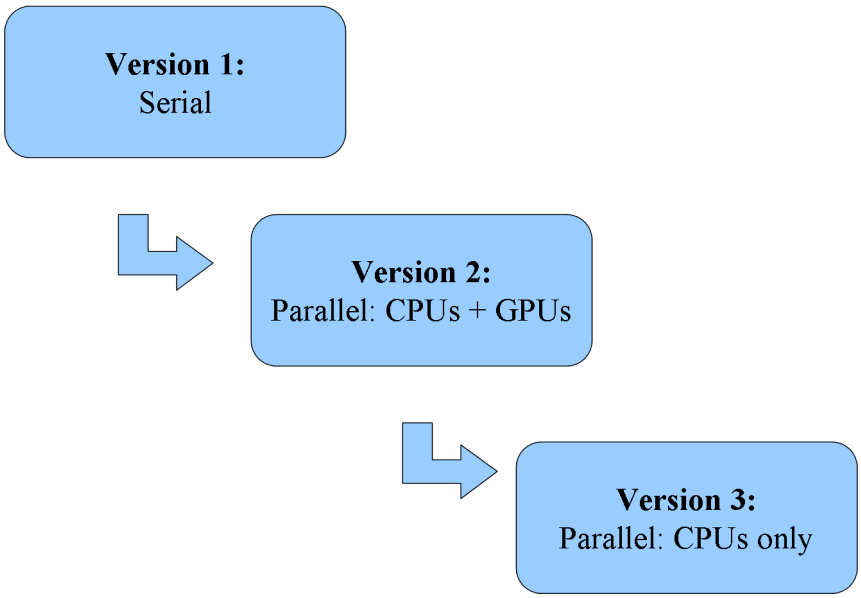

Figure 1.2: *ParaView versions*

The third part of the project was dedicated to writing OpenDAP [5] readers for ParaView. ParaView can read data files in netCDF format. This feature was extended to create readers for OpenDAP.

The last part of the project was dedicated to animations. ParaView has a built in animation tool which was used to produce real time animations.

To summarise, the aim of the project was to build a parallel implementation of ParaView software that would take full advantage of the target architecture in order to visualise climate data in full resolution and produce real time animations. Three main goals were set on the beginning of the project. The first was to investigate the performance of the Gemini cluster exploring its capabilities and identifying possible inefficiencies. The second was to experiment with different ParaView versions in order to find the most efficient parallel built for the provided cluster. The third was to expand the netCDF readers implemented in ParaView to OpenDAP readers for fast acess to climate data.

Chapter 2

Background Theory

2.1 HARMONIE Weather Model

Climate simulations and weather forecasting in Ireland are produced through a collaboration between Met Eirean [6] and ICHEC. Weather data are collected by Met Eirean from several locations across the country. These data consist of many variables containing information about temperature, humidity, pressure as well as latitude, longitude and height coordinates. These data are used as input data in the forecasting models run in ICHEC [7]. One of the main models used to forecast Irish weather is HARMONIE [8]. HARMONIE is a spectral non-hydrostatic forecasting model, which solves partial differential equations analytically.

The output data produced by this model are in GRIB, Grid in Binary, format [9]. This format is used for storing weather data in a grid in a way that in each point of the grid there is a value of a variable. Since this format is used mainly for climate, the data need to be converted in a format recognizable by visualisation programs. The data are converted and stored in netCDF format which is used by most of weather scientists.

The output files of this model were the source data in my project. Several files were stored on the Gemini cluster and were used in visualisations.

2.2 ParaView

ParaView was the most important technology that was used throughout the project. It is an open source, multi platform visualisation software. It is widely used by earth and climate scientists as it provides the user with powerful tools in the analysis and visualisation climate data. It produces high level and detailed visualisations which can be used to explore data sets both in qualitative and quantitative manner. The analysis of the data can be done either interactively with three dimensional images or programmatically via batch processing.

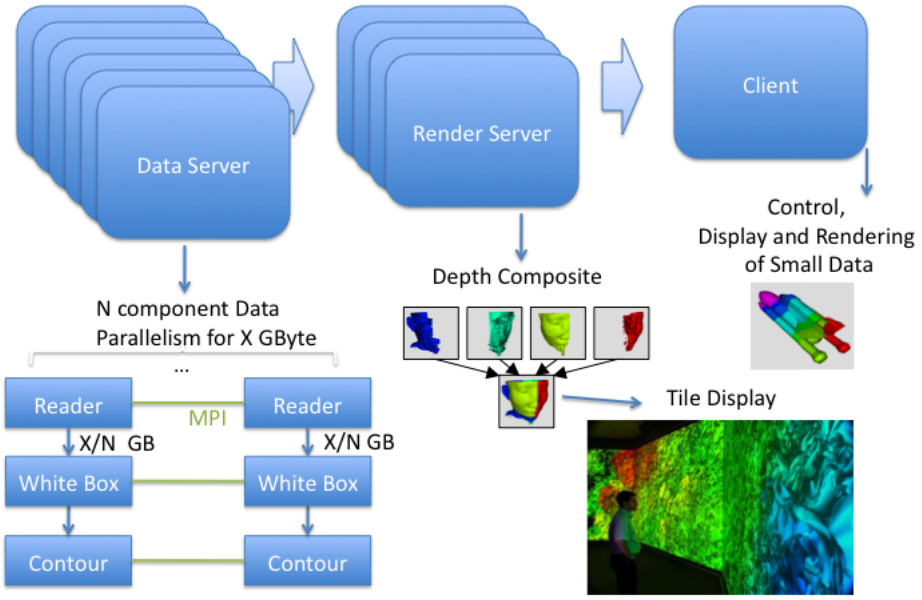

Figure 2.1: *ParaView Client - Server configuration*
[10]

ParaView software is designed to analyze a wide range of data sets. It can manipulate small files in a single machine all the way up to large datasets on supercomputers. ParaView has three main components: client, data server and render server. Client is the user interface, data server manipulates the data and produces geometric models and the render server produces the visualisations. While the client is a serial application, data and render servers can run in parallel. According to how these are combined ParaView can be built in different ways. When all three are combined ParaView runs as a serial application. This is efficient for small datasets. Since most of scientific datasets are several GBs large, the most common approach is to separate the client from the servers. Data and render servers can be either combined in a single client - multiple server configuration or separated in a single client - data server - render server configuration, according to the target platform ParaView is built on.

The provided architecture for my project was a Gemini cluster. Since it is a small cluster the most reasonable solution was to build ParaView in a single client - multiple server configuration as illustrated in figure 2.1. In this mode client is run on the users local machine while all the data manipulation and rendering take place on the server that runs in parallel in the cluster. To achieve parallelism the Message-Passing Interface, MPI, library is used.

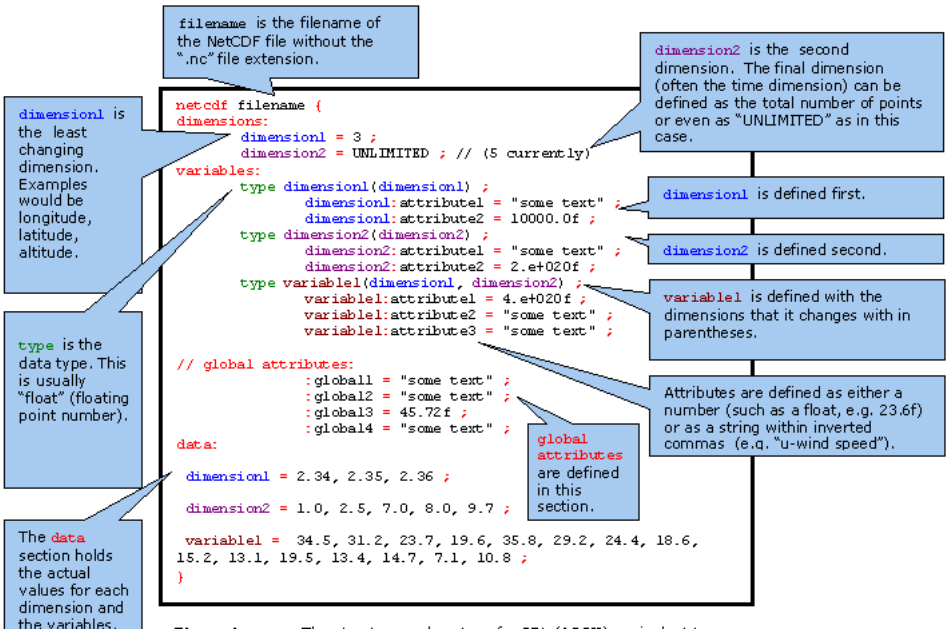

Figure 2.2: *netCDF file format*
[11]

2.3 netCDF Format

NetCDF is a data format widely used by weather scientists in meteorology, weather forecasting and climate change research. It consists of a set of libraries and programming interfaces used in creating, sharing and accessing scientific data. The netCDF format is platform independent providing portability and easy access and exchange of the files. All the files are stored in a self describing format. In each file a header is included providing a description about the layout of the file and information concerning the data arrays and metadata through name and value variables. In figure 2.2 an example of netCDF file structure is illustrated.

The three different styles of netCDF format include the the classic data model, the netCDF-4 and the netNCDF-4 classic model. The depth of knowledge on the format the user needs to have varies according to the software used to read the files. Software packages that use the reference implementation of the file are able to recognize and use the correct version of the format automatically. On the other hand there are many independent implementations of software packages reading netCDF files. In this case the user needs to specify the style of the format and provide all the details regarding the file manually.

The climate data used in the project were stored in netCDF format and read via ParaView. ParaView contains a modern built-in netCDF library and all the necessary readers are provided. Thus ParaView is able to recognize and access all the climate data in this format without any further specifications from the user.

2.4 CF Conventions

The netCF is a well defined format for manipulating and accessing climate data. One of its most useful characteristics is the portability across various platforms. To enrich the netCDF format the user of the database needs to include some extra information about the included variables. Metadata include all the information about the data in the file with a description of what each variable represents. There are several conventions that provided metadata. One of the most widely accepted is the Climate and Forecasting Convention. It was created in order to be used on climate and weather data for atmosphere, ocean and surface. The CF Convention provides each dataset with the corresponding metadata. This way every variable has a definite description of the physical quantity it represents, its temporal and spatial properties and its units. With these information available data from different sources can be identified and compared making it possible for users to build applications with strong display and extracting capabilities.

Apart from the capabilities it provides the user, it can also enable software tools to use the metadata guidelines on a file in order to perform operations and display specific subsets of the data without any involvement from the user.

Since the project involved a lot of interaction with climate data in netCDF format, CF convention provided an important tool in understanding and manipulation of the data.

2.5 OpenDAP

The Open Source Project for Network Access Data Protocol, OpenDAP, is an architecture widely used by earth scientists and meteorologists. Besides an architecture it is also a protocol that simplifies scientific data transfer. The design of the OpenDAP protocol is based on HTTP and allows access to remote data through an internet connection. The OpenDAP software provides access to data the same way a URL provides access to a web page.

The OpenDAP server is able to handle lot of different collections of data format. One of these formats is the netCDF file format. Compared to ordinary file transfer OpenDAP has two big advantages. The first is the ability to choose the specific part of the data that is need according to the provided metadata and access it, instead of transferring the whole file which could be up to several megabytes. The second one is the option to aggregate data located in different files with a single transfer. The OpenDAP protocol provides a robust and useful framework that simplifies large data transfer operations.

The Thematic Real-time Environmental Distributed Data Services, THREDDS, is a project developed aiming to bridge the gap between the providers and users of scientific data. The main goal of THREDDS project is to provide an efficient and convenient way of transferring and manipulating scientific data related to earth sciences. THREDDS is a web server providing metadata and data access through OpenDAP. The OpenDAP /

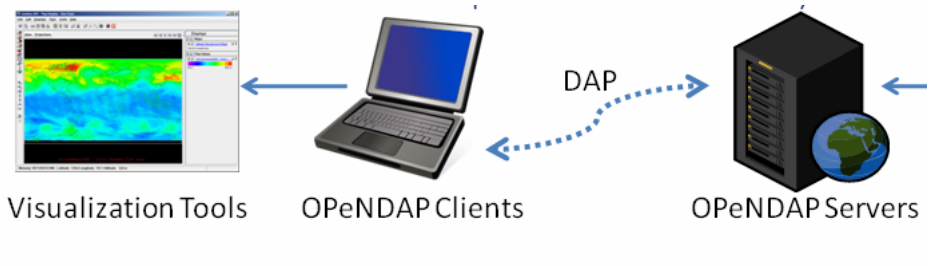

Figure 2.3: *OpenDAP and THREDD technology*

THREDDS technology is illustrated in figure **??** An important part of the project was implementing OpenDAP readers for ParaView to allow data transfer via THREDDS.

2.6 Hardware Specification

The Gemini cluster was dedicated to visualisation. It is a heterogeneous architecture cluster consisting of 2 nodes with 32 AMD processors and 2 NVIDIA M2090 GPGPUs each.

Gemini1 is the first node and it is a login and compute node. Gemini2 is the second node and is a compute node only. Jobs can be submitted to both nodes with the *qsub* command

In each Gemini node there are 8 NUMA regions. A diagram of the Gemini node is shown in figure 2.4.

As shown in figure 2.4 only 2 NUMA regions are attached to GPUs. The TORQUE configuration on the cluster schedule the submitted job to a NUMA region closest to the GPU.

Parallel applications were run in the cluster with the OpenMPI library [15].The latest version on Gemini is 1.6.

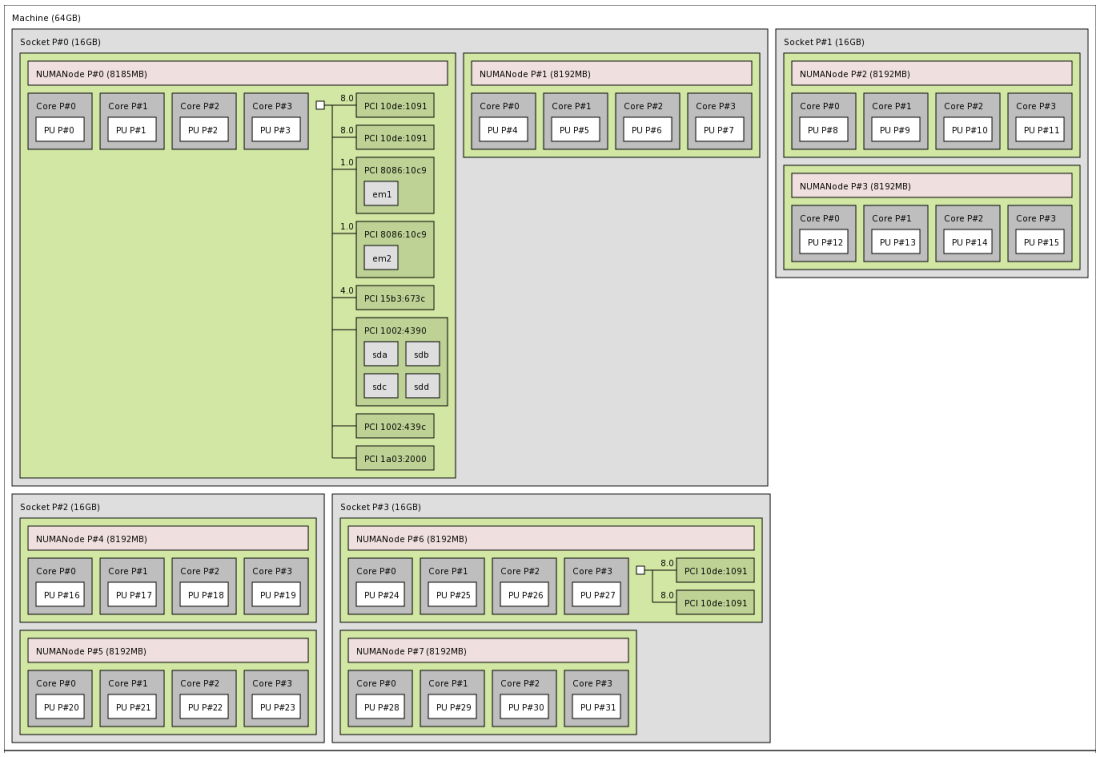

Figure 2.4: *Gemini node NUMA regions diagram*
[14]

Chapter 3

Benchmarks

The main goal of benchmarks was to investigate the performance of the target system. The aim was to produce qualitative results regarding memory performance, NUMA effects and scalability of applications. Since the Gemini cluster was dedicated to built ParaView and perform visualisations all the benchmarks were designed to address possible issues related to them.

3.1 SHOC Benchmarking Suite

The Scalable Heter Ogeneous Computing, SHOC, Benchmark Suite is a collection of programs testing the performance and stability of heterogeneous architectures. These architectures consists of multi-core processors and general purpose graphic processors, GPGPU. The benchmarks included are divided in two main categories: stress and performance tests. The stress tests use computationally demanding kernels to identify devices with component problems. The performance test are divided in levels according to their complexity.

As Gemini is a heterogeneous system, the benchmarks provided in the SHOC suite were considered appropriate for testing the cluster. From the collection of benchmarks one test from the stress collection and one of the performance collection were chosen.

3.1.1 Stability test

The stability test was performed to ensure the GPUs in the cluster performed efficiently. This test was designed to put the cluster under intense stress in order to identify devices with bad memory, inefficient cooling or other component problems. The computation used by this benchmark was a calculation of large size FFTs and inverse FFTs repeatedly.

3.1.2 Scaling

The benchmark selected from the performance collection was a test measuring the performance of a 2D, 9-point single precision stencil computation. This was used to evaluate the scaling of applications in the cluster. The test was focus on weak scaling where both the number of cores and data size would be increased. The value measured in each test was the execution time. The expected outcome was that the efficiency of the application would remain constant as the number of cores increases.

Due to delays on the original work plan caused by Gemini cluster malfunctioning this benchmark was not implemented.

3.2 Micro-Benchmarks

3.2.1 NUMA Effects

This configuration of the cluster was expected to have an impact on the performance of applications. Each Gemini node consists of 8 NUMA regions. A micro-benchmark was designed to test the impact of these regions on memory latency and bandwidth. The benchmark that was implemented was a simple ping-pong MPI program that sent the same message repeatedly between two processors. The message was an array of 500,000 double precision random numbers. It was passed back and forth between two processors for 100,000 iterations. The number of iterations and the length of the array were kept constant throughout all the tests.

The set of tests included sending the message between:

- 2 processors on the same NUMA regions
- 2 processors on different NUMA regions, both on Gemini1
- 2 processors on different NUMA regions, both on Gemini2
- 2 processors on different NUMA regions, first on Gemini1 - second on Gemini2

At the end of each test the memory latency and bandwidth were measured.

The smallest latency and greatest bandwidth were expected in the case where the two processors were located on the same NUMA region. The greatest latency and least bandwidth were expected in the case where the two processors were located on different NUMA regions and on different nodes. In this part binding processes on physical cores was necessary to ensure the processes were mapped correctly. The OpenMPI library provides several runtime arguments for binding processes. For the needs of ping pong micro benchmark the *numactl -bind-to-node* option was used. With this option it was possible to specify exactly on which node the MPI process run.

3.2.2 Memory Latency and Bandwidth

The second micro-benchmark was designed to explore the limitations memory latency and bandwidth could bring to performance. It was expected that as the data files got larger, memory delays would be a limiting factor on performance. The purpose of this micro-benchmark was to evaluate for what data file size these delays introduce a serious bottleneck. The micro-benchmark was designed to use MPI collective communications. The program first scattered a large array of double precision random numbers between all processes, performed a simple arithmetic operation on the values of the array and then gathered the values again. The outcome of this test would be memory latency and bandwidth according to increasing message sizes, for a specific number of processes.

Due to serious delays caused by Gemini hardware problems, the Memory Latency and Bandwidth micro benchmark and the scaling benchmark from the SHOC Benchmarking Suite were not implemented. It was decided to perform these tests directly from ParaView.

Chapter 4

ParaView

4.1 Building ParaView

The main part of the project was finding the most efficient built of ParaView on the specified platform. This process included building and testing different versions. All versions were built in a public directory on the Gemini cluster to ensure they could be accessed and replicated by all users.

The source code was downloaded from the official ParaView website [16] and was stored in a public directory on the Gemini cluster. ParaView versions were not built in the same directory as the source code. This way source code remained unchanged and could be used multiple times.

Even though configuring and building was a straightforward process, attention was needed to ensure dependency of directories, library paths and variables were set correctly.

A step-by-step guide with details on configuration and build is provided in Appendix A.

4.1.1 Cmake

Cmake is an open source build system providing powerful tools in compiling, testing and packaging software [17]. One of its most useful features is cross-platform compiling. This enables separating the source and the target directories, leaving the source code untouched. Another useful feature is specifying dependencies between directories automatically, hiding all the complication from the user.

Cmake software was downloaded from the official website [18]. Version 2.8.8 was built in a public directory on Gemini cluster. To increase its usability it was created as a module. Building ParaView with Cmake was a straightforward process. After the target

create target directory

enter target directory

module load CMake

ccmake path to source directory

Figure 4.1: *Cmake commands*

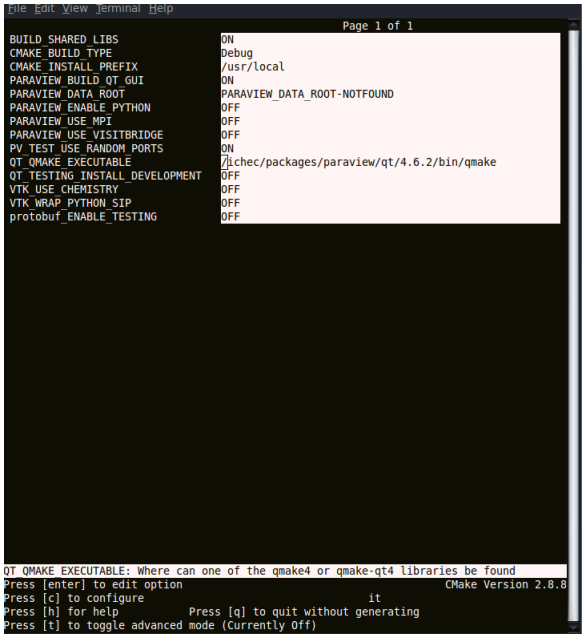

Figure 4.2: *Configuration panel for ParaView*

directory was created the Cmake module was loaded. With a simple Cmake command the source code was compiled in the target directory. The process is shown in figure 4.1

With the Cmake *ccmake* command and the appropriate path to the source code directory the configuration panel was displayed. As shown in figure 4.2 this panel included all the configuration options for ParaView to be built. Different versions of ParaView could be built by choosing the sensible flags and libraries according to the specific version needs.

When configuration was complete the *generate* option was available on the panel. This generated all the necessary make files for ParaView software. With the command *make* ParaView was built.

4.1.2 Qt Library

Qt is a cross-platform framework consisting of tools to build user interface applications [19]. Paraview uses Trolltech's Qt library for its user interface applications. This library is necessary to build the ParaView client GUI.

The Qt library source code was downloaded from the products website [20] and installed under the ParaView general directory on the Gemini cluster. Although the latest version is 4.8.2, it is not compatible with ParaView. ParaView still uses version 4.6.2 that is known to be stable.

4.1.3 Mesa Library

Mesa is an open source 3D graphics library [21]. Its implementation is based on the OpenGL specification [22], a system used for graphics and interactive rendering. In most Linux based systems user interfaces and graphics are provided by the X Window System [23] and the OpenGL library. Although this is the standard for local machines, it is not always the case for computer clusters. Mesa library can be used as an alternative to OpenGL library in systems with ill functioning X display configuration or absent of graphics software.

Mesa library source code was downloaded from the official website [25].Version 8.0.3 was downloaded and installed on Gemini cluster under the general ParaView directory.

As ParaView is a visualisation application it requires graphical and rendering software. By default it uses OpenGL as a rendering library. Mesa library can be a ParaView build option in cases of parallel implementation in a cluster without graphics hardware. A description on building ParaView with Mesa support can be found on ParaViews website [24].

OSMesa In the case where the provided hardware does not have an X display configuration the Mesa library provides the option of CPU based rendering. This is possible with the off-screen rendering interface OSMesa [26]. Instead of window displays, the rendering data are placed in a buffer in the main memory.

When ParaView is built with OSMesa support the client user interface is not available. This configuration can be used to build only the server components.

4.2 ParaView Implementations

4.2.1 Serial

The first version of ParaView was a serial one as it is the simplest and easiest to build. In this version the client, data and render server are combined into a serial application. This version is useful only for manipulating very small data sets. For larger problems it is too hard to visualise the data as the application is slow.

Variable	Value	Description
PARAVIEW_BUILD_QT_GUI	ON	This variable indicates the build of ParaView Client User interface
QT_QMAKE_EXECUTABLE	Qt library path	This variable specifies the directory Qt library is built in

Table 4.1: *Cmake variables for ParaView serial built*

Variable	Value	Description
BUILD_SHARED_LIBS	ON	Set on to build with shared libraries
PARAVIEW_BUILD_QT_GUI	ON	Set on to built Client User Interface
QT_QMAKE_EXECUTABLE	Path to Qt executable	Set to point to Qt library executable file
PARAVIEW_USE_MPI	ON	Set on to built ParaView with MPI
MPI_C_INCLUDE_PATH	Path to MPI include directory	Set to point to MPI include directory
MPI_C_LIBRARIES	Path to MPI library	Set to point to MPI library files
PARAVIEW_ENABLE_PYTHON	ON	Set on to enable python scripting

Table 4.2: *Cmake variables for ParaView parallel built both on CPUs and GPUs*

This version was chosen in order to familiarize myself with configuration options of CMake and building the application from the source code. Also it insured all the libraries needed were installed properly and running correctly. During compilation with Cmake the corresponding variables were set to indicate the correct path to the library files. These variables are shown in table 4.1

4.2.2 Parallel on Gemini with X Displays

The second version was a parallel version with CPUs and GPUs. In this version the client and server were separated. There is a single server component combining both the data and the render server. This way the data manipulation was performed on the CPUs and the rendering took place on GPUs. The client was run on a local machine while the server was run in parallel on Gemini cluster.

In order to enable ParaView to utilize the GPUs on the cluster, it was built with the OpenGL library support. As this version was designed to run in parallel, the use of OpenMPI library was necessary. The variables that were change during Cmake configuration are illustrated in table 4.2.

Variable	Value	Description
PARAVIEW_BUILT_QT_GUI	OFF	Set off for no Client User Interface
OPEN_gl_LIBRARY	OpenGL gl library	Set this variable to point to Mesa/gl.so file
OPEN_glu_LIBRARY	OpenGl glu library	Set this variable to point to Mesa/glu.so file
VTK_OPENGL_HAS_OSMESA	ON	Set on for OSMesa support
OSMESA_INCLUDE_DIR	OS Mesa directory	Set this variable to point to OSMesa directory path
OSMESA_LIBRARY	OS Mesa library	Set this variable to point to Mesa/libOSmesa.so library file
VTK_USE_X	OFF	Set off to disable X displays

Table 4.3: *Cmake variables for ParaViw build on CPUs*

4.2.3 Parallel on Gemini without X Displays

Tha third version of ParaView was a parallel version designed to perform both data manipulation and rendering only CPUs. This is possible with OSMesa library support which enables all the rendering to take place off screen. As OSMesa is a strictly CPU library a client component could not be built. This was not a problem as the server could connect with any of the clients built in versions described in sections 4.2.1 and 4.2.2.

Configuring ParaView with OSMesa support needed more attention to change correctly all the corresponding variables. Cmake compiling variables that were changed in this version are demonstrated in table 4.3.

4.3 ParaView Application

The latest version of the ParaView source code is 3.14.1. When ParaView was built from this version there were several runtime problems. The most frequent problem was failure in starting the client component in parallel builds. Even though the client is a serial application it failed to start quite often displaying error messages about the OpenMPI implementation. The same problem appeared in the serial version as well where no parallelisation was implemented. At first it was considered a misconfiguration issue and both serial and parallel versions were built again. When the problem persisted a new copy of the source code was downloaded, reconfigured and built. This approach did not solve the issue so the problem was reported in ParaView developers team as well as in ParaView mailing lists [27]. Through these reports it was found that several users were facing the same problems caused by bugs in the specific version. After version

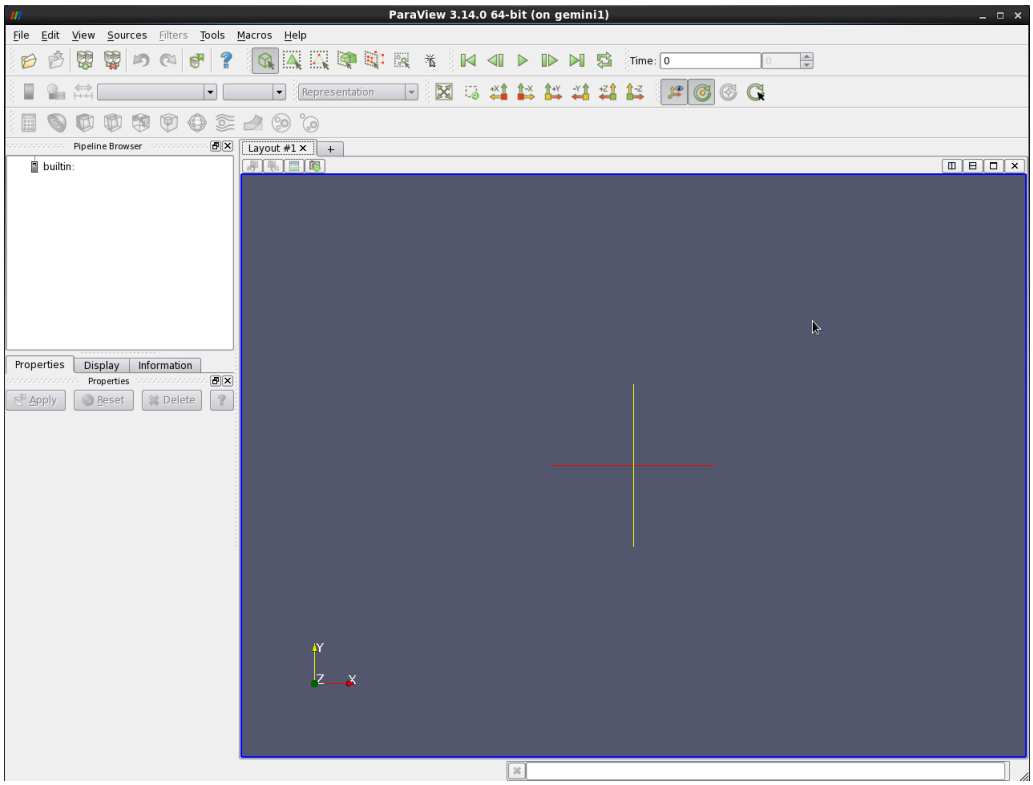

Figure 4.3: *ParaView user interface*

3.14.0 was installed these issues were resolved.

4.3.1 Serial Run

ParaView in the serial version was run directly from ParaView directory with the command: *./paraview*. The client and servers are combined in a single application. ParaView client user interface is demonstrated in figure 4.3.

4.3.2 Parallel Run

In order to run the ParaView server in parallel on the back end of the cluster batch mode was necessary. In this mode the server is launched in parallel via a script file submitted in the cluster as shown in figure 4.4. The batch file for ParaView is available in Appendix B.

To enable the rendering on GPUs only the runtime display device number flag was needed.

The most important part of running the client server mode was to establish the connection between client and server components. The connection options were managed by

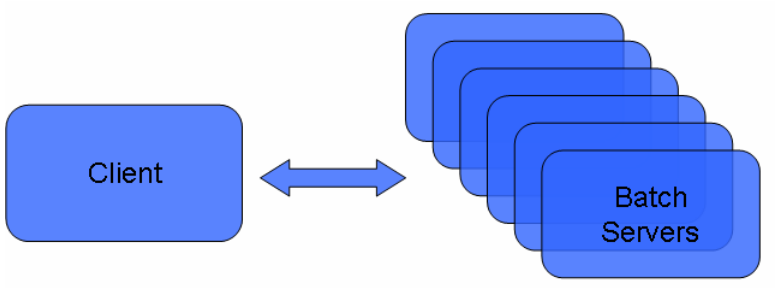

Figure 4.4: *ParaView server batch mode*

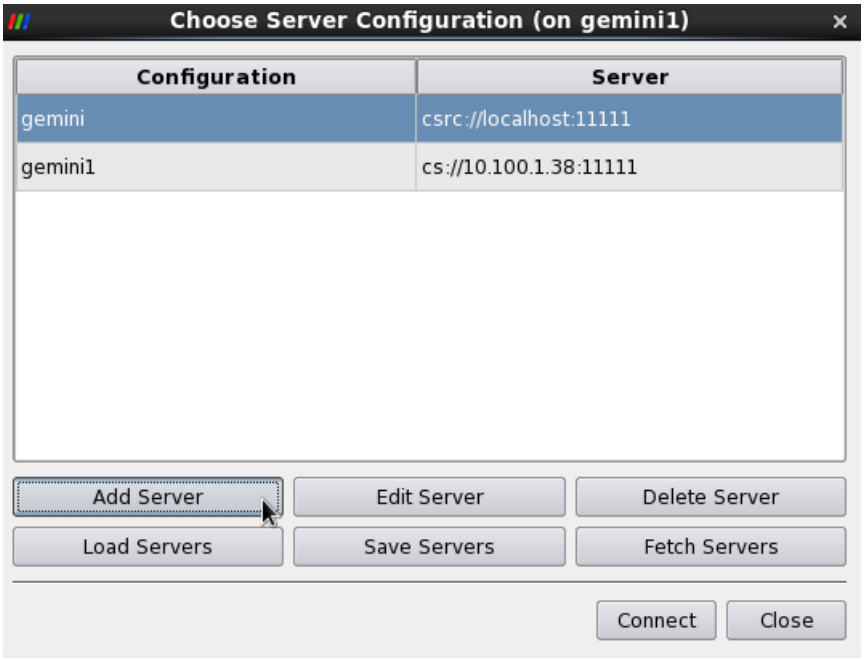

Figure 4.5: *ParaView server configuration options*

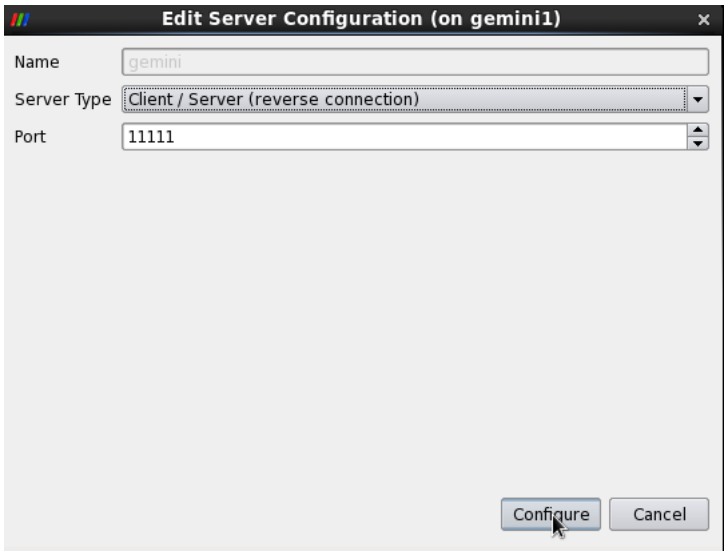

Figure 4.6: *ParaView reverse connection options on client*

the client as shown in figure 4.5. There are two options of connections: the forward and the reverse connection.

Forward is the default connection where the client will connect to the server. The server host IP address and port are specified directly from the client. This connection is the simplest but can not be used if the server is protected behind a firewall. As the firewall blocks incoming requests but allows outgoing requests the reverse connection provides the solution to make the connection possible. In the reverse connection the server connects to the client. The client host IP adress is specified on the server with the command

dsagfas

The client needs to wait for the incoming request from the server to establish the connection.

The server was run in parallel on the Gemini cluster. As the Gemini cluster was behind a firewall, the reverse connection configuration was chosen as shown in figure 4.6. The last step of configuration was to specify how the server would be started. As the server would be submitted to the back end of the cluster the launch configuration option was set to manual as shown in figure 4.7.

When configuration was complete the batch script for the server was submitted to the Gemini cluster and the connect option was selected on the client.

4.4 Animation

Most of the climate data include a time dimension. This can be used on ParaView to create animations in order to study both weather conditions and simulations in real

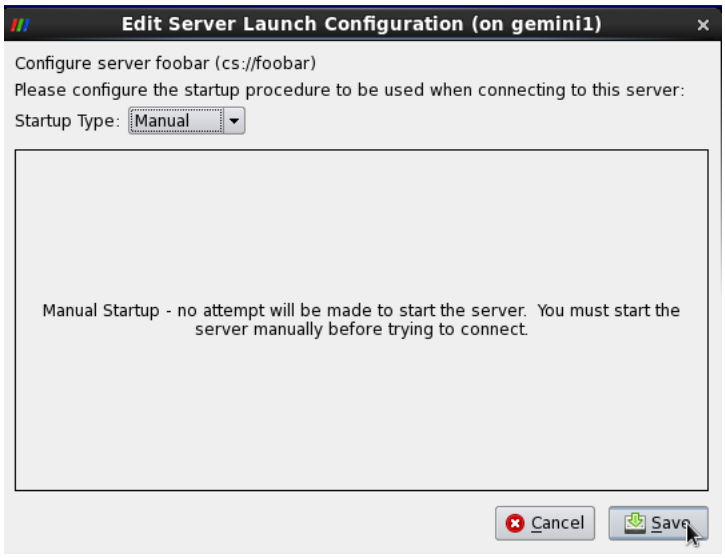

Figure 4.7: *ParaView server launch configuration options on client*

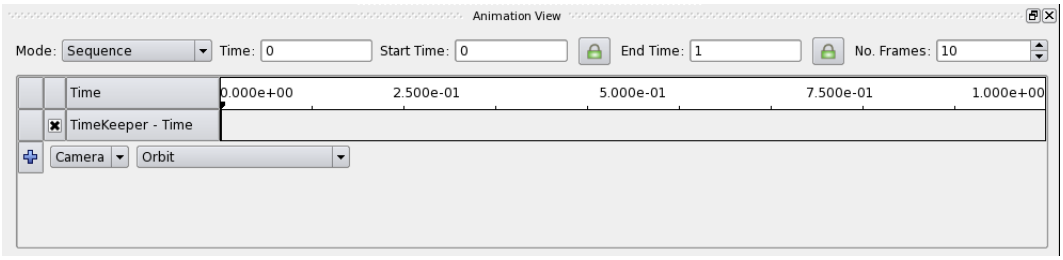

Figure 4.8: *ParaView animation tool*

time and long term changes of the climate. ParaView has a built-in animation tool to visualise climate data in all four dimensions: the three spatial dimensions and time.

The panel for animation settings is shown in figure 4.8.

Chapter 5

Results and Analysis

5.1 Benchmarks

5.1.1 SHOC Benchmarking Suite Stability Test

The stability test was run on both nodes of the Gemini cluster. In table 5.1 the specification parameters are shown. These parameters include information about the GPU type and device name, the problem size, the number ofiterations the FFTs and the inverse FFTs were calculated

The Stability test uses the FFTs as they have a sensitivity to small errors. The test cross checked the minimum, maximum and mean at values every step. If there are no differences the test continues. If not it reports a failure of the specific step. As shown in table 5.2 there were no failures during the calculations of FFTs and inverse FFTs. Thus the GPGPU installed on the Gemini cluster were working properly.

5.1.2 Ping-Pong Micro-Benchmark

The results of runtime, memory latency and memory bandwidth of ping ping micro-benchmark are illustrated in figures 5.1, 5.2 and **??** accordingly. The first three bars

Device Name	Index	MBs Tested	Blocks	Number of FFTs
Tesla M2090	0 Gemini1	5294.38	1355376	1355360
Tesla M2090	1 Gemini 1	5294.38	1355376	1355360
Tesla M2090	0 Gemini 2	5294.38	1355376	1355360
Tesla M2090	1 Gemini 2	5294.38	1355376	1355360

Table 5.1: *Stability test parameter spicification*

Test atts	Units	Median	Mean	Stddev	Min	Max
Check (max)	Failures	0	0	0	0	0
Check (mean)	Failures	0	0	0	0	0
Check (median)	Failures	0	0	0	0	0
Check (min)	Failures	0	0	0	0	0
Check (stddev)	Failures	0	0	0	0	0

Table 5.2: *Stability test results*

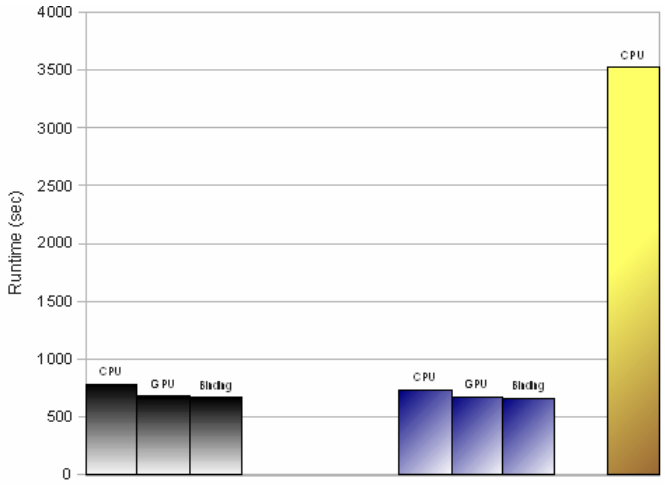

Figure 5.1: *Ping Pong runtime on Gemini 1, Gemini 2 , Gemini*

represent the results when both processes where located on Gemini 1. The second three bars represent the results when both processes where on Gemini 2. The last bar represent one process on Gemini 1 and one process on Gemini 2.

In each set of tests the runtime, memory latency and memory bandwidth were measured when the processes run on CPUs only, on GPUs only and with binding options.

The results when both processes run on the same node were quite expected. The slightly unexpected result was the speed up gained by utilizing the GPUs on the node. Even though it was very small, since there was no computational work involved, it was not expected. Even though there were no suggestions to explain it, it was not investigated further as the difference with the CPU runtime was really small.

The timing for binding processes with the closest GPUs was almost the same as without binding. There have been several issues concerning binding processes to NUMA regions. Two different OpenMPI binding arguments were compared: *-numactl* and *-hwloc*. The results were the same for both of these options. There were several problems with binding options caused by the TORQUE configuration on the cluster.

The most interesting results were produced when the whole cluster was used. As shown in the figures 5.1, 5.2 and **??** there is a big difference on runtime, bandwidth and latency between processes on the same node and processes on different nodes.

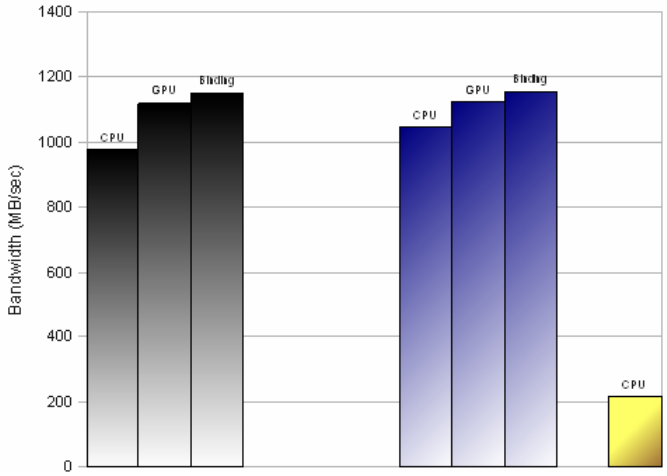

Figure 5.2: *Ping Pong bandwidth on Gemini 1, Gemini 2 , Gemini*

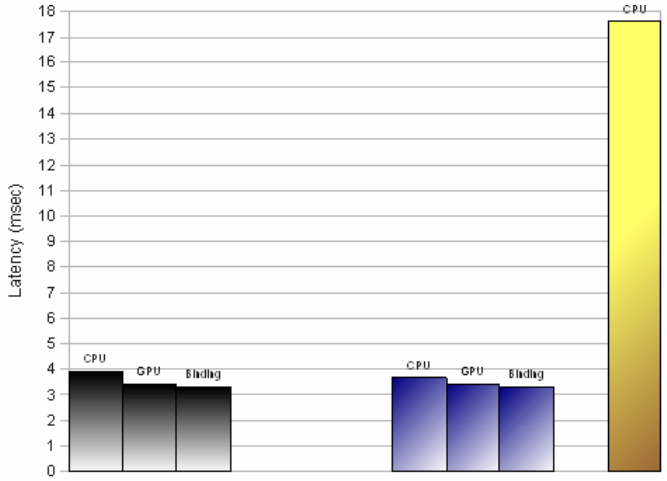

Figure 5.3: *Ping Pong latency on Gemini 1, Gemini 2 , Gemini*

A performances bottleneck between the two different nodes resulted in high latency and small bandwidth measurements. The two possible causes of this bottleneck were that it was either a network performance issue or an MPI communications issue. . The two nodes of Gemini cluster are connected with an InfiniBand which should not cause any delays. On the other hand if OpenMPI was not implemented correctly the messages could be transferred through an IP over InfiniBand.

This issue along with the suggestions indicating possible problems was handed over to the system administrator for further research. The OpenMPI version 1.4 on the cluster was replaced by the version 1.6 on the cluster. By the end of the project it was still not clear what was causing the bottleneck between the two nodes.

5.2 ParaView

Rendering times play a key factor in the quality and speed of visualisations. The ‘still rendering time indicates the time needed for the image to be rendered and displayed on the client as well as the time needed for the image to be restored after users interaction with the mouse. Small still rendering time ensures fast image reconstruction and full resolution of the graphics. The ‘interactive rendering time indicates the quality and resolution of the image through interaction. When the user interacts with the image through the mouse a new frame must be created with the new coordinates selected. Interactive rendering time is the time each new frame needs to be rendered and displayed. Small interactive rendering times produce a sequence of images during interaction.

5.2.1 Parallel with X Displays

After the first parallel version of ParaView was built several tests were carried out to measure the rendering time of the application. When ParaView was run as a serial application the rendering time would increase as the size of the input files increased as well. With the parallel version it was expected that both the still and the interactive rendering times would be reduced. The goal was to reduce interactive rendering time to under 100ms to enable real time image interaction.

The first set of tests included weak scaling of ParaView on the Gemini cluster. The input file size would increase as more processes were added as shown in table 5.3. As more cores processed larger input files the expected outcome would be rendering time to remain approximately constant. Each test was performed four times and the mean values for rendering time were calculated. The result for still rendering time is illustrated in figure 5.4

The result for interacting rendering time is shown in figure 5.5.

The results obtained were really unexpected. First of all, instead of stable rendering times expected on weak scaling, the timings kept increasing with the number of pro-

Number of Processes	Input File Size (MB)
8	67.512
16	100.825
24	174.519
32	352.748
40	479.624
48	637.178
56	702.251
64	3222.761

Table 5.3: *ParaView attributes used in weak scaling tests*

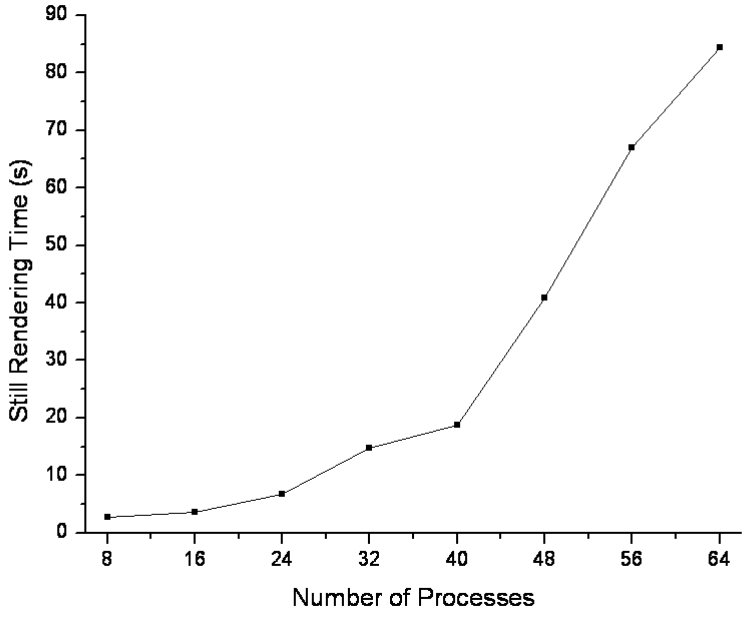

Figure 5.4: *ParaView parallel version with X displays still rendering scaling*

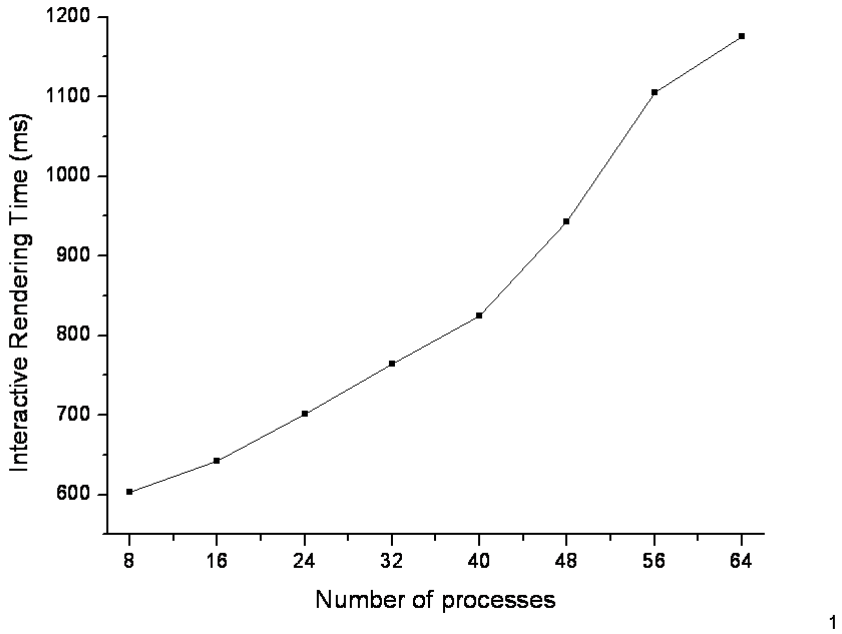

Figure 5.5: *ParaView parallel version with X displays interactive rendering scaling*

cesses. Second and most important the still rendering time not only increased with the number of processes but the value for 64 cores was 84s. The fact that still rendering was so slow indicated a serious problem with the application.

The first explanation was a ParaView misconfiguration. ParaView was built again from source with great care on configuration and compilation variables. The tests were run again but the results remained the same.

Second the GPU runtime command was disabled so the rendering would take place through the OpenGL library. This approach did not resolve the problem either.

The next choice was to disable the on-screen rendering command to force all rendering to be performed on the cluster. This approach not only did not solve the problem but it produced further errors. When ParaView was run with the off-screen rendering option, it produced the error message that remote rendering could not take place on the server side and would be disabled. The image appeared normally on the client but when interaction was tried there was no display as shown in figure **??**. When the interaction stopped the static image was visible again.

Results Analysis

After careful consideration it was found out that there as a problem with the X displays on the cluster. The parallel server could not use the X displays and OpenGL on the

cluster. While the server was able to perform the data manipulation in parallel on the cluster, it was not possible to continue doing the rendering in parallel as well. Instead each one of the parallel servers opened an X window on the local machine the client run on. The data from each server were sent on this window where the rendering took place. After rendering was complete each part of the image was combined in order to produce the final visualisation.

This explained why the still rendering appeared to be so slow. As the number of processes increased so did the X windows on the local machine. Instead of parallel processing and rendering followed by image reconstruction on the server side, only the data processing was taking place on the cluster. This caused a bottleneck on performance as all rendering took place on the local machine the client was run on.

The problem was handed over to the system administrators but it was not possible to find a working solution. For this reason a new version of ParaView was built without any X displays.

5.2.2 Parallel without X Displays

As a misconfiguration of X displays on Gemini cluster was discovered ParaView was built with OSMesa support using CPUs only disabling X graphic displays. Without graphical support only the server components were created. The client was run from the serial implementation of ParaView and connected to the corresponding server of this version. The same round of tests were run again with the attributes for weak scaling shown in table 5.3. Timing data were collected from ParaView timer log information collector. The results for still and interactive rendering times are displayed in figures 5.6, 5.7 accordingly.

Results Analysis

The still rendering time in this version varied from 0,1s to 0,2s, while the interactive rendering time varied from 65ms to 75ms. Even though the results were pretty close to weak scaling, both still and interactive rendering times had variations as the number of processes and input file size increased

However the most important part of these results was the dramatic decrease of both still and interactive rendering times. Compared to the parallel version with X displays, the still rendering performed approximately 40s faster and the interactive render performed approximately 600ms faster.

Another important result of this version was that the goal of achieving interactive rendering time smaller than 100 ms was achieved. This enabled for the first time real time image manipulation from the user, with the image remaining in full resolution throughout the interaction.

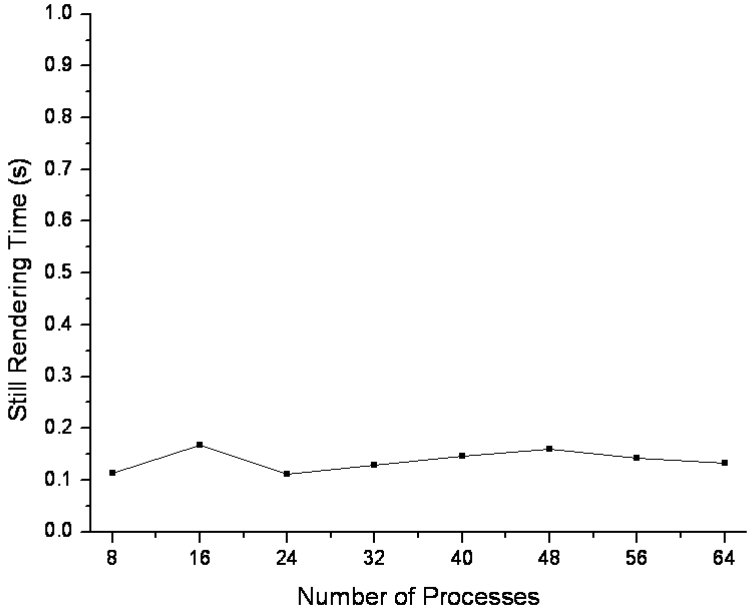

Figure 5.6: *ParaView still rendering scaling*

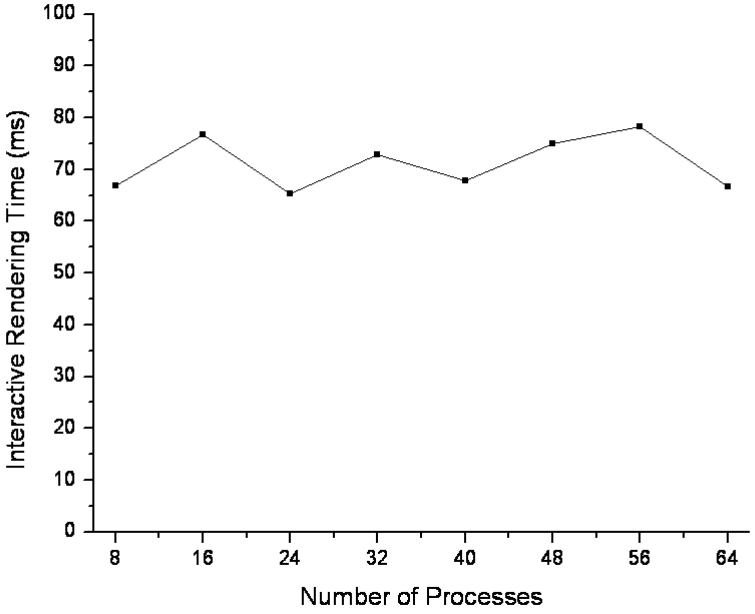

Figure 5.7: *ParaView interactive rendering scaling*

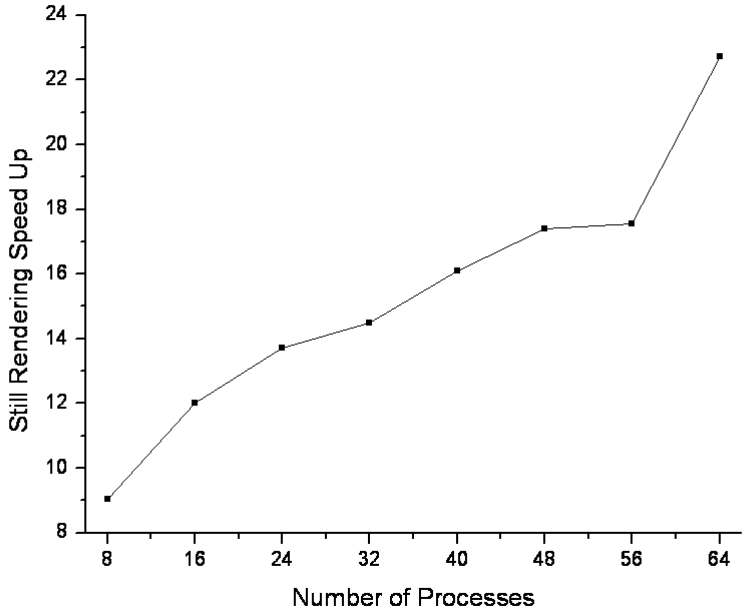

Figure 5.8: *ParaView still rendering speedup*

Thus it was concluded that the server on version without X displays was run properly as a parallel application. After the comparison between the two version it was decided that the ParaView version without X displays was the best configuration for the target architecture.

5.2.3 Parallel without X Displays Speed Up

Speed up of a parallel application is defined as the increase in performance the parallel application gained compared to its serial version. For a fixed problem size the ideal speed up increases linearly as the number of processes increases. It is evaluated by the equation

$$S = T_1/T_p$$

Where T is the time of the serial application and T_p is the time of parallel application run on p processes

The results of still and interactive rendering time are displayed in figures 5.8 and 5.9 accordingly.

As shown in in figures 5.8 and 5.9 the speedup approached linear scaling.

An example image as it is produced from this version is presented in figure 5.10

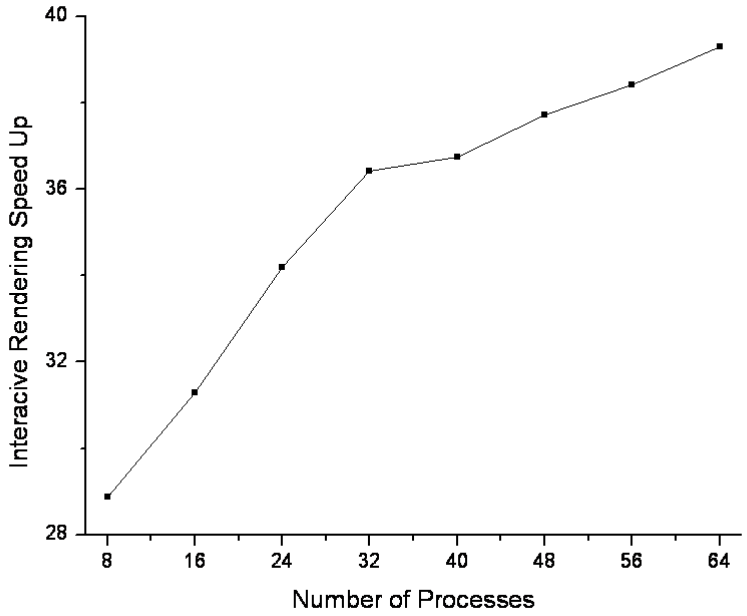

Figure 5.9: *ParaView interactive rendering speedup*

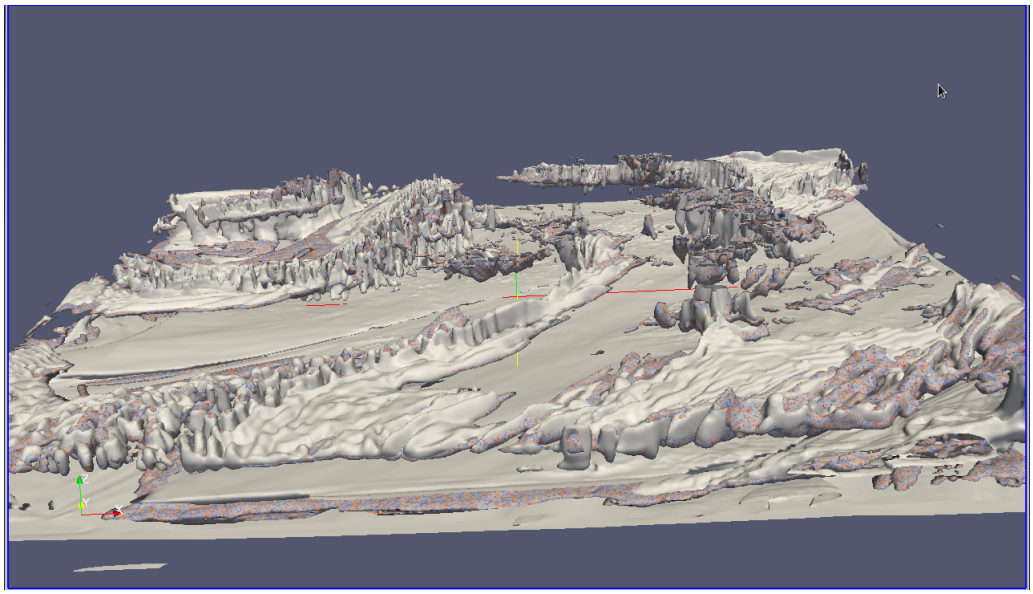

Figure 5.10: *Visualisation of clouds with ParaView*

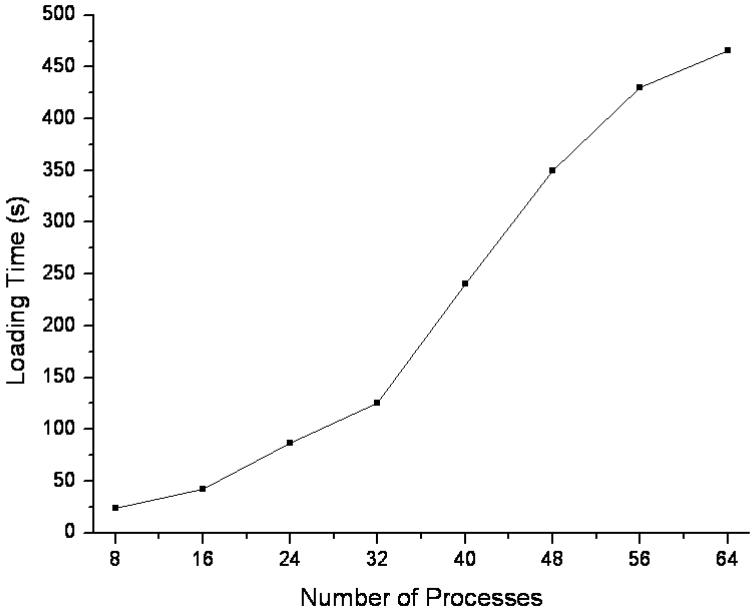

Figure 5.11: *ParaView CPUs version loading time scaling*

5.2.4 Parallel without X Displays Loading Input File Time

When ParaView is used to visualise large data sets the loading time of the input files can cause a bottleneck in performance.

For this reason tests measuring the loading time for various file sizes were performed. As ParaViews weak scaling is almost constant, the size of the tested file was chosen according to the number of processes as shown in table 5.3. The results are displayed in figure 5.11.

As shown in figure 5.11 the loading time increased as the number of cores and file size changed. It was expected to have the same behavior as the rendering timings shown in figures 5.6 and 5.7. It was not expected that the times would increase considering the fact that the files were stored in a shared memory directory on the Gemini cluster. A possible explanation was that processes instead of entering the part of the memory located on the core they were run on, accessed random memory parts. For example a process run on NUMA region 0 would access the memory on NUMA region 7 and vice versa. To test this hypothesis the processes were forced to access the memory on the core they were physically located with binding commands. For this purpose the OpenMPI library binding command *-bind-to-core* was used.

All the tests were run again with binding. The results remained the same. Another suggestion was that the bottleneck was caused because the data were stored in the cluster as serial files. When the parallel server was launched all the processes had to access the

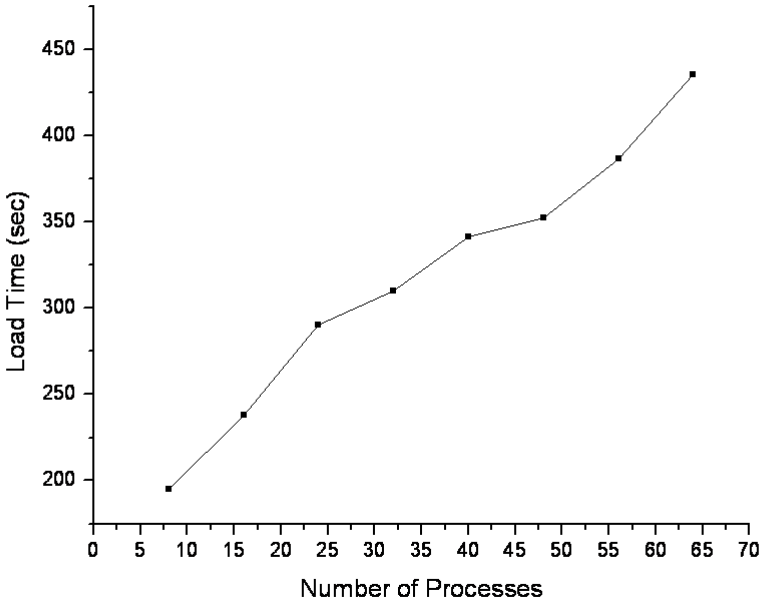

Figure 5.12: *ParaView 702MB file loading time scaling*

same file at the same time.

In order to confirm this assumption the loading time of the same file was measured as the number of processes increased. The chosen file was 702MB and the loading time was measured as the number of processes increased. The results that are illustrated in figure 5.12 show that the loading time increases almost linearly with the number of processes.

One solution could be a proper input output, IO, data base that would allow the data to be stored in parallel in a way that each process could have a part of the data at the beginning of the application. A proper IO database was not possible to be created on the Gemini cluster due to the time limitations of the project.

5.2.5 ParaView Performance on Different Client Hosts

One of the goals set in the beginning of the project was to create a parallel ParaView implementation on the Gemini cluster to enable scientists to run the client on their local machine and connect to the parallel server on Gemini. For this reason it was important to run the client component on different machines and monitor the performance of the parallel server. The only change made for these tests was to specify the client host on the server batch file.

The performance and scaling were not expected to change drastically as all the data

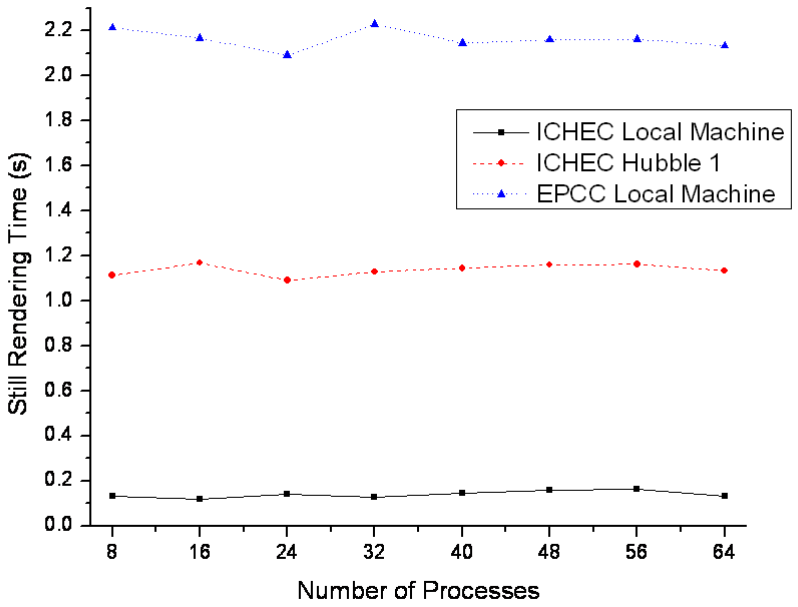

Figure 5.13: *ParaView still rendering on different client hosts*

were manipulated and rendered in parallel. The image was assembled and zipped on the server side and sent to the client as a simple image. The expected outcome would be delays in timing due to the network connections. First the client was run on my local machine in the ICHEC office. Due to fast network connection there was no difference in either still nor interactive rendering time.

Then the client was run from hubble 1, a machine located in the visualisation room in the ICHEC office. Even though both machines were located in the ICHEC office and shared the same network connection, a 0.6sec delay appeared on the results. This delay was not caused by the image transfer through the network. The client that run on hubble was set to stereoscopic mode visualisations in three dimensions. In this mode two images were displayed at the same time, one for the left and one for the right eye creating the illusion of depth in the visualisation. The displayed image was viewed with special glasses. As two images were displayed instead of one the delay in rendering time was explained. Despite that, interactive rendering still produced images in full resolution.

The last test was performed in EPCC. The client was run on a local machine in EPCC and was connected to the server on the Gemini cluster in ICHEC. This time network connection was expected to cause delays on rendering timings depending on the speed of the connection. The mean value of these delays was 2 s. During interaction the image was still in full resolution but the images no longer appeared to run smoothly in sequence. Instead discrete frames were displayed. Concerning the fact that the client and the server were run in different countries the result is pretty satisfactory. The com-

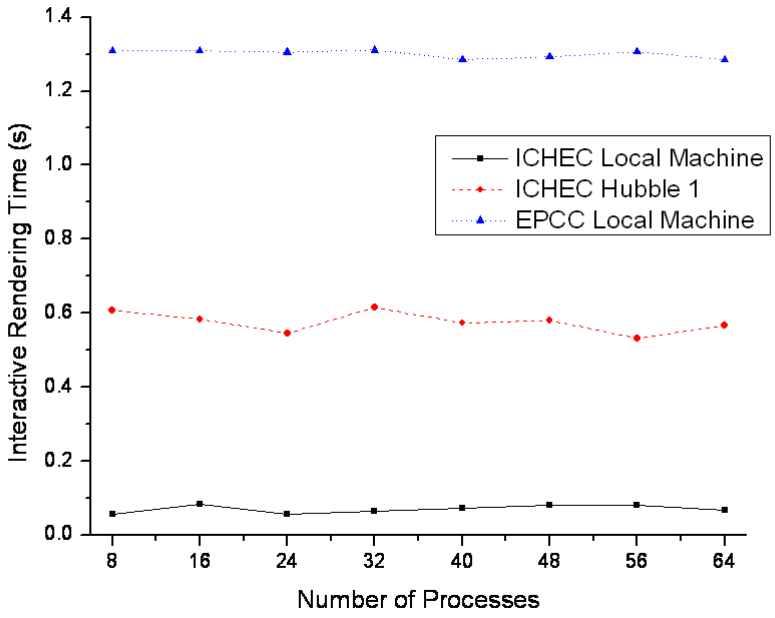

Figure 5.14: *ParaView interactive rendering time on different client hosts*

parison of still rendering time between the client on the ICHEC local machine, the client on ICHEC hubble 1 and the client on the EPCC local machine is shown in figure **??**. The comparison of interactive rendering time on the same machines is illustrated in figure **??**

5.2.6 Animations

The last part of the project was dedicated to producing real time animation with the in-built tool in ParaView. The key factor in animation is the interactive rendering time. To produce real time animations the interactive rendering time should be less than 100ms in order to display approximately 30 frames/s.

As illustrated in figure 5.7 the interactive rendering time for parallel implementation with OSMesa varied from 60 - 80 ms. Even though timings for interactive rendering are small, real time animations were possible only for small files. The maximum file size for animation was 60MB. For larger files instead of real time animation discrete frames were displayed. The display rate was 5 frames/sec.

As the interactive rendering was performed in real time it was expected that the animations would be produced in real time as well despite the size of the file. The most possible explanation for this behavior was the OSMesa off screen rendering implementation.

As OSMesa is strictly a CPU library without X displays the rendering of the data is not stored as images but as values in a buffer on the main memory. For a new image to be displayed all the information about the previous image needs to be cached in the buffer. While performing animations a lot of frames need to be cached one after the other in the buffer. As the size of the file grows larger so do the size of the data that are cached. This could cause a serious bottleneck on the performance and not allow the animation to take place.

Chapter 6

Conclusions

Overall I felt this was a very successful project. The original expectations and the goals set in the beginning were fulfilled. There were some parts of the project that I believe I could have accomplished more if there was more time. However considering the available time I feel the biggest amount of the work was achieved. I managed to investigate the different parallel versions of ParaView on the Gemini cluster and deliver the most efficient built. The ParaView parallel 3.14.0 version is installed in a public directory and can be accessed and used efficiently. This parallel built of ParaView enabled real time interaction of the climate data in full resolution, something that was not possible with the serial version. The second important feature introduced with this version is the animation of small data sizes.

One of the parts I would have liked to have accomplished more is the implementation of OpenDAP readers for ParaView. The initial implementation is already written in ParaView source code. Again due to time limitations I was unable to finish the tree structure required for OpenDAP in the ParaView directory that manages the input output of vtk files. Despite this being incomplete, I believe it is a good starting point and it should not be too hard for the readers to be completed. All the progress and changes are documented in ICHEC's visualisation internal wiki page so it should be easy for someone to continue and complete this work.

Another area of my project I would like to have experimented more with, is ParaView with X display rendering. It would be really interesting to have more time to figure out the problem with the OpenGL library on the Gemini cluster. With a parallel version where the rendering could be performed on the graphic cards on the cluster it may have been possible to generate more than 20 frames/sec and produce real time animations for larger data files.

Overall I believe my project was successful. It was a great learning experience which I enjoyed wholeheartedly. I have gained valuable experience by working in a big project as part of a team which I believe will help me in my future career. Moreover the fact that I was in a different country just added to the great experience.

6.1 Future Work Ideas

There are two main categories that provide opportunities for further work.

The first one concerns the parallel implementation of ParaView with graphical rendering. The X displays on the Gemini cluster were not working properly and it was not possible to utilize neither the OpenGL graphical library nor the GPGPU on the cluster. This is an interesting case that requires further research. It presents a new challenge to built ParaView with remote graphical rendering. this can lead to utilizing the GPGPU on the cluster as well.

The second interesting challenge is completing the OpenDAP readers on ParaView source code. The initial implementation exists and need to be taken further. The was not sufficient time to implement all the component that would enable ParaView to access files through OpenDAP. Thus there is no solid idea of possible difficulties or specific problems. Nevertheless it is an exciting project providing a lot of opportunities for further research.

Chapter 7

Project Evaluation

This part provides a general evaluation of the project. In the beginning several goals where set. The amount of work was divided to provide sufficient time for each one of them. On the other hand several risks with varying impacts were identified and a strategy was made to deal with them in case they actually appeared during the project.

Now that the project is finished I can compare with the initial plan and make a general evaluation of the progress and the work that was completed.

7.1 Goals

Before the start of the project three main goals were set. First of all benchmark the target architecture and provide an analysis about the system capabilities and identify possible issues that would effect the performance. Second one was to built a parallel version of ParaView visualisation software and perform real time animations. Last one was the implementation of OpenDAP readers for ParaView in order to access file via THREDDS.

Most of the original goals were fulfilled. The Gemini cluster was tested with benchmarks and micro-benchmarks and some performance issues were discovered. After building and comparing several parallel versions of ParaView the most efficient one for the specific system was delivered. Visualisations were performed in real time and full resolution and animations were possible for small file sizes. The only part of the project that was not completed was the OpenDAP readers for ParaView. Although this part started and progressed up to some point it was not complete due to time limitations.

7.2 Work Plan

The original work plan of the project is presented on the figure 7.1

Figure 7.1: *Original work plan*

As shown in the plan there was enough time for each part of the project. However in the first two weeks of the project the Gemini cluster had severe performance issues and it was not able to do any practical work during that time. That had an impact on my work plan and the available time for each part of the project.

7.3 Risks

An inital risk analysis was done before the start of th project as shown in figure 7.2

Many of the risks that were originally predicted had actually occurred. Both the hardware concerning risks appeared. They had the highest probability to occur and the biggest impact on the project. Right from the first week there were hardware inefficiencies and problems. These risks were resolved from the system administrator in ICHEC.

The risks concerning the ParaView software appeared as well but they had medium impact on the project. They were easily resolved with careful consideration.

7.4 Changes

The project did not change from the original idea and plan. Most of the issues were resolved so there was need for the project to change from its original form.

Risk	Likelihood	Impact	Risk Management Approach/Mitigating Actions
Hardware			
Poor performance	High	High	Identify and investigate all possible issues. Use benchmarking to monitor Network/bandwidth. Add GPU cards if necessary.
Power inefficiencies	High	High	Constant collaboration with administration team. Investigate all possible hazards and suggest solutions.
Software			
Parallel implementation in ParaView not performing properly	Med	High	Investigate and fully understand MPI implementation. Ensure communications are synchronised. Start testing with small data files that are easy to manipulate.
Animation built-in tool in ParaView not working	High	High	This is linked to the power efficiency of the cluster. Cooperation with administration team and supervisor.
Incomplete OPeNDAP reader for netCDF format	Med	Med-High	Cooperation with OPeNDAP providers. Take full advantage of the provided support.
Time availability			
Short period of time provided. 10-week project.	Med	Med-High	Efficient work plan to take full advantage of the available time. At the end of each week monitor progress.
Communication			
Not possible to fully investigate until the actual beginning of the project	Low	Low-Med	Clear plan and goals. Meeting with external supervisor and discussion at the start of the project. Meeting plan at the end of each week to discuss progress.
New working environment and communication failure	Low	Low-Med	Quick adaptation to new working environment. Setting communication with all associates from the start.
Project away from EPCC	Low	Low	Set weekly telephone meetings to discuss project with supervisor from EPCC.

Figure 7.2: *Original risk list*

Appendix A

ParaView Configuration and Built Instructions

```
1. Installation
Source code directory in Gemini:
$ cd /ichec/packages/paraview/3.14.0/source

Cmake 2.8.8 version in Gemini:
$ cd /ichec/packages/cmake/2.8.8/

It can be used as a module by typing: $ module load cmake.

Qt library 4.6.2 directory in Gemini:
$ cd /ichec/packages/paraview/qt/4.6.2/

2. Configuration

Create the target directory: $ mkdir target_directory
Go to target directory: $ cd target_directory
Load cmake module: $ module load cmake
Enable MPI: $ module load openmpi-intel/1.6
Specify intel compilers: $ module load intel-cc
Cmake with source code directory: $ ccmake source_directory

Configuration panel:
BUILD_SHARED_LIBS: ON
QT_QMAKE_EXECUTABLE: /ichec/packages/paraview/qt/4.6.2/bin/qmake
Make sure all MPI and Python flags are OFF.

For parallel configuration
```

```
PARAVIEW_ENABLE_PYTHON: ON
PARAVIEW_USE_MPI: ON

Press c for configuration
Configure until generate option is available

Press g to generate

3. Built
$ make.
To speed up the process type:
$ make -j 32.
With -j option a number of jobs can run simultaneously

When this is complete paraview is ready to use.

4. Run

Serial
$ cd /ichec/packages/paraview/3.14.0/serial

$ ./paraview

Parallel

$ cd /ichec/packages/paraview/3.14.0/cpu_mpi/bin

$ ./paraview &

To start the server: $ mpirun -np N ./pvserver

N is the number of process the server will run.
```

Appendix B

Parallel Server Batch File

```
# Set the shell - bash shell
#!/bin/bash
# Job name (default is name of pbs script file)
#PBS -N Server_Script
# Tell pbs to use XX nodes and YY processors per node
# Note: Cluster uses nodes=XX:ppn=YY
# but shared memory machine uses
# ncpus=XX to specify resources
#PBS -l nodes=2:ppn=32
# Resource limits: amount of memory and CPU time.
#PBS -l walltime=02:00:00

# Output and error files are combined
#PBS -o pview.out
#PBS -e pview.err
#PBS -j oe
# Specify server executable path
cd $PBS_O_WORKDIR
# Enable OpenMPI
module load openmpi-intel/1.6
module load intel-cc
# Command to start the server
# --bind-to-core: Bind the process with the physical core
# --use-offscreen-rendering:Necessary with OS Mesa support
# --reverse-conection: Server sends request to the client
# --server-port: Specify the port of the server. Default 11111
# --client-host: Specify the IP adress of the client machine

mpiexec --bind-to-core ./pvserver
--use-offscreen-rendering
```

```
--reverse-connection --server-port=11111
--client-host=10.100.1.38
```

Bibliography

[1] ParaView: general overview and online documentation
www.paraview.org

[2] CF-Conversions Documentation
http://cf-pcmdi.llnl.gov/documents/cf-conventions/1.6/

[3] netCDF Documentation
http://www.unidata.ucar.edu/software/netcdf

[4] Ruadhai Dervan. *3D Visualisation Techniques to Study the output of the Met Eireann Forecast Model, HARMONIE* Supervisors: Alastair McKinstry, Eoin Brazil. August 2011

[5] OpenDAP Documentation and Support
http://opendap.org/support

[6] The Irish Meteorological Service
http://www.met.ie/

[7] HARMONIE model in ICHEC
http://www.ichec.ie/research/met_eireann

[8] HARMONIE model
http://www.hirlam.org

[9] GRIB format description
http://en.wikipedia.org/wiki/GRIB

[10] Users Guide Client-Server Visualization
http://www.itk.org/Wiki/Users_Guide_Client-Server_Visualization

[11] Structure of a netCDF file
http://badc.nerc.ac.uk/help/formats/netcdf/index_cf.html

[12] netCDF tutorial
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.p

[13] THREDDS Data Server Documentation
http://www.unidata.ucar.edu/projects/THREDDS/tech/TDS.html

[14] Gemini node hardware layout
https://wiki.ichec.ie/mediawiki-staff-1.16.2/index.php/

[15] Open MPI: Open Source High Performance Computing
http://www.open-mpi.org/

[16] ParaView source code
http://www.paraview.org/paraview/resources/software.php

[17] Cmake build system
http://www.cmake.org/

[18] Cmake source code
http://www.cmake.org/cmake/resources/software.html

[19] Qt framework
http://qt.nokia.com/products

[20] Qt library sourse code
http://qt.nokia.com/downloads

[21] Mesa library
http://www.mesa3d.org/

[22] OpenGL graphics
http://www.opengl.org

[23] X Window System
http://en.wikipedia.org/wiki/X_Window_System

[24] ParaView implementation with Mesa library
http://paraview.org/Wiki/ParaView_And_Mesa_3D

[25] Mesa library download website
http://mesa3d.org/download.html

[26] Off-screen rendering Mesa interface
http://www.mesa3d.org/osmesa.html

[27] ParaView mailing lists
http://www.paraview.org/paraview/help/mailing.html