VE_Suite: Coupling visualization and computational environments to support on-the-fly engineering design

by

Song Li

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Carolina Cruz-Neira, Major Professor
Kenneth Mark Bryden
Daniel Berleant

Iowa State University

Ames, Iowa

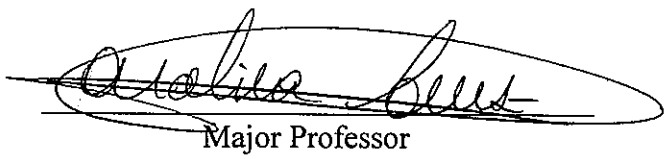
2003

Graduate College
Iowa State University

This is to certify that the master's thesis of

Song Li

has met the thesis requirements of Iowa State University



Major Professor

For the Major Program

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. Cruz-Neira for her guidance toward completion of this thesis. I would also like to thank Dr. Bryden for the interesting project he provided to me. I would also like to thank my co-workers and staff in Virtual Reality Application Center for the advice and help they provided.

Lastly, I would like to thank my parents for their support and encouragement.

This work was performed at Ames Laboratory under Contract No. W-7405-Eng-82 with the U.S. Department of Energy. The United States government has assigned the DOE report number IS-T 2209.

ABSTRACT

CFD (Computational Fluid Dynamics) is a widely used technique in engineering design field. It uses mathematical methods to simulate and predict flow characteristics in a certain physical space. Since the numerical result of CFD computation is very hard to understand, VR (virtual reality) and data visualization techniques are introduced into CFD post-processing to improve the understandability and functionality of CFD computation.

In many cases CFD datasets are very large (multi-gigabytes), and more and more interactions between user and the datasets are required. For the traditional VR application, the limitation of computing power is a major factor to prevent visualizing large dataset effectively. This thesis presents a new system designing to speed up the traditional VR application by using parallel computing and distributed computing, and the idea of using hand held device to enhance the interaction between a user and VR CFD application as well. Techniques in different research areas including scientific visualization, parallel computing, distributed computing and graphical user interface designing are used in the development of the final system. As the result, the new system can flexibly be built on heterogeneous computing environment, dramatically shorten the computation time.

CHAPTER 1 INTRODUCTION

Computational Fluid Dynamics (CFD) technology is widely used in the design of aircraft, automobiles, and power plants. CFD uses mathematical methods to simulate and predict flow characteristics in a certain physical space. The result of this mathematical computation is a dataset that typically contains vector and scalar fields in a three-dimensional space representing the flow's characteristics. Sometimes the computation results can be more complex, involving time and space-varying structures such as vortices, recirculation, and oscillation, which are very hard to understand by direct analysis of the numerical results. Thus, the visualization of these complex structures plays an important role in the CFD world, translating the numerical solutions into visual representations to make the CFD dataset comprehensive.

Traditionally CFD has been visualized in conventional desktop environments, which are inherently two-dimensional. Due to the complexity and three-dimensional nature of CFD computational results, desktop environments limit the possibilities for an in-depth analysis of the data at hand. For example, users' interactions with CFD data may require finding specific points and regions within the flow space. This task can be difficult with current desktop tools. The physical screen size of the desktop can also make the CFD data less understandable since it is hard to present a true 1:1 scale to viewers. Typically, the object shown on the screen will be significantly smaller than the real one, which also

limits users' understanding of the special properties of the data.

The use of virtual reality (VR) technology, in particular immersive VR, can significantly improve the understandability and functionality of CFD visualization applications. Immersive VR overcomes many of the desktop limitations by providing a 1:1 scale data display and direct 3D interactions with the different components of the flow. It also opens the possibilities of designing new analysis methods beyond the current desktop-based ones.

The need for immersive visualization of CFD has motivated the work presented in this thesis. Over the past two years, work has been done on several CFD immersive applications and key issues have been identified to enhance the analysis work required by CFD specialists. Based on this work, an immersive tool for CFD analysis, which is described in detail in this document, has been designed.

1.1 The Challenge of VR CFD visualization

CFD datasets are usually very large, on the order of gigabytes and terabytes depending on the simulated flow parameters. Managing these large datasets and extracting meaningful information from them is very challenging. There have been several efforts made to create CFD analysis tools in VR. However, most of the traditional VR frameworks for CFD have the following limitations:

- Application bounded to a specific operating system

Most of the CFD visualization tools are built with a specific computer platform in

mind. The visualization and all related tasks need to be compatible with a single platform. As discussed above, the CFD datasets are huge and extracting the visual representations (such as streamline, isosurface, etc.) is very time consuming. Under these circumstances, using as many computing resources as possible may be desirable, which requires the ability to handle a heterogeneous computing environment.

The financial factor also makes the need for cross-platform support important. It is desirable to run the immersive visualization on a platform with high-end graphics, such as the SGI Onyx systems. However, the computational tasks may not require a particular system, enabling us to use commodity equipment, such as Linux PC clusters.

- Lack of mechanisms to control the response time

In a VR environment, users expect a response time that matches their time frame. For example, if the user selects an area of the flow and requests a computational update with a hand gesture, he may expect a response within the next few seconds. But depending on the complexity of the calculations, the response may take more than a minute. Users can get frustrated and refuse to use the application because they do not have an understanding of their waiting time. The waiting time cannot be guaranteed, but it can be estimated. Most visualization tools do not take this into account, which can potentially create a barrier for users to accept VR as a tool to visualize time-varied CFD data.

- Single visualization program

Each process of the application has to load the same CFD datasets independently during startup time, which is not efficient.

1.2 Scope of research

Based on the discussion in the previous section and our own experiences on working with CFD specialists, this research focuses on the design and implementation of a framework for VR CFD visualization, or more commonly, for scientific visualization. Our goal is to make this framework cross-platform and high performance. This research also studies the new methods of interaction with CFD data by taking advantage of VR technology. The results of this thesis are then packaged into a VR visualization tool, VR-Suite.

To meet the research goals, this thesis is structured in the following stages:

1. Definition of CFD requirements for virtual environments

   To define the requirements we met with CFD specialists and reviewed multiple CFD applications. Among the issues that critically need to be considered: network communication ability to enable cross-platform visualization and computations; the use of parallel computing techniques to speed up the computation; and the use of the real-time system's concept to limit the waiting time.

2. Analysis of the existing VR scientific visualization systems

   Surveying and analyzing existing work allows the identification of techniques applicable to our needs and techniques that need to be developed to fully satisfy the requirements specified in the previous section.

3. Design

   Based on the requirements and the available techniques we have worked on a design to support real-time immersive CFD visualizations for analysis. Our design incorporated the connection to high-performance resources to perform and visualize

computations "on-the-fly".

4. Implementation

Several useful software tools were used in the implementation of our application. To ensure the feasibility, the simplified implementation was written first, and then we gradually extend the functionality.

Test the performance

Since parallel computing and network communication are involved in the implementation, we have included a set of performance tests to validate our implementation and to make sure our software uses network and computing resources efficiently.

5. Discuss results

From the data of performance testing, we can identify what benefits this research has brought to our system, VE-Suite, and the cost of using this framework.

CHAPTER 2 BACKGROUND

This research is built upon several widely used technologies such as virtual reality and 3D visualization, and is used by the CFD community as the post-processor of CFD data. This chapter gives brief background information about these technologies.

2.1 CFD

The goal of CFD (computational fluid dynamics) is to predict what will happen when fluids flow. A CFD problem often has the conditions of [1]:

- Chemical reaction (e.g., combustion)
- Mass transfer (e.g., perspiration)
- Mechanical movement (e.g., fan movement)
- Phase change (e.g., melting and freezing)

CFD software expresses the problem described by users as mathematical equations and gives numerical results by solving these equations. CFD can be used in many areas. For example, it can help vehicle designers achieve maximum performance at minimum cost, chemical engineers improve efficiency of reactors, and architects design more comfortable living environments.

This research focuses on the post-processing of the CFD computation's result, since the immediate result of CFD software is large and not user-friendly. The post-processing schemes are needed to make CFD results easy to interact with and understand.

2.2 Virtual Reality

In [2], Dr. Cruz-Neira provided the following definition of virtual reality (VR): "Virtual reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional, computer-generated environments and the combination of technologies required to build these environments."

This definition reflects several characteristics of VR. First, VR should provide a sense of immersion so that users will forget it is a fabricated environment and react as if they are in the real world, which the VR system portrays.

Second, a VR system should be interactive with users. Users can control the position and appearance of the virtual environment through input devices and related software. The virtual environment should also know the position of the user so that it can adjust the image and the user can have a better sense of immersion.

VR System

The VR system is a set of hardware and software components that presents virtual environments to users [3]. In accordance with the characteristics of VR, it can be divided into three parts: tracking and input devices, display devices, and the computer system.

- Tracking and input devices

A fundamental functionality of the VR system is tracking the position and orientation of the user's head so that the perspective of the image can be adjusted. The tracking system achieves this by placing a sensor on the user's stereo glasses and continuously receiving the position and orientation data from a sensor. This tracking scheme can also

be used to track the movement of the user's hand as well as the other body parts by using specific devices.

Another widely used input device, the "wand," can give users the capability to signal the VR application by pushing a button on the wand. The VR system also tracks the orientation of the wand so that users can easily control the VR application.

- Display

There are three major formats of VR display: head-mounted displays (HMD), single screen immersive projection displays, and CAVE (CAVE Automatic Virtual Environment)-like surround projection displays.

HMD

An HMD device is like a pair of glasses that puts a pair of screens in front of the user's eyes. Compared with the other two VR display techniques, HMD is easier to move, relatively inexpensive, and can provide complete visual immersion. On the other hand, HMD puts some weight on user's head, which can reduce the sense of immersion. The field of view is limited by using HMD, and another obvious disadvantage of HMD is that it can only be used by one user, so letting multiple people share the experience of the same virtual environment is quite difficult.

Single screen immersive projection displays:

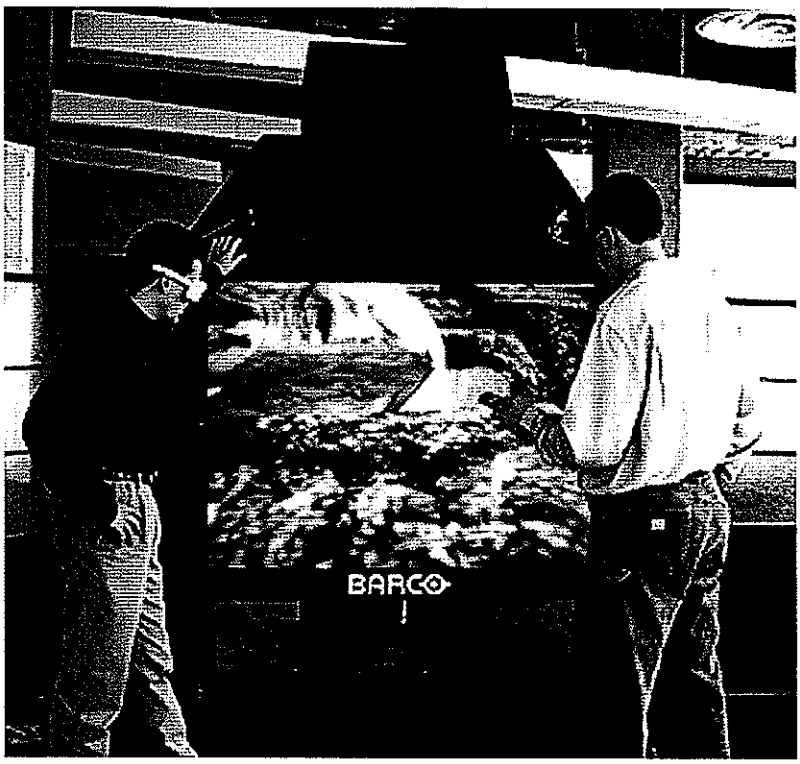


Figure 1. A single screen VR system

The single screen VR system usually has a much larger field of view than HMD, but it may stereoscopically display the objects at the edge of the screen incorrectly because in this situation one of the user's eyes receives a view of objects and the other eye does not [3].

CAVE-like surrounding projection display

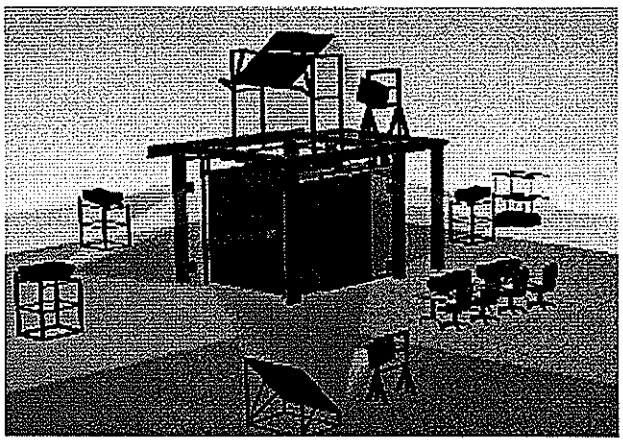


Figure 2. CAVE

The CAVE-like surrounding projection VR system has multiple adjoining screens that form a small room, and users in the room can view the stereo image surrounding them. The CAVE can provide the highest sense of immersion, but it is hard to install and adjust because multiple screens must be made to work accordingly.

- Computer system

In VR, computer systems play three roles:

- Control VR peripherals
- Generate visual objects and/or sound
- Perform computations

The critical concern of the computer system used in VR is its performance, since the

VR application continuously interacts with the users' coordinates and the movement of the wand. This requires that the computation of each frame be finished in a short and fixed period of time. If the computation is too intense and the VR system cannot handle it, users may suffer from high latency of frames. One approach to cope with this problem is to add external computing resources and let the VR system coordinate with them.

2.3 Visualization algorithm

The visualization algorithms are methods of transforming data to and from various visual representations, eventually generating graphics primitives that we can render [4]. The rest of this section briefly introduces some fundamental visualization algorithms:

2.3.1 Scalar algorithm

Scalars are single data values associated with each point and/or call of a dataset. Two of the most common scalar algorithms are:

- Color Mapping

We might want to map one scalar value to a color to make the data value displayable. In VTK, color mapping is usually implemented with a color lookup table. The lookup table uses the data value as the index. When a value needs to be displayed, the render first function searches the color lookup table for the value and picks up a color that is predefined in the table.

- Contouring

It is natural that when we see the different colors that map to different values, we tend to find the areas with same color. If the area is on a 2D surface, the 2D contour-line looks like what Figure 3 shows:

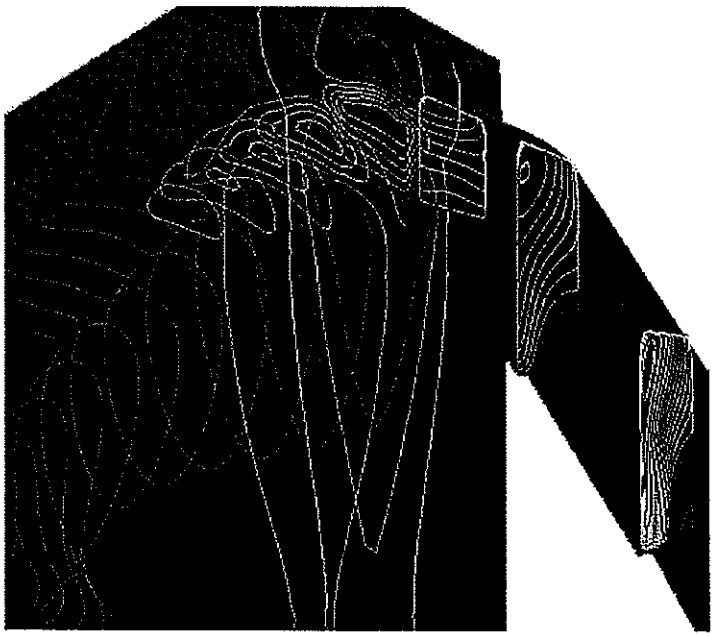


Figure 3. Contour lines of air distributor

3D contours are called isosurfaces:

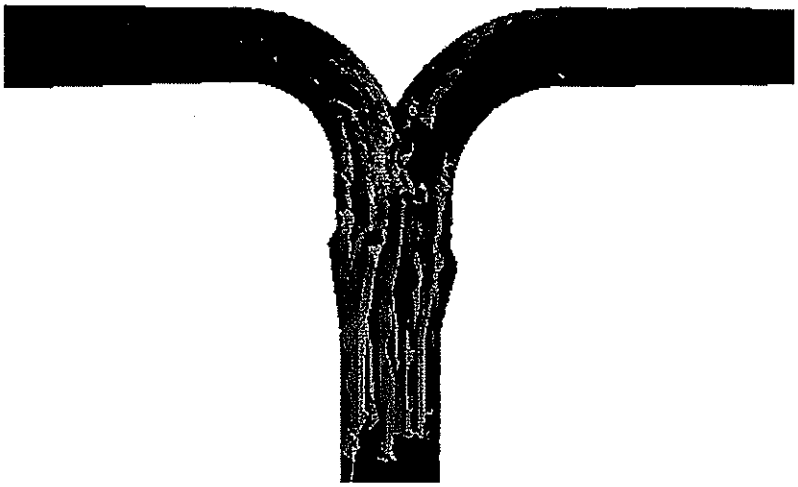


Figure 4. Isosurfaces of coal pipe

2.3.2 Vector Algorithms

In CFD visualization, we pay attention to direction in addition to magnitude, so the vector, the representation of direction and magnitude, is widely used. There are several ways to use visualization to understand vectors:

- Oriented glyphs

The most straightforward way to visualize and understand vectors is to draw an oriented arrow that represents the vector on the spot, and the color of each arrow is associated to the magnitude of this vector by using the color mapping technique.

Figure 5 shows an example of vector visualization:

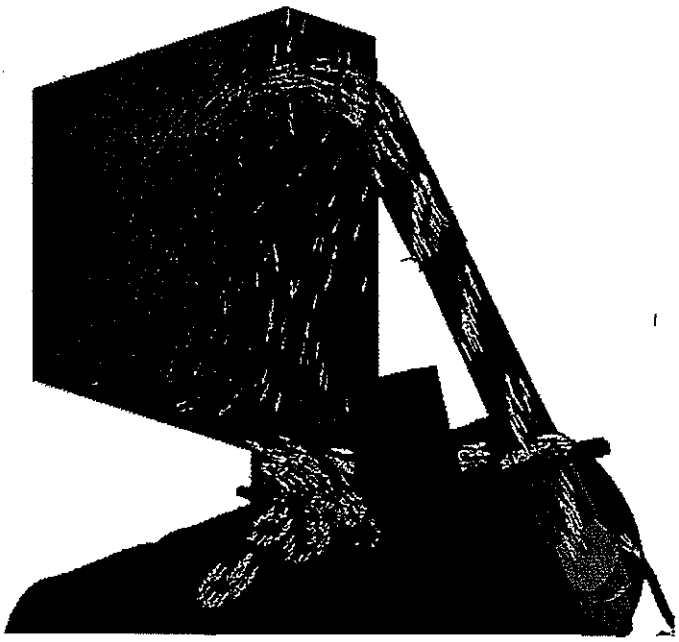


Figure 5. Vectors of air distributor

- Warping

In CFD visualization, the vector is often associated with motion. From the information provided by the vector data on a certain point, we can predict where the particle on this point will be located after a unit of time. An effective way is to "warp" or deform the geometry according to the vector field [4].

Figure 6 shows what the warped surfaces look like:

Figure 6. Warped surface of engine cylinder

- Streamline

One point of interest in CFD visualization is to display how a particle travels through space. The natural way to do that is to connect the point positions over many time steps, and in consequence, to generate a line that represents the numerical approximation of a particle trace. We can also use color mapping to color the streamline according to the magnitude on each point.

Figure 7. Streamline of air distributor

2.4 Cluster computing

There are two hardware architectures that can carry parallel computing tasks:

1. Shared-memory multi-processors(SMP) system

SMP system has more than one processor and these processors share the same memory. Figure 8 briefly shows how an SMP system is organized:

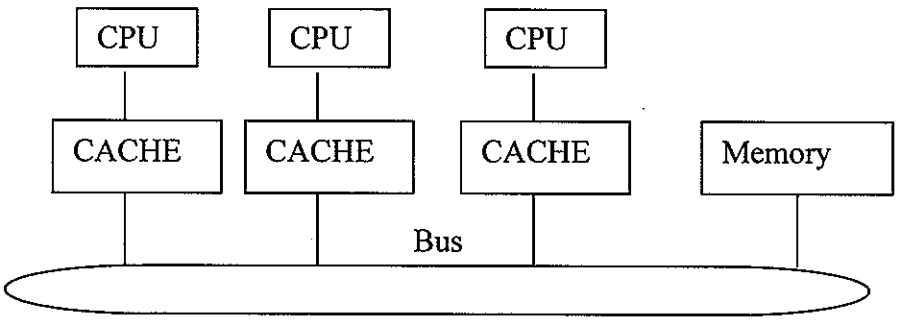


Figure 8. SMP system

2. Distributed memory system

The characteristic that distinguishes the distributed memory system from the SMP system is that it is made by several individual nodes, each of which is an independent computer with CPU, cache and memory itself. Cluster computers, which are used in this research, are concrete implementations of the distributed memory system.

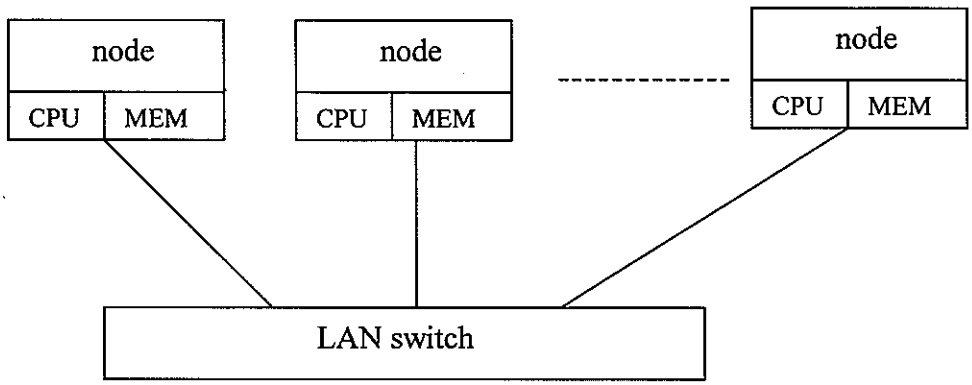


Figure 9. cluster computers

Cluster computers play an important role in this research as external computer resources of the VR system, which carries CFD visualization algorithms in parallel. We use the 18-node dual processor Linux cluster in the Virtual Reality Applications Center (VRAC) at Iowa State University as the hardware platform to run parallel code. Each of these PCs runs Linux as the operating system. Throughout the rest of this thesis, "Linux cluster" refers to this computing platform.

2.5 Previous work on CFD visualization

2.5.1 Desktop flow visualization – FAST

FAST (Flow Analysis Software Toolkit) [5], developed by the NASA Ames research center, is one of the pioneers of flow visualization. Based on the CFD dataset with Plot3D or unstructured grid format, FAST can generate most of the visual representatives such as isosurfaces, streamlines, and cutting planes.

2.5.2 Flow visualization in virtual environment – VWT

VWT (Virtual Windtunnel) [6], which was also developed by the NASA Ames research center, is a system designed to visualize unsteady flow dataset in a virtual environment. The VR hardware used for this work is BOOM (Binocular Omni Orientation Monitor) and dataglove.

2.5.3 Scientific VR visualization system using parallelism – ViSTA

In [7] the author describes recent research conducted by Aachen University of Technology, Germany, which put the computation task in scientific visualization over the distributed system and combined it with VR hardware and software. The purpose of introducing the parallelism and distributed system is to lower the cost of the computing resource, especially for large CFD datasets, and to achieve adequate performance of VR economically.

CHAPTER 3 DEVELOP TOOLKITS

Since VE-Suite is built with many software development tools and libraries, a brief introduction of these tools is necessary before introducing the design and implementation of the project. The tools can be divided into four categories:

- VR library – VR Juggler
- Scene building – Performer, VTK and translator
- Parallel computing – MPI
- Distributed computing and networking – CORBA

3.1 VR Juggler

The framework of the visualization part is based on VR Juggler, a common VR application development API developed by Iowa State University [8]. VR Juggler provides the following features to the VR application developers:

1) Operation system independence – VR Juggler isolates system-specific dependencies behind a simple virtual platform (VP) interface, implemented as a kernel [8].

Figure 10 shows the interface of VR Juggler and VR applications:

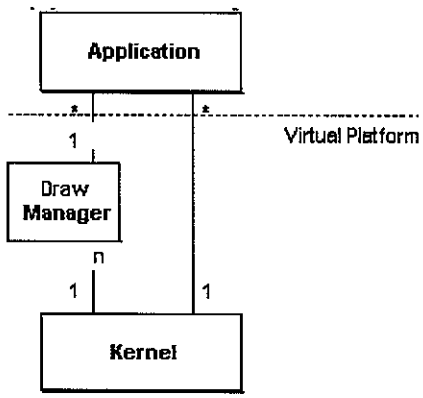


Figure 10. VP and application

2) Device abstraction – VR devices can be abstracted into several basic classes of input and output like positional, orientation, digital, analog, glove, and gesture [8].

3) Simple operating environment – allows highly specialized applications to run simultaneously[8].

4) Support multiple graphic APIs – support popular scene graph libraries such as OpenGL, Iris performer and OpenSG. Encapsulates all API-specific graphics behavior in draw manager [8].

The applications using VR Juggler are in the form of objects that inherit "vjApp" and re-implement the method in the same manner as "preFrame()" and "postFrame()."

Figure 11 shows the relationship of classes:

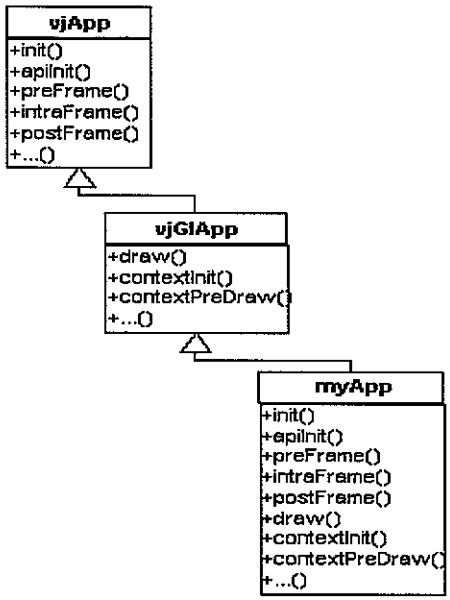


Figure 11. Class hierarchy of VR Juggler

All the VR resources are controlled by VR Juggler's kernel, and when the methods overridden in application objects such as preFrame() are called, the kernel passes control to the applications and lets them use the resource that is needed.

VR Juggler is used as a VR software development toolkit because of the following features:

- Its object-oriented architecture, which is suitable for a large, expandable application
- Its provision of easy-to-use APIs
- Its ability to be used in the VR facilities in VRAC at Iowa State University (the C6 and the C4), which is where this research is being conducted.

3.2 Scene graph toolkits – VTK, performer and translator

VR Juggler provides a framework for the VR visualization program, but it does not provide an application programming interface (API) to handle graphical tasks. Applications need to integrate a graphics API or scene graph manager to handle visualization tasks.

Among a variety of graphics APIs and scenegraphs available we decided to use Performer™, which is a high-performance, object-oriented scene graph library developed by SGI [9]. Performer™ was chosen based on the following reasons:

1) Its object-oriented architecture fits the VR Juggler.
2) Its high performance when running on SGI's hardware, which is the main hardware platform of the visualization program.
3) The availability of translator with VTK, which is discussed in the next section.

VTK

All the scientific visualization algorithms such as streamlines, surfaces, etc. described in Chapter 2 are implemented by VTK (Visualization ToolKit [10]), which is a set of software libraries that implements a specific visualization algorithm on the input dataset and generates the output dataset which is in the format of vtkActor, a class defined internally by VTK.

VTK can finally render vtkActor on a desktop, while for a VR Juggler application, we need to translate vtkActor to Performer's object so that VR Juggler can recognize and render it. This job is done by vtkActorToPF, a freeware package developed by Paul

Rajlich in NCSA [11]. This small tool takes vtkActor as input and outputs Performer's pfGeode or pfGeoSet, which can be used by VR Juggler.

3.3 MPI

The message-passing interface (MPI) is a widely used parallel computing standard [12]. The basic idea of MPI is running the same code on different computers/processors simultaneously and letting them coordinate with each other by passing messages. The MPI system will assign an integer called "rank" to each process so that it can be identified. In the computing environment of a Linux cluster where each node has independent address space, and the controlling of the computing process cannot be executed internally, the message passing scheme is the best way to perform parallelism.

MPICH [13], A software library implementing the MPI standard, is used in our project to perform parallel computing.

3.4 CORBA

CORBA (Common Object Request Broker Architecture) is an open distributed object computing infrastructure standardized by the Object Management Group (OMG) [14]. It is used to enable communication among different computing resources in VE-Suite.

Figure 12 shows the architecture of CORBA:

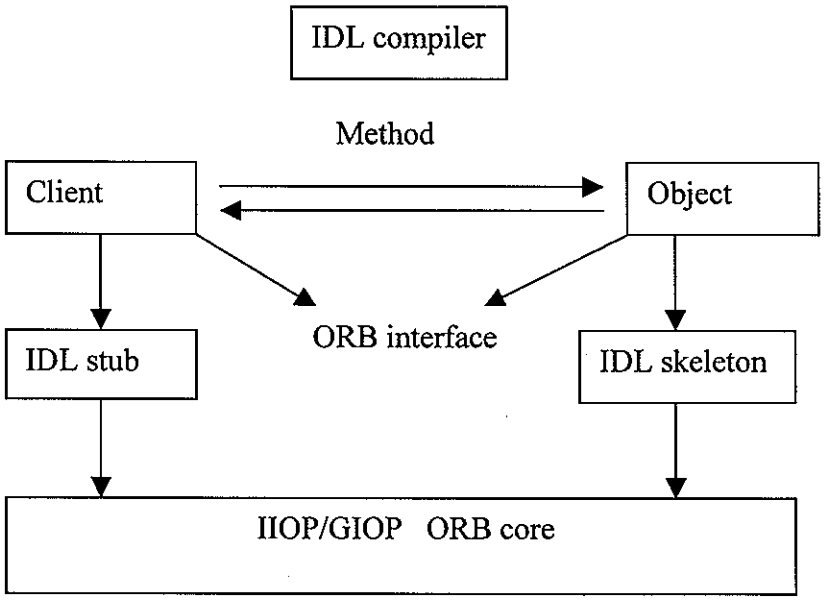


Figure 12. CORBA architecture

Briefly stated, CORBA uses a common interface description language (IDL) to specify the interface between client and server and uses an IDL compiler to generate the code used for each side, which are noted as "IDL stub" and "IDL skeleton" in Figure 12.

Figure 13 describes this process:

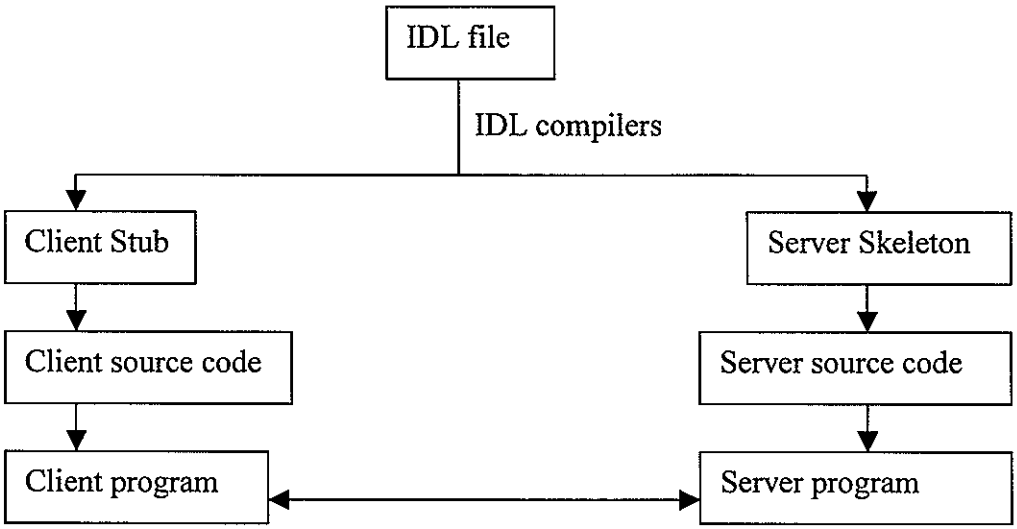


Figure 13. Process the IDL file

A single copy of the IDL file can be compiled as two different kinds of source code, which can be in the format of different languages. As shown in Figure 13, the IDL-to-Java compiler can be used to generate java code on the client side, and IDL-to-C++ compiler can be used to generate C++ code on the server side. As Figure 12 shows, CORBA uses GIOP/IIOP to make the client and the server communicate with each other without consideration of the difference of platform and language.

We use CORBA in this research because it provides very user-friendly interfaces to developers to call methods on remote machines and our research on improving the performance of a CFD VR application heavily depends it.

In the implementation of VE_Suite, we use a software library implementing CORBA standard called OmniORB [15].

CHAPTER 4 DESIGN OF VE-SUITE

4.1 Overview

The main goals of this research are to design a platform-independent software system that interactively visualizes CFD numerical results in a VR system and to be able to use as many computing resources as needed. A natural solution is to adopt client/server architecture. We need to divide the application into two parts: the server takes the time-consuming calculation tasks, and the client acquires the results from the server and renders them in the virtual environment. We can also take advantage of this architecture by running the server on a Linux cluster, which is inexpensive and flexible, and running the client on an SGI machine to utilize its high graphical performance.

Figure 14 describes a conceptual overview of the system:

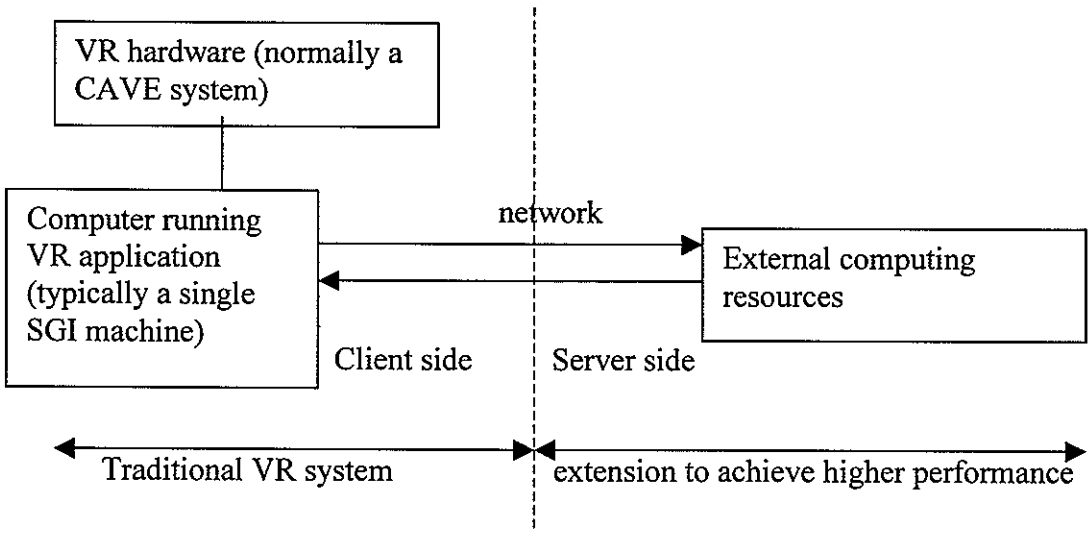


Figure 14. VE_Suite System overview

In this research the computing tasks on the server side are running on the Linux cluster. They should be executed parallel to each other to fully utilize the computing resources of all nodes. Based on different application, task parallelism and data parallelism have been implemented. The details will be presented in the next section.

There are two kinds of communications between client and server: the client sends a request to the server and the server sends the resulting data back to the client. The similar communications exist within the nodes of the server's cluster. We assign one node as the gateway of the whole cluster, this node acts as a client while all other nodes are servers. So conceptually, there is one command channel from the client to each node, and one data channel from the nodes to the client.

We also need to consider the user interaction with the virtual environment. With the growth of the functionalities of VR software, the limitations of the traditional interaction devices such as the wand and the punch glove become more obvious. The reason basically is that the wand and the glove are not programmable, so all the interaction options must be implemented inside the virtual environment, which introduces complexity for developers. For this reason, besides using the wand, this software system also uses a PDA, which connects into wireless network as an input device, which gives developers the option of making the menu system independent of the VR application. The advantages of doing this are, first, it makes the VR application simple, especially when the menu is very complex; and second, it makes the menu system portable and allows it to meet different requirements because the program running on the PDA can also run on various desktops and perform the same functionality.

More user interaction features in addition to menu selection should also be considered:

- A simple designing tool. The goal is to let engineering designers input parameters that could affect the CFD calculation result. These parameters are normally obtained from geometries. So the client software running on the PDA should provide a drawing panel to the users, generate parameters from the geometries they draw and send the parameters to the programs that use them.
- Multiple-server selectable. We need to consider the situation that there are multiple computing resources available, but in the mean time they are shared by different users. Before sending jobs to the server, client software should take a poll of all known computers to gather information about the availability of each computer and let the user choose one.

Figure 15 illustrates a more detailed overview of the whole system:

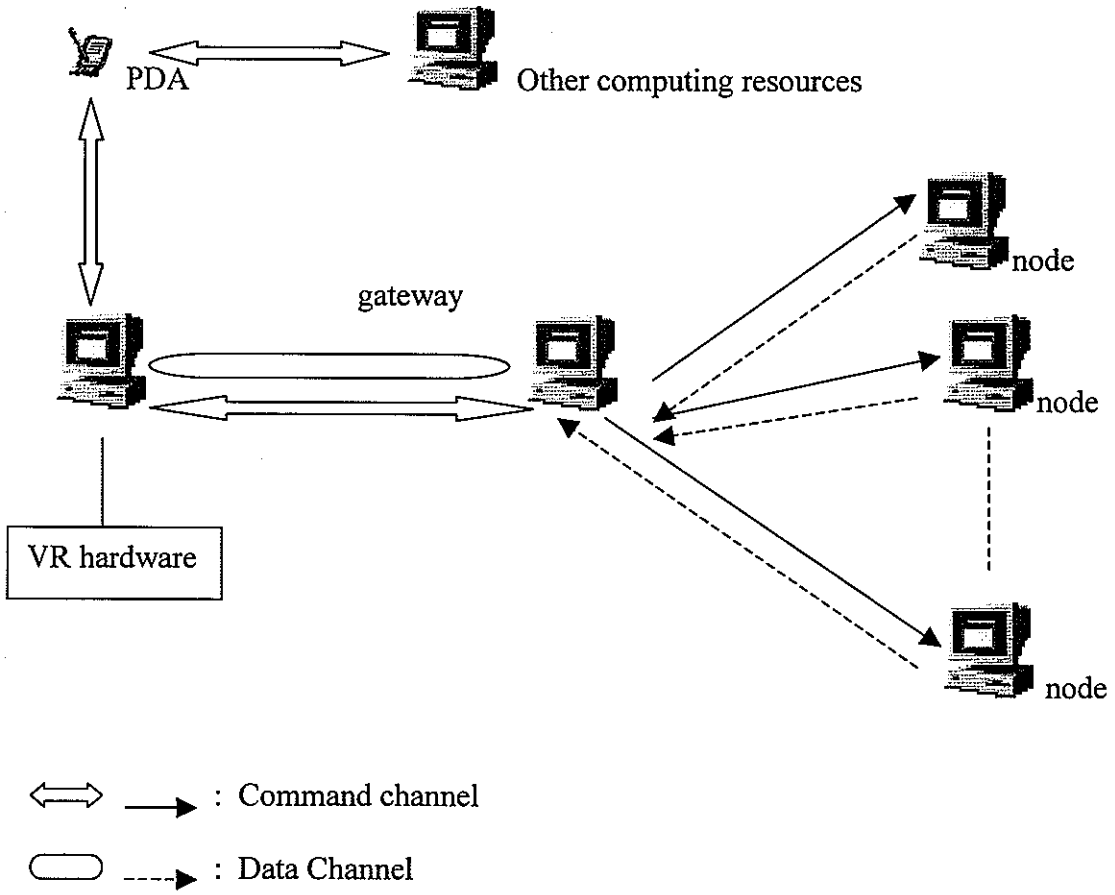


Figure 15. VE_Suite system overview (2)

4.2 Parallelism

In [16], three schemes of parallelism are introduced: task, pipeline, and data parallelism. In this section the characteristics of CFD VR application are analyzed and the appropriate parallelism scheme to use is chosen.

All the CFD dataset visualization functions can be categorized into three types:

1) Visualize pre-calculated data: The visualization results are pre-calculated and stored as files on a disk. The program's job is reading those files and

rendering them. Since no actual calculation is needed in the run-time, the speed of these kinds of functions is very fast, and parallelism schemes cannot help a lot to speed up the application.

2) Visualize data independent tasks: The vector algorithms described in Chapter 2 are normally data independent. The examples are isosurfaces, cutting planes, etc. If we have different tasks process different parts of the dataset simultaneously, the result of each task will not affect the result of others. For these functions, we can use data parallelism; that is, we can cut the whole dataset to several small pieces and let multiple computers process them simultaneously.

Figure 16 shows how data parallelism works.

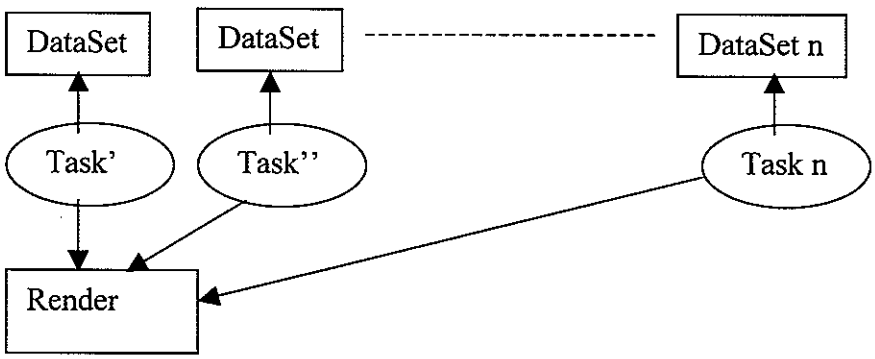


Figure 16. Data Parallelism

Visualize data dependent tasks: Streamlining is such a function. Calculation of every point on a streamline is based on the information of the previous point. This feature means that a streamline cannot be calculated in parallel in the point-to-point level; thus it is data dependent and data parallelism is not able to apply to this kind of function.

However, it is more common to show multiple streamlines than a single streamline. An example of visualizing 9 streamlines is shown in Figure 17:

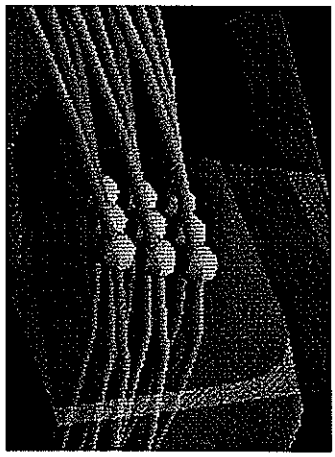


Figure 17. Multi-streamlines

In VTK, every streamline is integrated forward and backward; thus two tasks are required to visualize a single streamline. In the case shown in Figure 17, 9 streamlines need 18 tasks, which gives us enough tasks to perform task parallelism.

Figure 18 shows how task parallelism works:

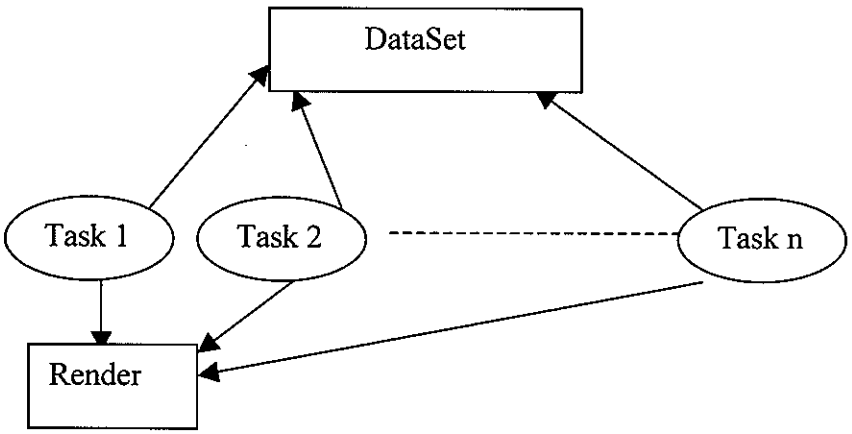


Figure 18. Task Parallelism

As shown above, task parallelism occurs when each node executes independent modules, but they do not necessarily use different datasets. If we divide the whole dataset into several sub-datasets, and let each task use a different one, data parallelism would occur.

4.3 Networking between client and server

4.3.1 Command channel

The information the user needs to tell the system can be divided into three parts:

1) Which menu was selected; this can be represented by a menu id number
2) Necessary parameters, such as the length of the streamline
3) The current state inside the virtual environment, such as the wand's position and orientation

As shown in Figure 19, this information is in the command channel flow in the sequence of:

1) User interface (PDA or other VR input devices such as the wand)
2) Computer running the VR Juggler application
3) Gateway of all nodes in the cluster
4) The individual node that uses VTK to calculate the results.

This channel is conceptual. We use different techniques on different parts. From 1-3, the information is transferred by CORBA, as Figure19 illustrates:

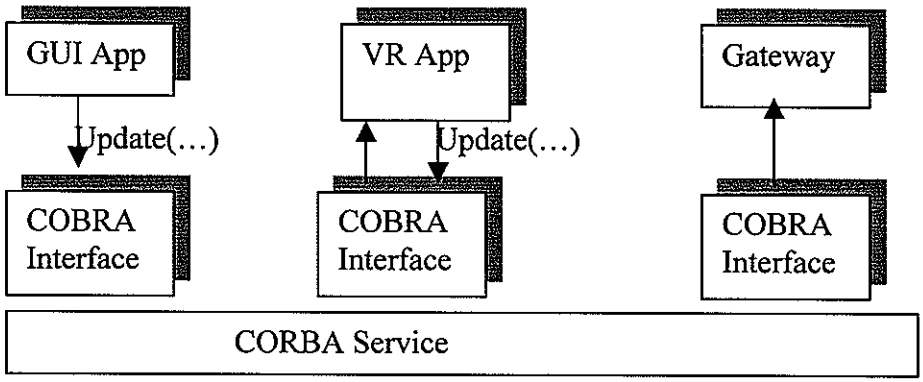


Figure19. Command channel using CORBA

The CORBA Interface in the figure forms a layer between the application's need to send a command and the real CORBA service. It contains all the routines of CORBA communication needed, so that the upper level application can simply call the function "Update" and put all the information as the parameters of this function.

From 3-4, we use MPI to continue the command channel. The reason is that the number of nodes participating in the task execution could be varied, and MPI provides an easy way to set up the group of nodes. The gateway node replicates the information received from CORBA and sends a copy to each node if it is needed.

4.3.2 Data channel

Data channel works in the reverse direction of command channel. Once the result is generated on each node, it is sent back to the gateway node. Since the result is in the format of a VTK object, the most appropriate technique to use on a data channel is

VTK's built-in data communicator which uses an IP socket. This communicator can serialize a VTK object before sending it, which hides the complexity from the developers.

The gateway node appends the received VTK objects into a single object, sends it to the VR Juggler application running on the other machine, and finally visualizes it.

In this architecture, in the view of VR application, the gateway node acts as a representative of the whole cluster and hides the parallelism scheme behind it. While in the view of each node, the gateway node is a representative of the VR application, it sends commands to nodes and each node does the task on its own dataset and lets the gateway node take the completed result.

CHAPTER 5 IMPLEMENTATION

This chapter introduces the implementation of the new functionalities described in Chapter 4. It can be divided to three parts:

1. Graphical user interface
2. Using CORBA to call functions on a remote computer
3. Implementation of parallel computing

5.1 Graphical user interface

User interface (UI) is the means by which users can interact with a software system. We want to let users have the ability to interact with VE-Suite and to control the behavior of the system on any computer they feel is convenient, so one consideration of choosing the way to implement the user interface outside the virtual environment is whether it will run on different operating system, including Microsoft® Windows, different kinds of Unix systems and possibly specific operating systems for PDAs. To achieve platform independency, we use Java Swing to implement the graphical interface, since the execution environment is relatively easy to build for Java.

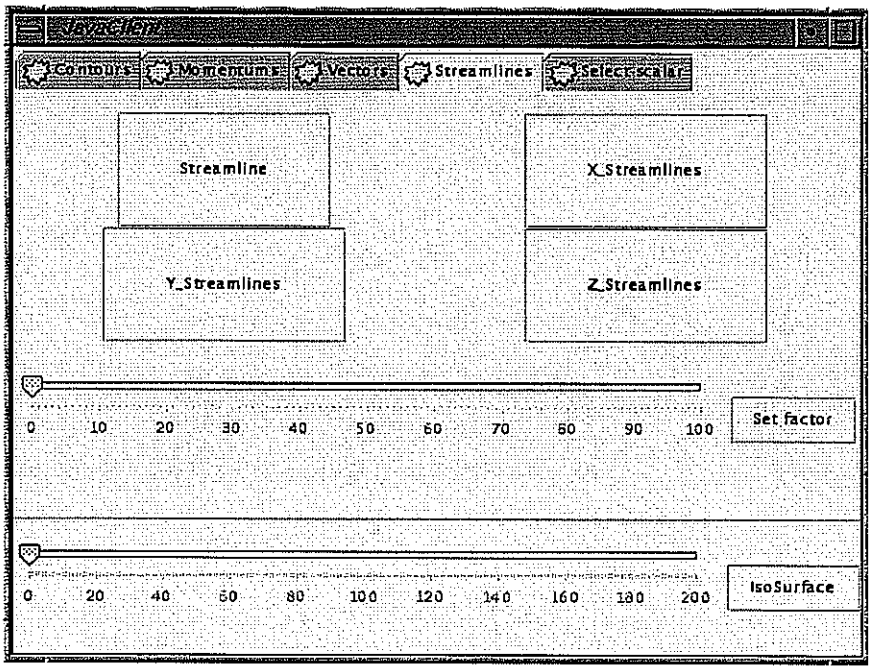


Figure 20. Appearance of GUI

Figure 20 shows a typical interface window. We use the sliders to specify a parameter and use a button to confirm an action. When a button is hit, the ID associated with this menu item and the current parameter values are sent to the VR application via CORBA.

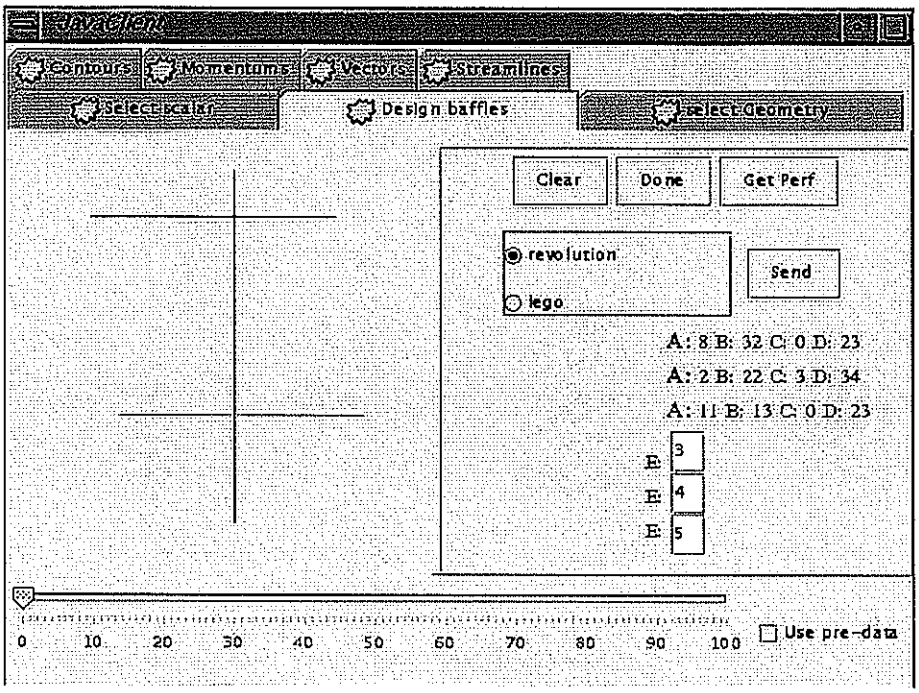


Figure 21. Appearance of designing page

Figure 21 shows what the designing page looks like. The left part of the window is a drawing panel, and the right part of the window shows all the parameters generated based on the geometries on the left. The user might push the "Get Perf" button to find out how much of the computing resources have been used on the select host. A pop-up window might return this information:

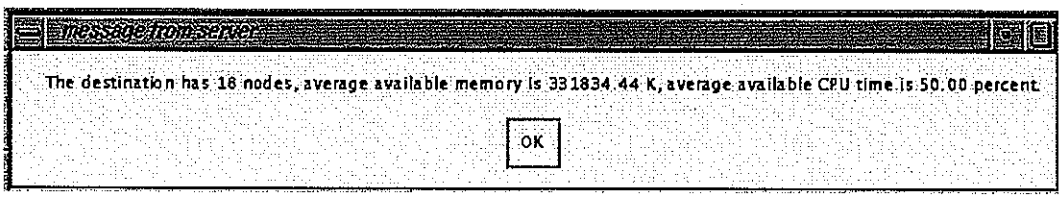


Figure 22. Performance report

Once the user pushes the "Send" button, the parameters will be sent to the selected computer, and an acknowledgement message will show up once the job on the server is done:

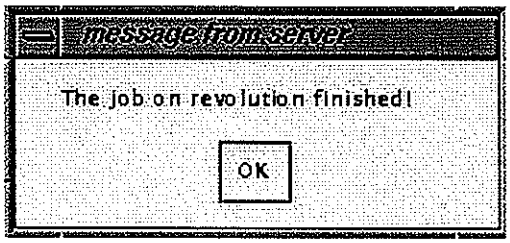


Figure 23. Acknowledgement window

5.2 CORBA communication

We use the procedure of the client talking to the VR application as an example to introduce how CORBA communication is implemented. The parameter transfer between the Java GUI and the VR application is a two-step communication with the role of client and server switching between steps.

Step 1: Java GUI works as client and VR application works as server, as shown below:

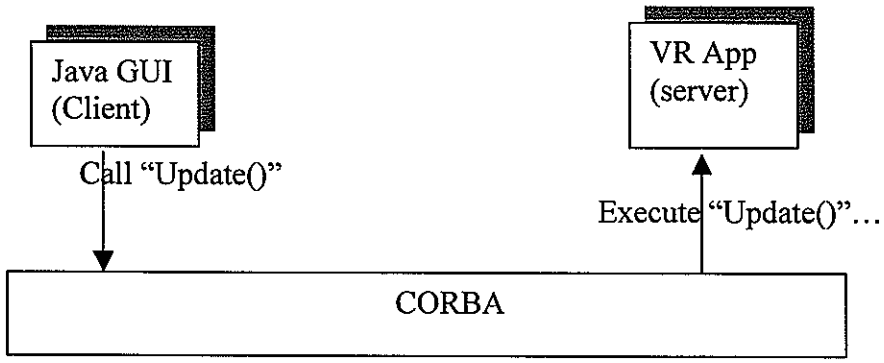


Figure 24. Step 1 of communication

The IDL file would look like:

```
interface VjObs
{
      void attach(in Observer o);
      void detach(in Observer o);
      void update();
};
```

When the client is brought up, it calls the remote method "attach" to let the server establish an object reference of client. When the button is hit, the client calls the remote method "Update." At this time, the roles of client and server have been switched, since the content of "Update" is to call the methods of a previously established object reference.

There is another IDL file with the content below which defines the methods that the VR application should call back:

```
interface Observer
{
      long update();
   long get_iso_value();
      long get_sc();
   ......
};
```

Figure 25 shows the operation:

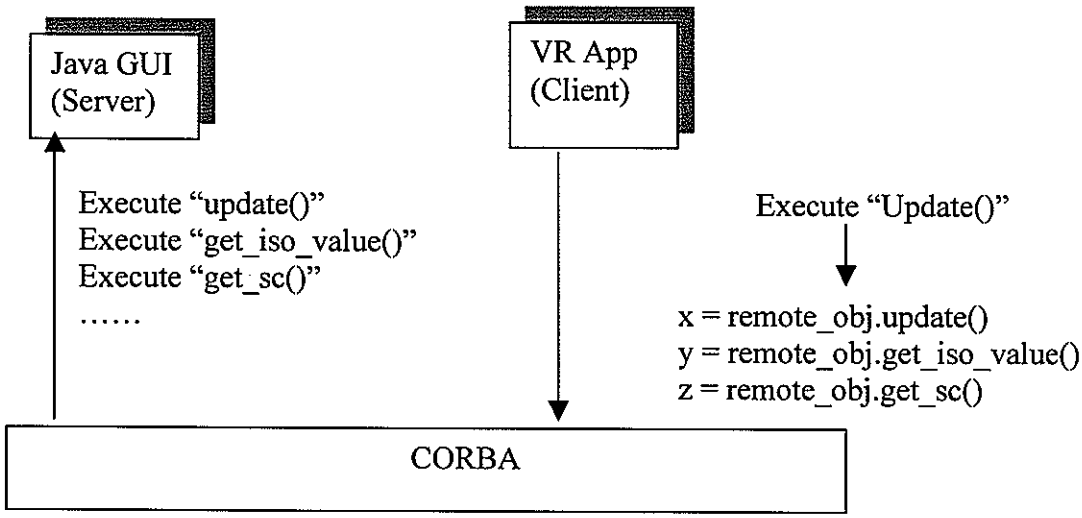


Figure 25. Step 2 of communication

The VR application becomes the client and by calling the individual remote method gets the data needed from the Java GUI. This 2-step method is suitable for the situation when multiple GUIs need to connect to the VR application.

The communication between the VR application and the gateway node should always be one-way, so we use a simpler way to implement it. This time the IDL file looks like:

```
interface Observ
{
boolean update(in short id, in float x0, in float x1, in float x2, in
float n1, in float n2, in float n3, in short t, in long iso);
};
```

The operation is very similar to the one shown in Figure 25: the VR application is the client and the gateway node is the server. All the data that the client wants to pass to the server is in the form of parameters of interface function "update."

5.3 Parallelism

Once the gateway node receives all the parameters, it sends a copy to each node and waits for the calculation result to come back from the nodes. We use MPI's pack and unpack functionalities to transfer multiple parameters. Figure 26 shows how the message and the data are passed among the nodes:

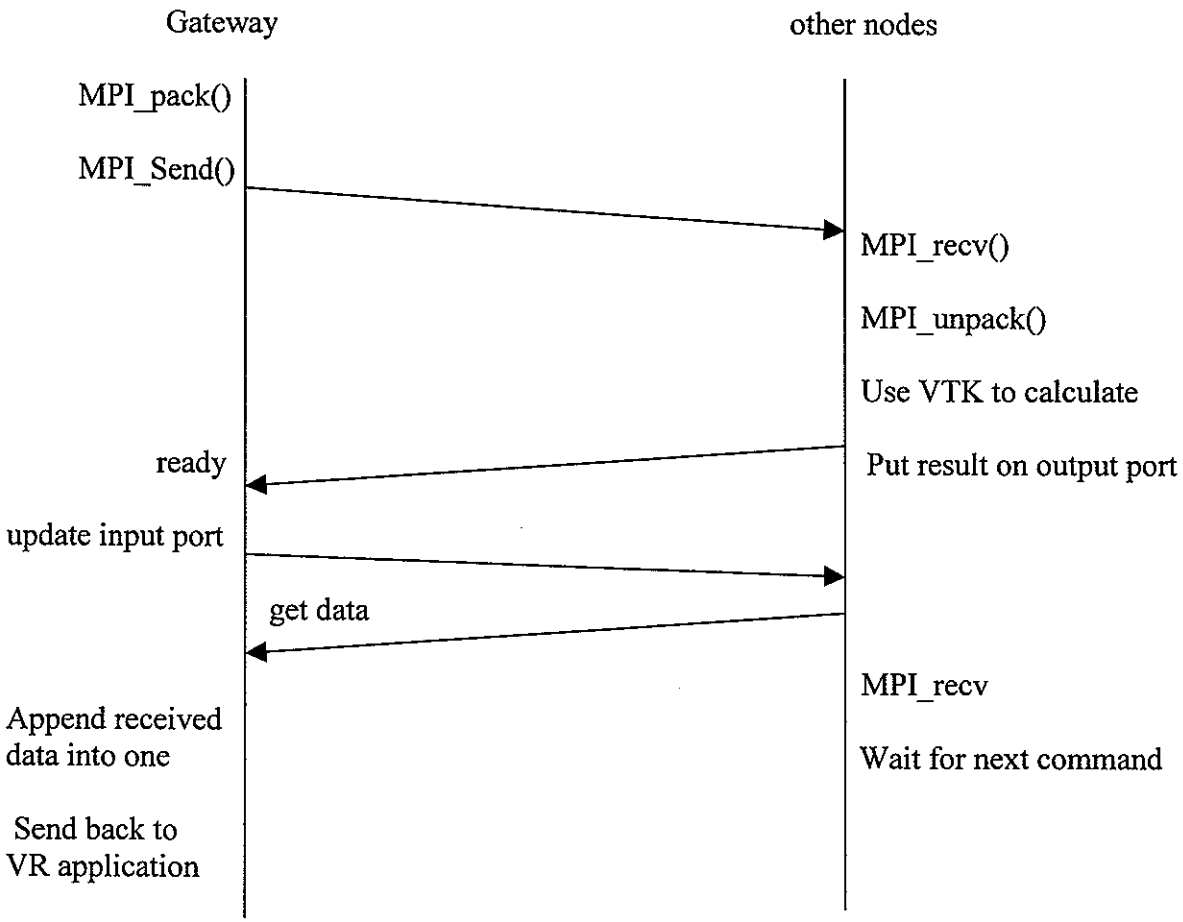


Figure 26. Communication among nodes

This figure shows one-to-one communication between the gateway and one of the other nodes. In the real code, the gateway should figure out how many nodes are required

and every operation on the gateway side is a for-loop with the number of nodes as the iteration number.

CHAPTER 6 RESULTS AND DISCUSSION

This research focuses on how to provide VR CFD application an easy-to-use user interface for mechanical design engineers and to shorten the waiting time without significant investigation on computing hardware. The result of the user interface research has been presented in Chapter 5; this chapter presents the performance issues related to the integration of all the CFD immersive visualization components.

6.1 Test scheme

We will perform the same VR visualization functionality in different ways and compare the differences.

Test Conditions

Condition 1: We run the VR application on a single computer, which has 24 SGI R12000 processors and 12G memory, with the IRIX 6.5 operating system. In this test case, the VTK algorithms are executed on a single processor serially and the VR application threads run on the other processors and waits for the VTK results. We refer to this test case as "serial method" in the rest of this chapter.

Condition 2: In this case, we run the parallel application as described in Chapters 4 and 5. We use the same computer that we used in the serial method as the VR visualization machine, and put the VTK algorithms on a Linux

cluster whose node has double 1.2G AMD athlon™ processors and 1G memory. We refer to this test case as "parallel method" in the rest of this chapter.

Test data

The visualization functionality used for this test is isosurfaces. The reason for choosing isosurfaces is that they need to compute the whole dataset and are normally the most time-consuming applications in CFD visualization.

To illustrate the relationship between the size of the dataset and the efficiency of VE-Suite, we conduct the test on 6 datasets:

Dataset	Number of Cells
Dataset 1	52805
Dataset 2	171834
Dataset 3	222562
Dataset 4	327848
Dataset 5	562234
Dataset 6	5154463

Table 1. Size of datasets

Metrics

- Using the serial method or the parallel method. The serial program only runs on an SGI machine, while the parallel program runs on both SGI and Linux machines (as described in Chapter 5).

- Dataset used for testing. Size and other features of datasets may impact the computation time.
- If the parallel method is used, the number of nodes of the Linux cluster to be used is another metric. Basically each dataset has been tested by using two node numbers. For example, dataset 6 was tested by using 8 nodes and 18 nodes. Since the computation time depends on the distribution of scalar value in dataset, the number of nodes is only meaningful with the same dataset, so the node numbers vary among datasets.

6.2 Results

The tables below illustrate the performance of visualizing isosurfaces on different datasets by using different methods. When using the parallel method, the number of nodes will be specified. For example, D1-14n means dataset 1 using 14 nodes. Each method samples 10 times with a value relative to the scalar range of the whole dataset. For example, if the scalar range is from 100 to 200, "10%" means using 100+10%*(200-100) = 110 as the value to show the isosurface. The unit of data is second.

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D1-serial	**30.7267**	4.60107	0.516513	0.206918	0.110543	0.0726659	0.066491	0.0575	0.059	0.059
D1-14n	**4.72787**	3.55413	3.24719	3.24829	3.18387	3.26292	3.14235	3.15895	3.15609	3.29313
D1-6n	**3.28387**	2.39747	2.02382	1.94596	1.91625	1.92168	1.90534	1.94488	1.9589	1.99818

Table 2. Result of Dataset 1

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D2-serial	0.586116	**2.39146**	2.22161	1.25024	0.936345	0.809715	0.769993	0.758271	0.421494	0.232205
D2-8n	2.57435	**2.42352**	2.41747	2.32112	2.1791	2.14407	2.25032	2.22852	2.24413	2.20609

Table 3. Result of Dataset 2

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D3-serial	0.305894	**7.81915**	0.513509	0.337849	0.308349	0.295086	0.285245	0.28442	0.283953	0.272268
D3-15n	5.05821	**5.67355**	3.37331	3.34737	3.29762	3.34562	3.42707	3.41143	3.38132	3.38254
D3-8n	3.41595	**4.76198**	2.25114	2.28833	2.25622	2.2612	2.28075	2.27077	2.32028	2.32265

Table 4. Result of Dataset 3

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D4-serial	16.4725	**20.5867**	12.7035	7.24117	3.79828	1.39388	0.915596	0.751317	0.609767	0.482974
D4-16n	5.53829	**5.72969**	4.67471	4.02454	3.76443	3.58848	3.48467	3.44507	3.48921	3.52155
D4-11n	5.44797	**5.93058**	4.7292	3.76605	3.26115	2.92462	2.88409	2.84893	2.80122	2.76955

Table 5. Result of Dataset 4

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D5-serial	1.68949	**19.4419**	18.2627	17.5164	16.7918	16.6799	16.5859	16.4317	14.8791	7.98304
D5-15n	7.29813	**7.09208**	6.80555	6.90701	6.74972	6.95281	6.43488	6.30072	5.86917	5.41047
D5-8n	6.12926	**6.00217**	5.83362	5.7119	5.49424	5.49848	5.41511	5.27502	4.98122	4.59071

Table 6. Result of Dataset 5

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
D6-serial	**350.88**	275.205	248.861	232.296	186.972	151.999	126.659	97.6477	68.8388	57.0035
D6-18n	**44.5772**	34.6117	31.1722	29.4762	24.8443	22.129	19.9272	16.5446	13.1427	11.4068
D6-8n	**53.838**	30.9279	19.5295	11.9714	8.29421	5.3158	3.36362	2.82762	2.84626	2.86095

Table 7. Result of Dataset 6

6.3 Discussion

Overall performance

The reason to sample 10 times with different isosurfaces values for one dataset is the unequal distribution of scalar values in a specific dataset. This distribution is unpredictable so we cannot simply specify an isosurfaces value and expect the operation of extracting isosufaces using this value to reflect the feature of the dataset. For example, if half of the total points in a dataset have the scalar value of 20% of the scalar range, while there are nearly no points with the scalar value of 80% of the scalar range, the operation with 20% of scalar value obviously can better represent this dataset, and we can only know this fact after comparing the results of all the executions. Based on this reasoning, from all the data presented in section 6.2, we choose the longest computation time that represents the worst-case scenario from 10 samplings and the correspondent best performance of the parallel method. These longest computation times are marked as bold characters in the tables above. Furthermore, if t1 is the time taken by the serial method and t2 is the time taken by the parallel method, we compute the improvement of the parallel method by:

Improvement = (t1-t2)/t1

Table 8 show this information:

	serial (s)	parallel (s)	improvement
D1	30.7267	3.28387	89.31%
D2	2.39146	2.42352	-1.34%
D3	7.81915	4.76198	39.10%
D4	20.5867	5.72969	72.17%
D5	19.4419	6.00217	69.13%
D6	350.88	44.5772	87.30%

Table 8. Overall performance improvement

For most cases in Table 8, the parallel method greatly improves the response time. Especially for dataset 6, the response time of 350 seconds yielded by the serial method is not endurable for users. By using the parallel method, the response time can be reduced to be within a tolerable range. The results of dataset 2 through dataset 6 show a possible trend that the parallel method yields better performance improvement on bigger datasets. More tests on different datasets are needed to support this conclusion. As a result, the parallel method is not a good choice for a dataset that is not big enough. Dataset 2 is such an example; it cannot obtain any performance improvement by parallelism.

Dataset 1 is an exception regarding its size. The reason for the computation time by serial method is that the dataset's shape is quite complex (refer to Figure 4 in Chapter 2). The parallel method cut the whole dataset to several small datasets. This operation not only reduced the size of dataset but also simplified its shape, as the result achieves significant performance improvement.

The factor of the data size:

The following diagrams illustrate all 10 samplings of two typical datasets. In Figure 27, dataset 4 can be considered as a typical small dataset. The computation time for the

parallel method has relatively little change compared to the change of the serial method because in this scenario the overhead of MPI and IP communication is the major part of the total time, and they are constant regardless of the size of the data. As a result, parallelism is not a good solution for a dataset with relatively short computation time.

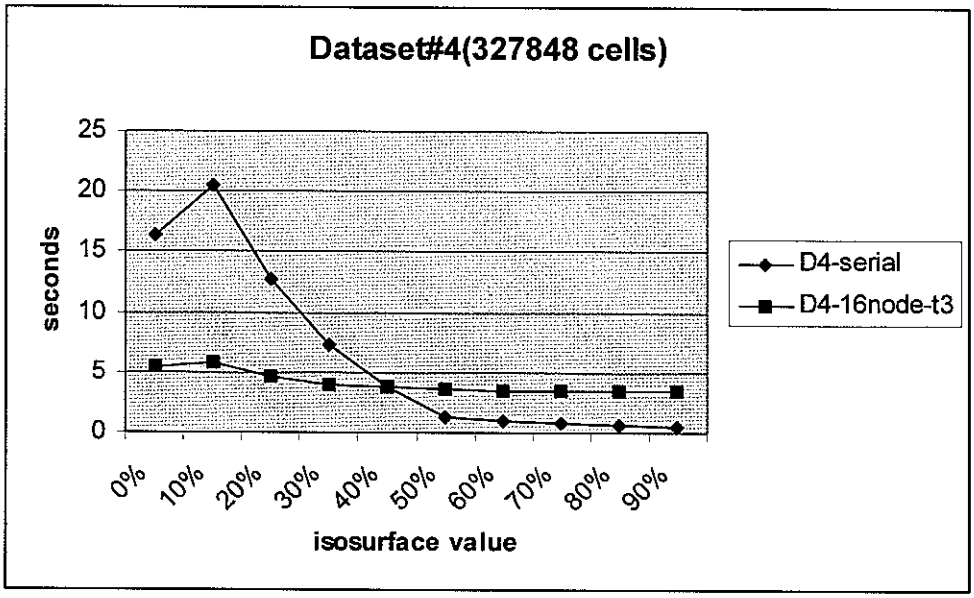


Figure 27. Performance (small dataset)

In Figure 28 dataset 6 represents the large dataset. The parallel method always outperforms the serial method since the time spent on computation of the dataset is a major part of the total time.

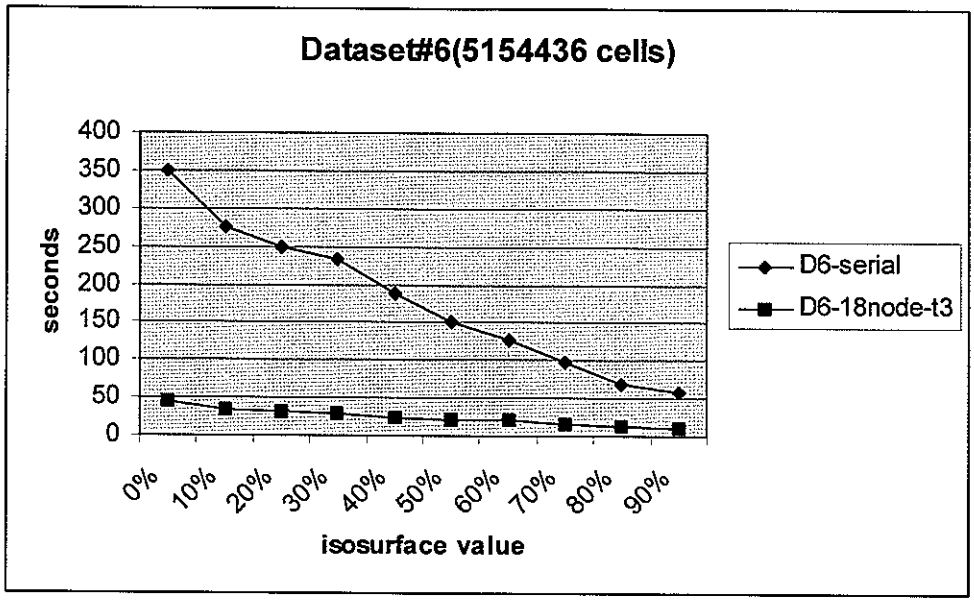


Figure 28. Performance (large dataset)

The factor of the number of nodes:

It is obvious that using more nodes in the parallel method cannot guarantee the improvement of performance due to more overhead of MPI and more data transmission over the network. Figure 29 shows the comparison of using 18 nodes and using 8 nodes on dataset 6. When it takes more time to compute (first point), using 18 nodes yields a better performance. While the response time gets shorter, 8 nodes becomes the better choice.

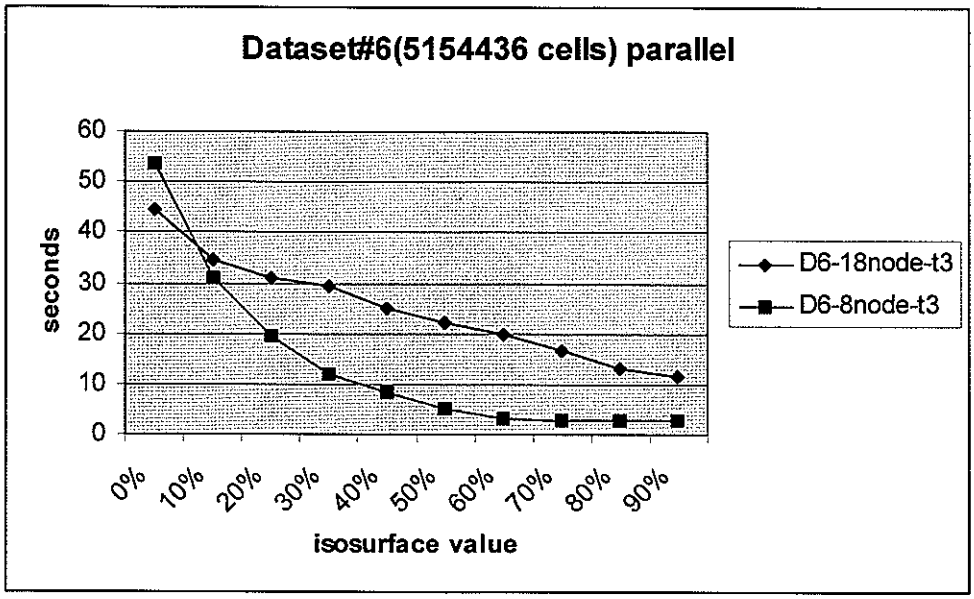


Figure 29. 18 nodes vs. 8 nodes on dataset 6

Another typical situation is for a small dataset, shown in Figure 30: fewer nodes (6 nodes) always yields a better performance.

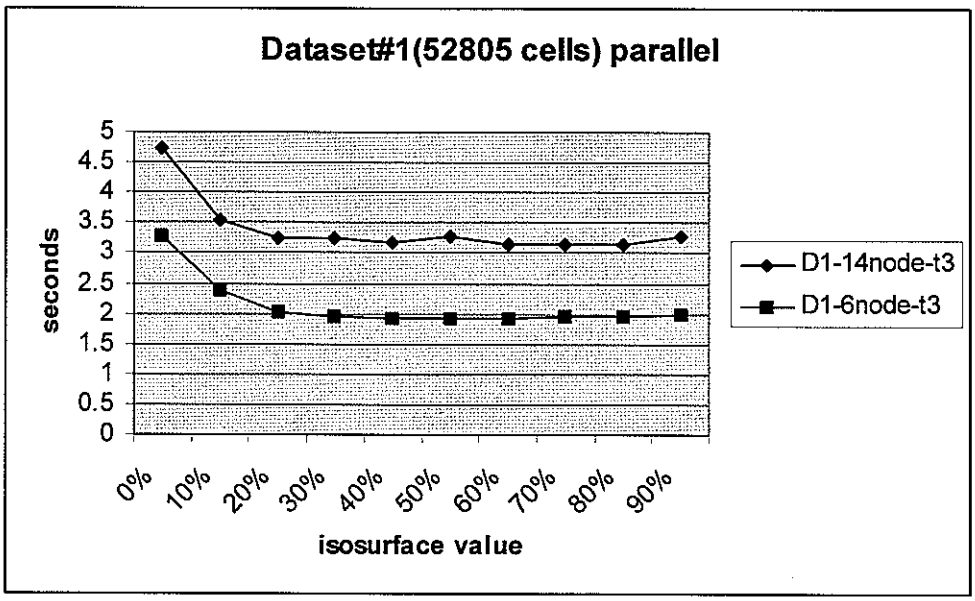


Figure 30. 14 nodes vs. 6 nodes on dataset 1

The factor of network bandwidth between cluster and visualization workstation:

First suppose the whole procedure can be divided into 3 stages as in Figure 31:

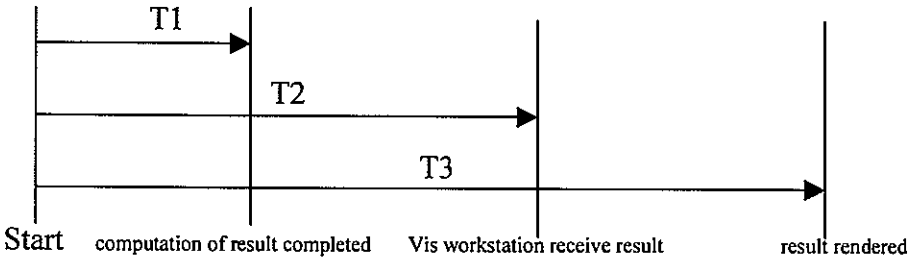


Figure 31. Definition of t1, t2 and t3

We can divide the whole execution time of the parallel method into 3 parts:

- Time for generating a result on the cluster, denoted as t1
- Time for transmitting a result from the cluster to the visualization workstation, denoted as t2-t1,
- Time for rendering the result, denoted as t3-t2

Both the serial and the parallel methods have the first and third parts. There are MPI overhead and data transmission time in t1, but they can be considered as an inevitable part of parallelism.

The second part, which is the time used to transmit the computed result from the cluster to the visualization workstation is an extra part for the parallel method. To measure the impact of this part on the overall performance, we define the ratio of result transmission time to computing time as:

Ratio = (t2-t1)/t1

Again dataset 6 is a typical example to illustrate this ratio, as shown in Figure 32:

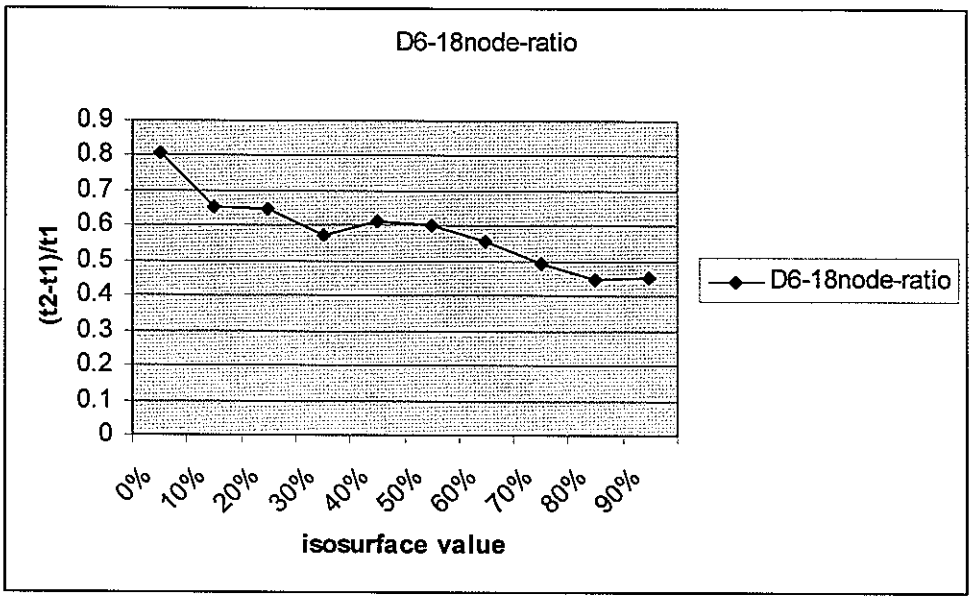


Figure 32. Ratio of Data transmission time and computing time

Combining the information in Figure 32 and Figure 29, we can conclude that increasing the computing time causes an increasing ratio of data transmission time to computing time; in other words, in the testing environment for this project, the bandwidth between the cluster and the visualization workstation is the bottleneck of the whole procedure.

6.4 Conclusions

VR CFD applications are facing challenges with larger CFD datasets and the more complex interactions needed by CFD design engineers. This research presents a new architecture of VR CFD applications. The key ideas behind this architecture is the

integration of the visualization and computations into a single high performance computing environment.

This thesis incorporates all existing technologies that can help VR CFD applications to cope with those challenges. Our implementation, VE_Suite, integrates a variety of software components together:

- Software components perform basic visualization tasks: VR Juggler, VTK, Iris Performer™ and VTKActorToPF
- Software component performs parallel computing: MPICH (implement MPI standard)
- Software component performs distributed computing: OmniORB (implement CORBA standard)
- Software component provide GUI: Java Swing

This research also illustrates the feasibility of developing a cross-platform VR CFD application. VR_Suite runs on different hardware such as SGI workstation, Linux cluster and tablet computer with various operating systems. Two developing languages, C++ and Java, are also integrated into one system.

As a result, this research successfully distributes the CFD visualization tasks to different computing resources and makes it work with the virtual environment. The discussion of performance proves that this distribution can improve the performance significantly, especially for a CFD dataset that is very large or has a complex shape so that it needs a large amount of computing time.

This project also helps to simplify the CFD VR application by moving the VTK computing tasks out of the main object of the VR application and making a standalone

user interaction console. These efforts make the whole software system more flexible and manageable.

Complex and multimodal user interaction is achieved by using a standalone console and a tablet computer. Especially for mechanical designing purposes, the user can draw and reshape the geometry in the console, select the computing resources, and visualize the relevant CFD results in the running time.

6.5 Future work

- Implementing pipeline parallelism

  This project implements data parallelism and task parallelism as described in Chapter 4. There is one more parallelism scheme: pipeline parallelism, which lets the software system put the result on every node into the visualization pipeline right after it is completed. Implementing pipeline parallelism may improve the performance for operations such as the isosurface, which does not require that the results appear in order.

- Improve extensibility and flexibility

  We need single configuration console to tell VE_Suite how to distribute computation tasks to different computation resources. If new resources are added, for example, a new node is added to Linux cluster or the users are authorized to use a new computer, the system can be easily reconfigured.

- Use computing power on SMP machine

Current VE_Suite uses SGI workstation as visualization facilities. There is still computing power on SGI workstations that can be used to perform VTK algorithms. Doing this can also bypass the network bottleneck in the current system.

BIBLIOGRAPHY

[1] "What is CFD?" http://www.fluent.com/solutions/whatcfd.htm (verified Jun 12, 2003).

[2] Cruz-Neira, C., Sandin, D. and DeFanti, T., "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," ACM SIGGRAPH, 1993

[3] Bierbaum, A., "VR Juggler: A virtual platform for virtual reality application development", MS thesis, Iowa State University, 2000.

[4] Schroeder, W., et al, "The Visualization Toolkit An Object-Oriented Approach to 3D Graphics", Prentice Hall PTR, 1998.

[5] "Flow Analysis Software Toolkit," http://www.fluent.com/solutions/whatcfd.htm (verified Jun 12, 2003).

[6] Bryson, S. and Levit, C., "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows", RNR Technical Report RNR-92-013, Oct 1991.

[7] Reimersdahl, T. et al, "ViSTA: a multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI", IPT2000, Iowa State University, Ames, 2000.

[8] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., and Cruz-Neira, C., "VR Juggler: A Virtual Platform for Virtual Reality Application Development", IEEE VR 2001, Yokohama, March 2001.

[9] "Iris Performer™ homepage", http://www.sgi.com/software/performer/ (verified Jun 12, 2003).

[10] "VTK homepage", http://www.vtk.org (verified Jun 12, 2003).

[11] "vtkActorToPF" homepage", http://brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF/ (verified Jun 12, 2003).

[12] "MPI homepage", http://www.mpi-forum.org (verified Jun 12, 2003).

[13] "MPICH homepage", http://www-unix.mcs.anl.gov/mpi/mpich/ (verified Jun 12, 2003).

[14] "OMG CORBA homepage", http://www.corba.org (verified Jun 12, 2003).

[15] "OmniORB homepage", http://omniorb.sourceforge.net/ (verified Jun 12, 2003).

[16] James, A., et al, "A parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets", Los Alamos National Laboratory – Technical Report # LAUR-00-1620.