Conceptual Design and Implementation of a Pipeline-Based VR-System Parallelized by CORBA, and Comparison with Existing Approaches

Andreas Gerndt,* Mark Asbach, Torsten Kuhlen, Christian Bischof
Center for Computing and Communication,
RWTH Aachen University, Aachen, Germany

Stefan Lankes,† Thomas Bemmerl
Chair for Operating Systems,
RWTH Aachen University, Aachen, Germany

Abstract

The object-oriented Virtual Reality toolkit ViSTA developed at Aachen University utilizes the Visualization Toolkit (VTK) in order to implement scientific visualization applications. VTK already offers parallelization possibilities. Its parallelization strategy cuts off the visualization pipeline between nodes and distributes these parts over several processes. By contrast, ViSTA makes use of an MPI-based parallelization framework whose parallelization components are decoupled from the algorithmic layer. Algorithms implemented here merely use sequential VTK pipelines for the computation.

Our new approach also parallelizes VTK's pipelines; however, the parallelization is based on CORBA. The advantages over MPI-based implementations are a straightforward integration into an object-oriented framework and the handling of complex data structures. In this paper, we present our CORBA-based implementation and compare it to others. We show that CORBA should be preferred for complex and object-oriented environments and that it has speed-up properties in parallel environments similar to MPI-based approaches.

Keywords: Visualization Pipeline, Parallelization, CORBA, MPI, Virtual Reality

1 Introduction

In general, Virtual Reality toolkits organize their rendering objects by means of scene graphs. This makes it possible to navigate and to manipulate virtual objects interactively. However, VR applications in the field of scientific visualization additionally need data flow design patterns since the objects actually handled are not graphical objects. These have to be created first and have to be updated frequently. One of the main approaches to control this visualization data flow is the pipes and filters pattern. Each node of this pipeline gets data from source nodes, deals with it, and offers the modified results as an input for the following neighbor nodes.

ViSTA [van Reimersdahl et al. 2000], the VR toolkit developed at Aachen University, also makes use of visualization pipelines in order to implement scientific visualization. This functionality is not completely implemented from scratch; rather the essential skeleton classes are offered by a toolkit called *Visualization Toolkit* (VTK) [Schroeder 2001].

The pipeline handling is always similar. First, the pipeline is assembled; thereafter, each node of the pipeline can be modified with appropriate parameters. Finally, the user can send an update command to the last node, typically the render node. This triggers off an execution through the whole pipeline starting with the first node, which usually contains simulation data. Afterwards, a repeated update command again goes through the pipeline but without executions because all nodes are still up-to-date. If in the meantime changes occurred at one node only the pipeline from this modified node to the last node is executed.

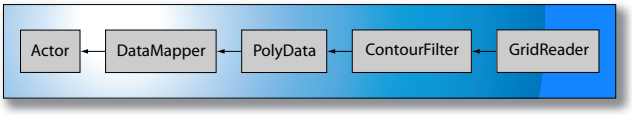

Figure 1: Data flow in a visualization pipeline.

Figure 1 shows a simple pipeline to compute iso-surfaces, in the following used as test case in this paper. The actor is a graphical object and is updated every time the rendering loop of the VR application is triggered. Hence, changing iso-values, which are parameters of the contour filter, displays modified iso-surfaces immediately.

Because VR applications require real time interaction, the achieved frame rates have to be above a certain frequency. Therefore, update computations especially of more complex visualization pipelines must be optimized. An obvious possibility is parallelization.

In this paper, we describe common parallelization schemes first. Pipeline distribution approaches based on these basic schemes are explained next, followed by a short introduction to CORBA and an explanation of our CORBA-based parallelization approach. Thereafter, test bed and results that show the usability and effectiveness of this new approach are presented. Conclusion and a glimpse at future work close this paper.

2 Parallelization Approaches

In general, all present operating systems offer functionalities to split a running process into two or more separated threads. Most system providers develop own solutions, although a common standard called *pthread* [Nichols et al. 1998] has been existing for some years. Because multiple threads spawned by the same process are executed in the same memory environment, the main disadvantage is the restriction merely to one computer node. Therefore, the possible speed-up is limited by the number of local processors. Furthermore, commonly used resources like shared I/O devices are bottlenecks and restrict the speed-up even more. The main idea of *pthread* is enabling concurrency, not parallelism.

*e-mail: gerndt@rz.rwth-aachen.de
†e-mail: stefan@lfbs.rwth-aachen.de

Long Paper

A second scheme uses messages to communicate between processes that are strictly separated. The most well-known standard is the *Message Passing Interface* (MPI) [Gropp et al. 1999]. Each process runs as a copy of the others and can be distinguished by a unique ID, which must be used to address communication partners. MPI can be used on shared memory systems as well as on distributed memory systems like PC clusters. The generalized version MPICH also runs on heterogeneous cluster systems.

2.1 VTK

VTK is an open source project and is frequently used in the field of scientific visualization. However, it is not a Virtual Reality toolkit but can be integrated into VR systems as a visualization enhancement. Therefore, we first consider the parallelization of VTK as an approach to speed-up VR applications.

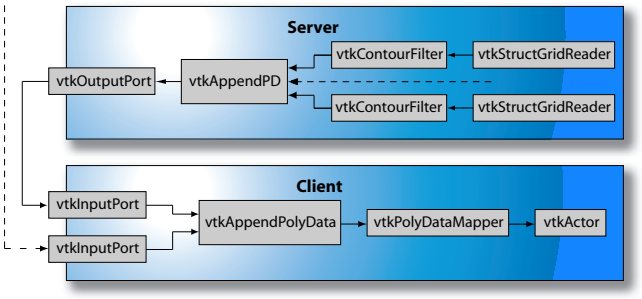

Figure 2: Pipeline distribution via ports.

By default, a number of filters can be executed in parallel using multi-threading. The currently executed filter is split into multiple threads, computes partial results, and joins them to one single output result. The whole parallelism is encapsulated in one node. In order to exceed this restricted parallelization possibility, a new pipelining mechanism was introduced [Ahrens et al. 2000]. Since then, normally used input and output links can be extended by input and output ports that define the crossings between different processes (Figure 2). Thus it allows splitting the pipeline into independent parts, which are able to run on distributed processes.

A multi process controller manages connection and propagation of data flow between distributed pipeline parts. This approach hides network communication and makes the parallelization aspects transparent for the user. However, in order to build up communication links between different processes, process numbers are used. Additionally, port tags have to be managed. This makes it more difficult to handle and to avoid errors.

Multi-threading as well as MPI can be used for the communication. We applied both variants and compared them with the others.

3 Viracocha

Although *ViSTA* applies VTK functionality in order to implement visualization pipelines, it follows a different parallelization approach. Instead of distributing parts of the pipeline, it distinguishes between separate computation jobs and distributes whole pipelines to single processes. Only the results are sent back to the visualization client where they can directly be hooked into the rendering loop. Therefore, *ViSTA* itself does not know anything about the creation procedure; and the computing server removes the pipeline each time an extraction job is finished.

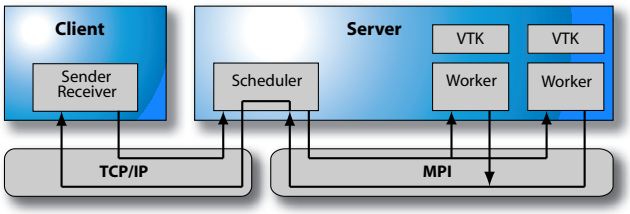

Figure 3: Viracocha's communication concept.

The parallelization management is implemented in a separate module called *Viracocha* [Gerndt et al. 2000; Schirski et al. 2003] and is depicted in Figure 3. Just like *ViSTA*, it is platform independent. The computing processes (called worker) communicate via MPI, while requests and data are sent via TCP/IP between the visualization client and the scheduler of the computing server.

Each worker can apply several commands in order to compute requests. These commands are fixed to specific problems; i.e., assembled extraction pipelines cannot be modified by the user anymore. One can only control the computation by offered command parameters, which must be transmitted before the computation. Despite this, these commands can consider load balances and other dynamic strategies for an optimum computation. The computed results are sent back to the visualization client and from then on they are decoupled from any pipeline interdependencies.

On the one hand, this asynchronous computation is an advantage because update events by the rendering loop need not be propagated through the whole pipeline, which could be a real-time problem due to network latencies. On the other hand, interactive iso-value modifications, a usual activity during interactive explorations, cannot simply apply by means of slightly changed pipelines and triggered update events.

4 CORBA-based approach

The *Common Object Request Broker Architecture* (CORBA) is an industry-defined standard [OMG 2002; Siegel 2000] for distributed object-based systems. This standard has been specified by the *Object Management Group* (OMG), a nonprofit organization with over 800 members, primarily from industry. The main goal of CORBA is to implement distributed objects on heterogeneous systems. Therefore, CORBA is independent of operating system, programming language, and CPU.

CORBA uses the remote object model described in [Tanenbaum and van Stean 2002]. The interface for these objects is specified in CORBA's *Interface Definition Language* (IDL). An interface definition with IDL provides syntax for expressing methods and their parameters; however, IDL has not the ability to describe the semantics of CORBA objects.

To implement a CORBA object, the interface definition must be compiled by a special IDL compiler. This compiler creates a skeleton for the server and a proxy for the client in a specific programming language (such as C++ or Java). Afterwards, the concrete class has to be implemented by deriving from the skeleton.

Figure 4 describes a remote object invocation. If a client invokes a method of the CORBA object, which is specified by the IDL, the proxy automatically marshals the method invocation into small

messages, which the client process sends to the server process. This is done in cooperation with the *Object Request Broker* (ORB), which is a core component of CORBA. On the server-side, the skeleton receives incoming messages, unmarshals them, and invokes the right method of the CORBA object. In a similar way, the result of the invocation has to be sent back from the server to the client.

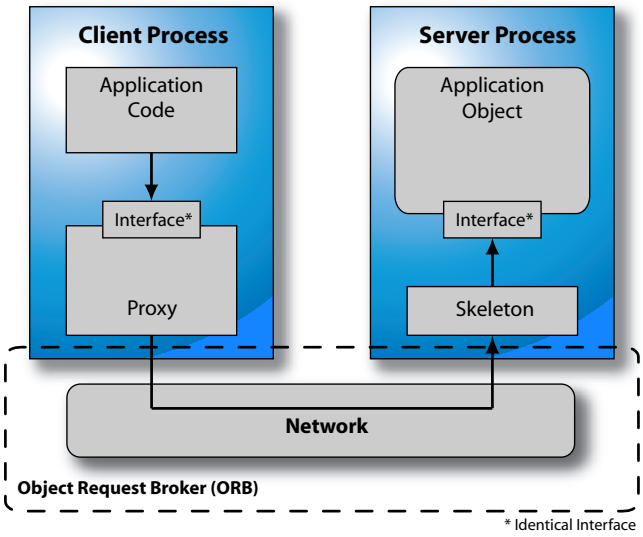

Figure 4: Scheme of CORBA's remote object model.

In order to invoke a method, the client has to know the location and the supported protocol of the CORBA object. All these information are encoded in the *Interoperable Object Reference* (IOR), which the OMG defines as address mechanism to an object.

CORBA's standard protocol is based on TCP/IP, which is called *Internet Inter-ORB Protocol* (IIOP). But every CORBA implementation can define its own inter-ORB protocols. For instance, a shared memory based protocol achieves better performance values than a TCP/IP-based protocol and is an attractive alternative for VR systems. The ORB hides these protocols and the developer does not need to know, which protocol is used by his application.

The description of the object invocation shows that the developer has no direct contact to the underlying network. All messages are automatically generated by the proxy and skeleton. Therefore, the developer can concentrate on solving the intrinsic problem and does not need to design the underlying communication architecture. This reduces the error rate and increases the stability of distributed applications.

4.1 CORBA-Based Visualization Pipeline

Our new parallelization approach takes the design of VTK in order to parallelize pipelines, even though it implements the selected schemes slightly more consistent. Instead of using specialized port objects with adapted methods, we just derived CORBA classes from VTK filter and data objects and implemented input and output methods in order to enable inter-process pipelining. Therefore, it is now possible to link these CORBA-based nodes simply into the pipeline like other native VTK pipeline nodes.

At first we implemented two classes in our prototype: a poly data mapper and a poly data source that work as a poly data bridge between two pipeline pieces. The implementation equivalent to VTK's distributed pipeline is depicted in Figure 5.

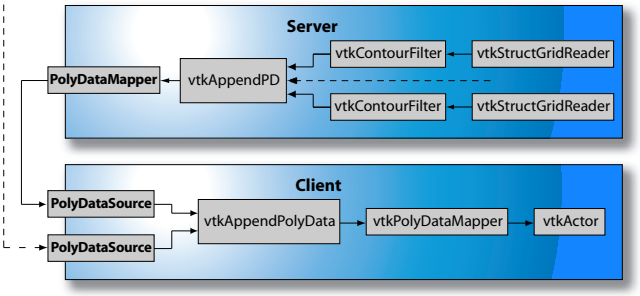

Figure 5: CORBA-based poly data objects control server / client pipeline.

Multiple identical server processes exist in parallel and offer their VTK visualization pipeline via CORBA. A client process controls the servers, demands different parts of the data set from each server and conjoins the results. Even in our approach, some additional code must be invoked at start-up, which initializes the CORBA environment. Furthermore, the application programmer is still responsible to distribute the load of the pipeline to several processes. The following code extract shows the connection of an already existing server pipeline to the CORBA-based poly data mapper object:

```
PolyDataMapper_Impl * pPDM = NULL;
pPDM = PolyDataMapper_Impl::New();
pPDM->SetInput(pMyVtkPipeline->GetOutput());
PolyDataMapper_var pPDM_var = pPDM->_this();

// hand over the control to the ORB
pOrb->run();
```

Listing 1: Object activation on a server process.

In contrast to already described solutions, CORBA does not dictate any process IDs or similar by the program code. A CORBA client just needs to de-reference the correct IOR of a server object instance in order to bind it to the client process. It is important to emphasize that this is an object-based identification independent of running processes, which can additionally be simplified by special name services. The listing 2 depicts the needed steps at the client side.

The main advantage is the transparency of the location of running server objects. The references stored as IOR strings can point to an object in the same process, in a second process on the same machine, or on a remote host anywhere on the internet. Furthermore, the CORBA application is capable to append and to remove additional computing processes during run-time. This is usually a restriction of MPI-based applications, where the applied processes are determined at start-up.

The next advantage is that we can apply object-oriented methods to extend the functionality of our interface classes. For instance, one could derive new specialized classes from poly data mapper.

Our implementation, however, uses a kind of class factory to enable client access to the sever pipeline. The poly data mapper only enables pipeline communication and data propagation. But the factory creates all needed objects of the server controlled pipeline including mapper objects. Therefore, two classes have to be known on the client side: the factory and the mapper class. In our prototype, the factory defines iso-surface computation. The appropriate IDL interface looks like the code in listing 3.

```
vtkAppendPolyData * pAPD = NULL;
pAPD = vtkAppendPolyData::New();

for (int i=0; i<entries; ++i)
{
  CORBA::Object_var obj;

  // create proxy for server object
  obj = pOrb->string_to_object
     ((const char *) iorString[i]);
  PolyDataMapper_var pPolyDataMapper =
     PolyDataMapper::_narrow (obj.in());

  // connect to remaining pipeline
  PolyDataSource * pPDS =
     PolyDataSource::New();
  pPDS->SetInput(pPolyDataMapper);
  pAPD->AddInput(pPDS->GetOutput());

  // decreases vtk reference counter only!
  pPDS->Delete();
}
```

Listing 2: De-referencing remote objects.

```
interface IsoSurface
{
  PolyDataMapper GetOutput();
  oneway void SetData(in string DataFileName,
    in string ScalarName);

  attribute float FirstValue;
  attribute float SecondValue;
};
```

Listing 3: Iso-surface IDL description.

Instead of poly data mapper IORs, our client process now uses IOR references of created iso-surface objects in order to modify server pipelines. These iso-surface objects are also used to receive respective poly data mapper proxies, which allow access to the end points of established server pipelines. These proxies are connected via poly data sources to the remaining client pipeline. A released update event is now propagated via the CORBA managed poly data bridge to the end of the merged pipeline, which thereafter computes data in reverse order and finally returns the wanted results for the rendering. Once defined, for instance iso-values can be modified and read like attributes of any locally defined class:

```
oldValue = pIsosurface_var->FirstValue();
pIsosurface_var->FirstValue (newValue);
```

Listing 4: Access to remote object attributes.

In addition to this convenient handling, it is possible to verify the correct type of an object. This is not available in the other approaches. Based on the IDL description, this matching test is already applied if a downcast of a reference to a certain class is carried out. The validity of a reference is checked explicitly. If an invalid reference is utilized, an exception will be thrown.

5 Test bed

A newly implemented approach should be compared to already existing approaches and tested under several conditions. We executed our CORBA implementation on different computer systems in order to extract iso-surfaces of a computational fluid dynamics (CFD) simulation. The run-time results are compared to the results of the three other described implementations: *Viracocha*, VTK with MPI (VTK-MPI), and VTK with multi-threading (VTK-MT).

5.1 Multi-Block Data Sets

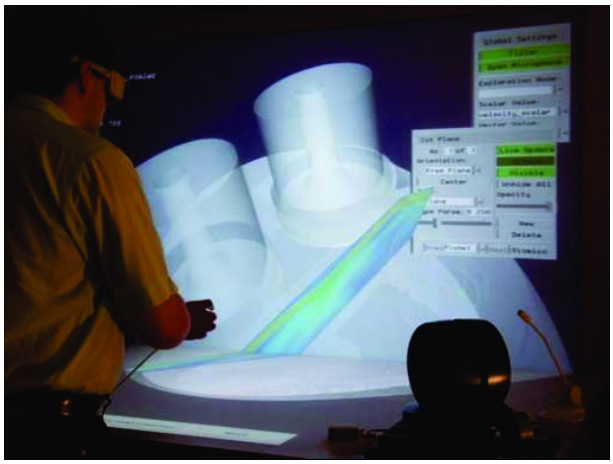

Figure 6: Interactive exploration of a multi-block engine data set at a Holobench.

We pursue data parallelism, for it can easily be mapped to multiple pipeline threads. Therefore, we applied multi-block data sets, which are already stored in a number of single structured grid data files. Each of these can now be directly assigned to one grid reader, the source node of our parallelized pipeline. Additional data subdividing schemes are possible but not considered here.

Our example data set describes the inflow of a 4 valves combustion engine (Figure 6) [Abdelfattah 1998]. The domain is decomposed into 23 single blocks, which all have different data sizes[1]; and the inflow simulation is divided into 63 time steps. We arbitrarily used time level 20 for our test case and distributed its single blocks to the available processes. There, two velocity scalar values were set to compute iso-surfaces.

5.2 Hardware and Software

In order to prove the new CORBA approach to be efficient in parallel VR environments, we applied different tests on hardware installed at the computing center of Aachen University. A Sun Fire 6800, a shared-memory system (24 GByte) with 24 processors (Ultra Sparc III, 800 MHz), came into operation. Because of our 23 block example data set, that multi-processor system was perfectly suitable for the applied tests. In this case, each processor is dedicated to one of the 23 server pipelines or to the client pipeline.

The CORBA performance measured on the Sun Fire was compared with results on a dual processor (500 MHz) Linux PC with 880

[1]The data blocks are composed of 1600 to 13000 grid points and summation of all block sizes are 22 MByte.

MByte main memory, and a single processor (1 GHz) Linux PC with 256 MByte RAM. All systems were connected to an SGI Onyx 2 visualization host with 4 processors (195 MHz) and 2 GByte shared main memory.

All tested applications were based on VTK version 4.2. As communication software, we applied a frequently used CORBA implementation called *The ACE ORB* (TAO). It is freely available and open source as well. The applied version 1.3a fulfills our requirement of platform independency. On all platforms, only the IIOP protocol was used. In comparison, MPI was already installed on all considered systems. The Sun Fire makes use of an optimized implementation that uses shared memory for communication and data exchange. On the Linux PCs, we used MPICH, which communicates via TCP/IP.

6 Results

We carried out a variety of tests. Our first series was executed on the Sun Fire only. Here, each approach had to show its strengths and shortcomings. The second series exclusively examined the CORBA approach in order to depict its behavior in a distributed network-based environment.

6.1 Parallelization Characteristics

To get a measure of effectiveness, we built up a non parallelized VTK pipeline with 23 data set readers, one per data set file. The determined update time was the minimal time needed to process the iso-surface computation. A parallel version that uses the client and only one server process would always be slower because of communication overhead. Furthermore, because of distributing the block data files evenly on all processes, we supposed that those that have most of the blocks would restrict the possible speed-up. These assumptions lead to a line called "Minimum" in Figure 7 describing the theoretically achievable processing time, with no data transmission overhead and equal block processing times.

The next step was to start the same test application for each approach on the Sun cluster varying the number of parallel processing pipelines from 1 up to 23. Figure 7 shows the time consumed to load 23 structured grid data sets into memory, to extract iso-surfaces for two iso-values, to send partial results to the client application, and to assemble the final polygon data set there. The client process, incidentally, is not considered in the figure. Additionally, *Viracocha* needed a further scheduler process, which was also not counted here.

Obviously, the CORBA approach performs nearly as good as VTK's MPI approach. The VTK version based on *pthread* performs rather badly - probably because a multi-threaded application holds data structures that are concurrently accessed by multiple processors in a combined address space, which can lead to cache thrashing.

In order to handle arbitrary file formats, Viracocha uses the generic data set reader of VTK. This flexibility is achieved at the expense of performance how one can see clearly in the figure. However, using the specific structured grid reader, the performance is similar to VTK-MPI or CORBA respectively.

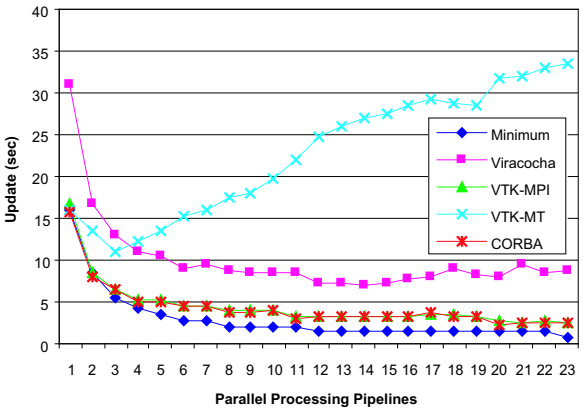

Figure 7: Time needed to update the complete pipeline.

6.2 Real Time Aspects

The first tests have shown that CORBA as well as MPI can be used for parallelization approaches without considerable restrictions. The next test should inspect the capability to work reliably in time critical applications.

The most time consuming portion of the previous measurement was the time needed to load data files. However, a data set is often loaded once and then used for multiple subsequent extraction executions. Therefore, not only the time needed to update the whole pipeline including data loading but also the update time of an already initialized and later on modified pipeline, for instance to show iso-surfaces for changed iso-values, can be of particular interest. Interactive data exploration makes permanently use of the latter one.

Since the sole iso-surface extraction is much faster than the process of loading and data pre-processing, parallelization overhead is more significant now.

The time line of a theoretical minimum in Figure 8 was determined in the same way we estimated the minimum that includes data loading. Viracocha was not considered here, because it does not provide update features for already computed iso-surfaces yet. The remaining approaches showed parallelization gain, but it was less impressive with the CORBA variant. Slight differences in the inter-process interface design probably result in fewer time lags for VTK's MPI approach.

Note that we do not make use of TAO's real-time CORBA possibilities yet. A significant speed gain is expected to result from replacing parts of the VTK update mechanism by the CORBA event service.

6.3 Network Application

The last test should emphasize CORBA's capability to distribute applications across boundaries of local computer systems. An advisable installation makes use of a visualization host optimized for real time rendering. On the other side, a high performance cluster computes time consuming jobs. Our test bed made use of an SGI Onyx 2 as visualization client and applied the Sun Fire and Linux PCs with one and two processors as server for the parallel computation.

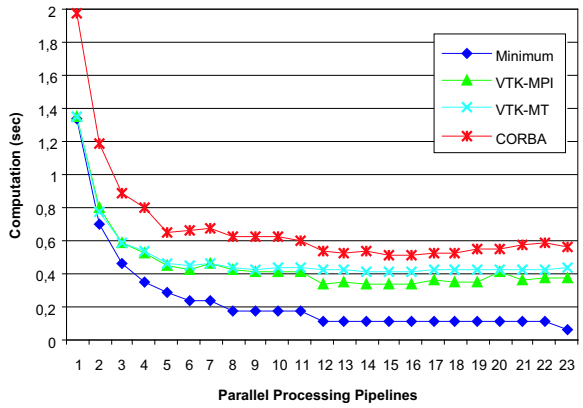

Figure 8: Update time for frequently modified iso-values.

As depicted in Figure 9, without pipeline parallelization, the Onyx needed 42 seconds. By running a parallel version directly on the Onyx, we already received a gain by including 2 processors. A mentionable result, however, is that we obtained a maximum speed-up employing 23 processes even though it is just a 4 processor Onyx. Probably, in that situation, the operation system is capable to schedule the available resources a little better.

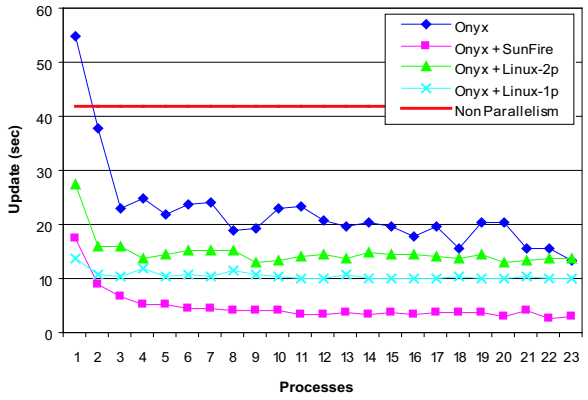

Figure 9: Update time for network connected components.

The same effect could be considered using the Linux PCs. Here, we could measure a slightly higher speed-up when using more processes than available processors. The last measurement, the combination of Onyx and Sun presented the expected efficiency. The achieved speed-up is similar to the one shown in Figure 7 slightly reduced by network latencies.

The importance of latencies can be demonstrated when needed computing time is low and communication cost increases. The results are shown in Figure 10. On the one hand, it is obvious that the Onyx had problems to manage parallel extraction updates. It was many times slower than the sequential version.

On the other hand, the heterogeneous platform combinations connected via Ethernet could not achieve a considerable speed-up as well. Despite this, it is remarkable that already off-the-shelf PCs

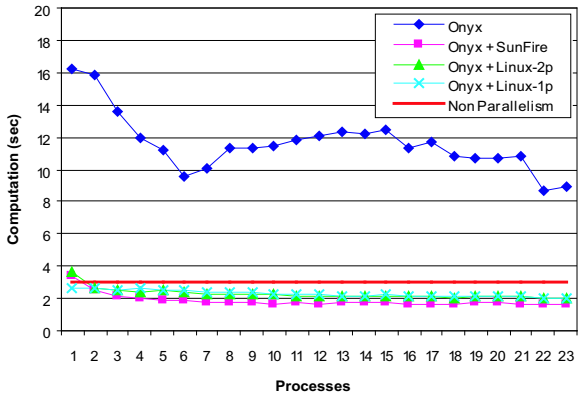

Figure 10: Network in-between and update without loading data sets.

could shorten the computing time, in contrast to the non parallel execution. Summarizing the results, also in this last test case, CORBA could be implemented successfully without measurable disadvantages.

7 Conclusion

In this paper, we described the necessity to parallelize a pipeline-based VR system. Our presented approach, which is based on CORBA, can be integrated seamlessly into object-oriented frameworks. Main advantages are type safety, maintainability and extensibility. It can easily be integrated into parallelized VR systems and offers possibilities to distribute any parts of this system to arbitrary nodes of a heterogeneous environment.

We also showed that our solution achieves sufficient performance values. In comparison to approaches with MPI or multi threading, the described CORBA solution had similar acceleration qualities. A slightly higher speed-up could be considered for MPI solutions when computation time was already short. However, the differences are tolerable because the advantages of our software design clearly outweigh the disadvantages.

8 Future Work

Changing implementation details might remove noticed differences. Furthermore, a different CORBA implementation could also lead to higher performance. Therefore, we will compare the currently used CORBA implementation TAO with several other non commercial (like ROFES [Lankes 2003; Lankes et al. 2003]) as well as commercial CORBA versions (like *ORBIX* or *VisiBroker*).

Up to now, we only use IIOP, the standard communication protocol of CORBA. However, CORBA is not restricted to this protocol but offers a variety of optimized communication possibilities, for instance by means of shared memory. This is obviously the preferred communication interface for the SMP Sun Fire. MPI for Sun makes already use of it. Therefore, we will evaluate and integrate these more progressive solutions also for our CORBA approach.

A really important aspect for Virtual Reality is a CORBA extension called Real Time CORBA (RT-CORBA). This extension, which is explained in [OMG 1998; Schmidt and Kuhns 2000], makes it possible to develop distributed real-time applications. For instance, RT-CORBA can assign higher priorities to processes that demand really strong real-time constrains. For such processes, RT-CORBA tries to fall short of dictated maximum latencies. A variety of mechanisms, like multiple connection channels, can be made available.

We have not made use of real-time facilities of TAO in order to parallelize the visualization pipeline yet. It might enable a more continuous data throughput. Moreover, RT-CORBA appears to be very interesting in order to distribute further components of Virtual Environments that need short response time. For instance, the tracking system could be decoupled from the visualization system and shift to one or more separated computer nodes. One area of application could be physically-based modeling, which requires quite a lot of hardware resources in order to reach high sampling rates. The range of use seems to be manifold, which is why we will investigate the possibilities to develop a distributed VR system by means of RT-CORBA.

Acknowledgements

The authors are grateful to the Institute of Aerodynamics, Aachen University, for the combustion engine data set kindly made available.

References

ABDELFATTAH, A. 1998. *Numerische Simulation von Strömungen in 2- und 4-Ventil-Motoren*. Shaker Verlag, Aachen. PhD thesis, RWTH Aachen University.

AHRENS, J., LAW, C., SCHROEDER, W., MARTIN, K., AND PAPKA, M. 2000. *A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets*. Los Alamos National Laboratory. Technical Report #LAUR-00-1620.

GERNDT, A., VAN REIMERSDAHL, T., KUHLEN, T., HENRICHS, J., AND BISCHOF, C. 2000. A Parallel Approach for VR-based Visualization of CFD Data with PC Clusters. In *Proceedings of the 16th IMACS World Congress*.

GROPP, W., LUSK, E., AND SKJELLUM, A. 1999. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. MIT Press.

LANKES, S., JABS, A., AND BEMMERL, T. 2003. Integration of a CAN-based Connection-oriented Communication Model into Real-Time CORBA. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), 11th Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2003)*.

LANKES, S. 2003. *Konzeption und Umsetzung einer echtzeitfähigen Verteilungsplattform für eingebettete Systeme*. Shaker Verlag. PhD thesis, RWTH Aachen University.

NICHOLS, B., BUTTLAR, D., AND FARRELL, J. 1998. *Pthread Programming*. O'Reilly & Associates.

OMG TECHNICAL DOCUMENT ORBOS/98-10-05. 1998. *Real-time CORBA – Joint Submission*.

OMG TECHNICAL DOCUMENT FORMAL/02-06-01. 2002. *The Common Object Request Broker – Architecture and Specification*, 3.0 ed.

SCHIRSKI, M., GERNDT, A., VAN REIMERSDAHL, T., KUHLEN, T., ADOMEIT, P., LANG, O., PISCHINGER, S., AND BISCHOF, C. 2003. ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments. In *Proceedings of the 7th International Immersive Projection Technologies Workshop, and 9th Eurographics Workshop on Virtual Environments*, 77–85.

SCHMIDT, D. C., AND KUHNS, F. 2000. An Overview of the Real-Time CORBA Specification. *IEEE Computer 33*, 6, 56–63.

SCHROEDER, W. 2001. *The VTK User's Guide*. Kitware Inc.

SIEGEL, J. 2000. *CORBA 3 – Fundamentals and Programming, Second Edition*. OMG Press.

TANENBAUM, A. S., AND VAN STEAN, M. 2002. *Distributed Systems – Principles and Paradigms*. Prentice Hall.

VAN REIMERSDAHL, T., KUHLEN, T., GERNDT, A., HENRICHS, J., AND BISCHOF, C. 2000. ViSTA: a Multimodal, Platform-Independent VR-Toolkit Based on WTK, VTK, and MPI. In *Proceedings of the 4th International Immersive Projection Technology Workshop*.