ICAT 2003
December 3-5, Tokyo, JAPAN

Distributed multigrid algorithms for interactive scientific simulations on clusters

Ronan Gaugne, Sylvain Jubertie, Sophie Robert

Laboratoire d'Informatique Fondamentale d'Orléans
Université d'Orléans, BatIIIA Rue Léonard de Vinci, BP6759, 45067 Orléans Cedex
ronan.gaugne@lifo.univ-orleans.fr, sylvain.jubertie@iut.univ-orleans.fr,
sophie.robert@lifo.univ-orleans.fr

Abstract

In many scientific domains, researchers can really benefit from implementing simulations in a Virtual Reality (VR) development in increasing the interactions and therefore the comprehension of physical phenomena. To comfort this direction, two problems need to be resolved: the cost of the VR platform and the coupling between the simulation and the visualization. Our project is to propose an original VR environment based on software tools allowing to replace advantageously a supercomputer by PCs cluster and which rests on the ability of simulation parallelization in respect of the coupling with the visualization.

Key words: Virtual reality, real-time simulation, distributed multigrid algorithm, cluster platform, interaction.

1. Introduction

In the world of supercomputers, the cluster architectures built with PCs and a powerful network offer competitive performances for scientific applications. Compared to the dedicated supercomputers like Cray machines, these architectures have the great advantages to be both scalable and modular in addition to the high performance-price ratio. The virtual reality domain applied to the scientific modeling induces some strong constraints on performances to improve the rendering of interactive physical phenomenon while delivering real-time simulations. While the quality of low cost graphic cards increased, some recent software developments such as NetJuggler [2], SoftGenLock [1] or WireGL [7], dedicated to the automatic parallelization rendering computations, allow researchers to efficiently develop VR applications on CAVE or WorkBench environments for instance. Consequently, the cluster architectures become a relevant choice to implement VR real-time applications.

In many scientific domains, researchers can really benefit from implementing simulations in a VR development. Such applications make it possible to increase the interactions and therefore the comprehension of physical phenomena. But this domain still remains poorly exploited because existing platforms are designed either for simulation or visualization. The goal of our project is to provide scalable implementations which take advantage of large PC's clusters capabilities. Our approach allows the coupling of simulation and visualization parts to provide real-time implementations.

In order to experiment our work on a representative scientific modeling, we choose to simulate the fluid motion. Indeed, the mechanics of fluids is a relevant domain which can clearly take advantages of simulation in a VR environment. Furthermore, this kind of application requires realistic graphical rendering and can profit of interactions such as obstacles additions and on the fly parameters modifications.

The following section is dedicated to the description of our VR platform based on the dual VRJuggler and NetJuggler associated to the SoftGenLock tool. We describe how these open tools allow us to exploit all the capabilities of a cluster in replacement of dedicated supercomputers. The section 3 gives a detailed description of our fluid motion based on the multigrid method. Finally before the conclusion, the section 4 describes the fluid motion implementation on our VR platform to illustrate its richness.

2. VR platform from NetJuggler and SoftGenLock

The development of VR applications is not an easy task due to the complexity of the components like the displays, the devices or the computer center. We describe here a set of tools whose combination constitutes a powerful and easy to use VR development environment working on a PCs cluster.

The open source library VRJuggler [5] offers an environment for VR application development which eases the management of all the components of a VR platform such as the displays and the input or output devices. The VRJuggler library is composed of a kernel which pilots the application, some managers for each part of the system and a set of configuration files. From

This work is partially supported by the RNTL project GeoBench

these components, users can develop VR applications where proxies make possible to completely abstract the physical devices. The set of configuration files describes these devices whose data is processed and restored by the input manager to the VRJuggler application via these proxies. A rich graphical interface ensures the application description in terms of displays, input or output devices and dynamic application reconfiguration. Finally, the integration of graphics API as OpenGL or performer, offers to VRJuggler users a good development environment which is exploited for CAVE or Workbench platforms piloted by supercomputers.

In order to replace the supercomputers by PCs cluster we have to solve two major problems.

- The management of the devices distributed on the different nodes of the cluster,

- The video signal synchronization due to the distribution of visualization computing on the cluster.

A response to the first point has been provided by a recent open source called NetJuggler that allows to extend VRJuggler on clusters [3]. Exploiting the decomposition of VRJuggler in a kernel and some managers, NetJuggler consists in additional managers to control the distribution of the displays or the devices on different nodes of the cluster. This control is totally transparent to VRJuggler and so to the user. The principle of NetJuggler is to replicate a VRJuggler session on all the nodes of the cluster. Since some of these nodes support effectively the devices, NetJuggler ensures the broadcast of their data. Each VRJuggler session reacts as if the device was local. The MPI API is used to express communications between the different sessions and NetJuggler replaces the proxy by the management of a client server proxy for each device. In order to define the server and the clients, we need to add to the VRJuggler configuration files the node name of the host for each input and output. From this information and the definition of the cluster, NetJuggler is able to construct and manage the network implied by the distribution of the devices on the cluster.

For the second problem induced by VR platforms based on clusters, three level of synchronization can be identified: DataLock, SwapLock and GenLock. The first one that deals with the data coherency is naturally solved by NetJuggler as all the sessions process the same rendering computations. The SwapLock that concerns the buffer swap is also treated by NetJuggler by using an additional common synchronization barrier before the buffer swap. In case of active stereo, a frame level synchronization called GenLock needs a particular attention. The solution we choose to integrate in our VR platform is an open source software called SoftGenLock for linux platform. Independently of the graphical cards, this software detailed in [2] relies on a real time linux kernel so that each graphic node computes a precise time delay between its video signal and the master one. Then the graphical node can locally decide to adapt its signal generation speed. The synchronization is not an exact one but is obtained by control and regular modification.

The gathering of all these tools, as described in Fig.1, provides a performing VR environment for PCs cluster which has been successfully experimented by the BRGM research center[1] [8] on a WorkBench VR platform used to visualize geological simulations such as consequences of flood.

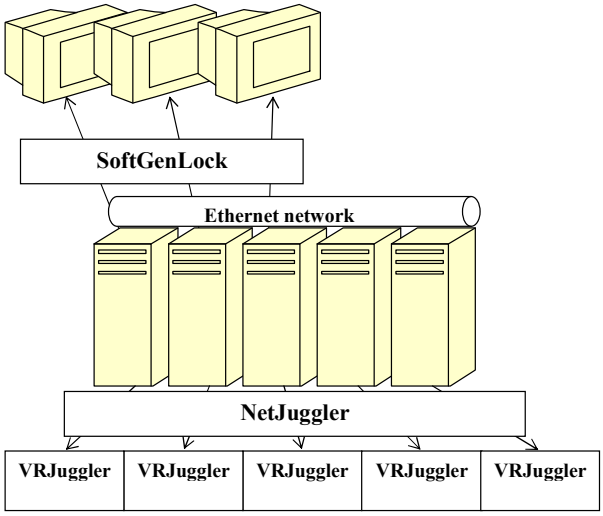


Fig. 1 Our VR platform

3. Fluid motion from the multigrid approach

In [9], Jos Stam proposes an interactive simulation of fluid motion which is based on the resolution of Navier-Stokes equations. Stam aims at offering a simple realistic rendering of the fluid motion. We are interested in adapting his approach to get closer to physical phenomena. In regard to these objectives, the multigrid algorithm allows to build an interesting Navier-Stokes equations solver. The section gives a brief description of the fluid modeling and of the multigrid concept.

3.1. The fluid motion equations

Our fluid motion is expressed as a velocity field u and a density field ρ from the below Navier-Stokes equations using the Helmholtz-Hodge decomposition as explained in [9]. The term v designs the viscosity rate, κ the diffusion rate and f, S respectively the density and the velocity added by a user.

[1] http://www.brgm.fr

$$\frac{\partial \rho}{\partial t} = -u\nabla\rho + \kappa\nabla^2\rho + S$$

Eq 1 Density equation

$$\frac{\partial u}{\partial t} = P(-u\nabla u + \nu\nabla^2 u + f)$$

Eq 2 Velocity equation

Then fluid flows are produced by the density moves through the velocity field.

To model the fluid motion, we use a finite representation of the fluid with a couple (ρ, u) for each point of a grid of interest. The equations (1) and (2) need then to be solved on this finite model at each time step Δt. Their right part admits three terms which are separately solved as each one represents a particular effect on the fluid. For the density equation, the first term called advection expresses that the density follows the velocity field, the second one that the density diffuses at a certain rate, and the last one that the density is increased by external sources. Concerning the right part of the velocity equation, we find the analogue steps of advection where a velocity vector evolves in regards to its neighbours, viscous diffusion and additional force. The operator P called projection is an additional step for the velocity due to the Helmholtz-Hodge decomposition and the mass conservation of fluids.

Without getting into a detailed description of the mathematical modelling, we need to precise an important feature of the Navier-Stokes equations solver which concerns the diffusion and the projection steps. Indeed, these steps need a discretization of the ∇^2 operator and lead to a linear system. The corresponding solver which is very time consuming is the heart of the fluid algorithm. The Stam choice is a Gauss-Seidel relaxation that is sufficient to get a convincing rendering. However, a poor convergence speed of this algorithm compels to reduce the simulation quality. To get closer to physical phenomena, the multigrid method appears to be a good candidate to meet the constraints of real-time, scalability.

3.2. The multigrid method

The multigrid algorithm is a general concept to solve a differential equation starting from the discretization in a linear system [6]. Let a grid of size $n \times n$ be the finite representation of the fluid, the linear system is then of the form $Ax=b$ where A represents a $n^2 \times n^2$ float matrix usually very sparse. If M denotes an elementary iterative method to solve the linear system from an initial solution such as the Gauss-Seidel relaxation, then the multigrid algorithm consists in the following steps:

1. Use M to approximate a first solution x_1 for x,

2. Estimate the error and solve an intermediary linear system,

3. Use M to re-compute a new solution x_2 for x,

which can also be iterative.

The multigrid originality comes from the step 2 process that leads to solve an equation of the form $Ae=b-Ax_1$ where x_1 is the approximated solution found in step 1 and e represents the error factor between the approximated solution and the exact one. As the process is iterative, the multigrid method allows to safely decrease the step 2 complexity. Two operators, a restriction operator and a prolongation operator, relate the $n \times n$ grid called the fine grid to a coarse one of size $n/2 \times n/2$ as described in Fig.2. The step 2 linear system of the detailed algorithm described in Fig.3 is then 16 times less complex.

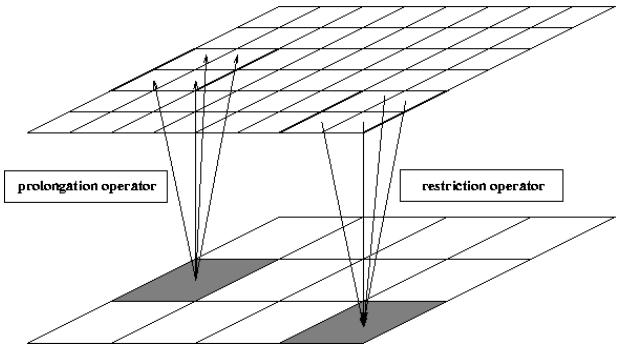


Fig. 2 : A two levels multigrid algorithm

a. Find x_1 such that $Ax_1=b$ from an initial vector (random one for the first iteration)

b. Apply the restriction operator to x_1, A and b resulting respectively *to x'*, A' and *b'*

c. Find *e'* such that $A'e'=b'-A'x'$

d. Apply the prolongation operator to *e'* resulting to e

e. Find x_2 such as $Ax_2=b$ from $x'+e$ as initial vector for this new iteration step.

Fig. 3 : Multigrid algorithm

This algorithm can be recursive in solving the step 2 by a new two levels multigrid method. It can be useful in order to decrease the global complexity when the initial grid is strongly fine.

Where the Stam Gauss-Seidel solver needs 20 iterations to converge, the multigrid algorithm reaches an equivalent level of accuracy with 5, 7 and 8 iterations respectively for the steps 1, 2 and 3, always with a Gauss-Seidel as elementary solver M. A gain of 20% is then obtained in term of complexity thanks to the use of the coarse grid in the step 2.

4. Fluid motion in a VR application

4.1. Parallelizing simulation in our VR environment

In section 2 the NetJuggler environment has been described as a good solution for using PCs cluster in the place of supercomputers. However, the cluster power seems to be under used as the rendering session is replicated on all the nodes. In many cases this is sufficient to deliver efficient VR applications. But it is not the case in scientific domains where the simulation part is really greedy in computations. NetJuggler uses the MPI API as communication library and therefore offers an easy way to integrate parallelized simulation in the VRJuggler rendering loop. As described in [4], the simulation can be any parallelized program using the MPI API or any MPI based tool. The program is then a SPMD one where all the additional communications are transparent to the NetJuggler API. The only constraint for the programmer is to deliver the simulation results to all the nodes of the cluster without altering the NetJuggler process.

When the simulation follows a data parallel model without the node identifier in the program or when a library as PETSc is used to take in charge the parallelism and above all the data flow, NetJuggler ensures the scalability and portability of the application from configuration files. A large number of scientific simulations can be developed in respect of these assumptions. In these cases, the user can really benefit from the NetJuggler management. On the other side, for advanced user it is possible to control completely its parallel simulation by specifying directly in the program the role of each simulation node. This approach has the great advantage to enable a precise management of data flow in order to reinforce the adequacy of the simulation with the visualization constraints. Nevertheless, one can notice that in both cases even nodes which are not attached to a screen do perform graphical operations.

We chose the advanced approach to design our fluid motion with more control parallelisms to experiment the optimizations that a strong link between the visualization and the simulation can offer.

4.2. Fluid motion implementation

The fluid motion in a VR application is a representative example of requirements in terms of efficient simulation and interactivities with visualization. As described in section 3, we chose to model the fluid motion from Navier-Stokes equations which lead to an algorithm whose main step is the repetitive solving of different linear systems for the diffusion and projection steps. In order to reach a real-time simulation we parallelize our solver by decomposing the fluid domain on the different cluster nodes. The most significant part of this parallelization is the multigrid algorithm because it could generate a great number of communications.

To avoid this disadvantage we use the Gauss-Seidel as elementary *M* method and modify the linear system resolution in order to restrict the communications to border exchanges as described in Fig.4. The multigrid steps become the followings:

1. Perform Gauss-Seidel relaxation on the local fine grid
2. Exchange borders with neighbour nodes
3. Restrict the error to the local coarse grid
4. Perform Gauss-Seidel relaxation on this coarse grid
5. Execute the prolongation of the result
6. Perform Gauss-Seidel relaxation on the fine grid
7. Exchange borders for the next step.

The parallelization of the sequential multigrid algorithm would have consisted of exchanging the borders between each iteration of the step 1, 4 and 6. But this version delivers analogue simulation accuracy and convergence speed while ensuring scalabity and low cost communications.

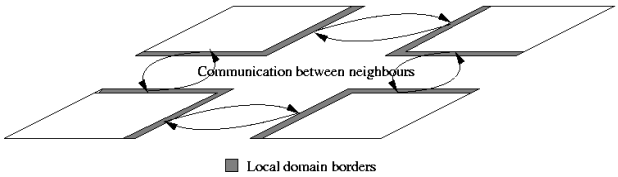


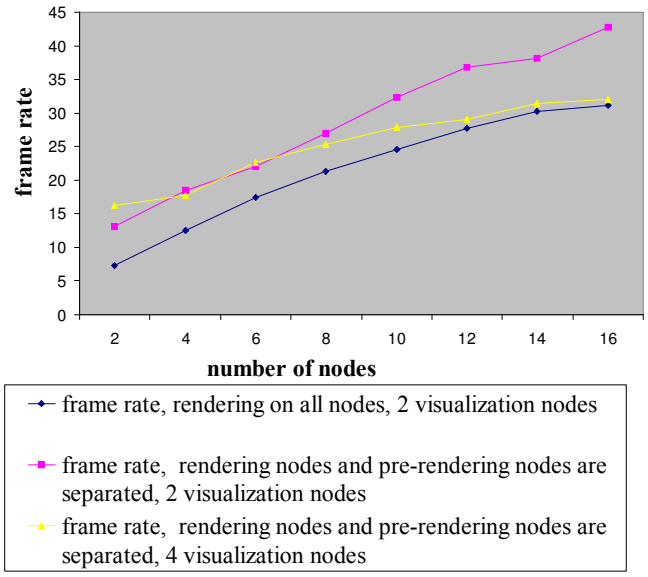
Fig 4 : Border communications

We have implemented our VR application based on the fluid motion on a cluster of eight Intel BI XEON DP 2.6GHz nodes. These nodes are equipped with Nvidia GeForce FX5900 with 128Mo RAM and are connected to a Giga Ethernet network. As described in Tab.1 the application runs on two or four graphical nodes where the simulation is distributed on all the cluster nodes as this part is the most expensive. Two fluid sources for a 160x160 grid are simulated with a mixer materialized by a little triangle which is controlled by a wireless joypad as input device. The joypad is also used to move the scene to the user suitability.

Usually, in case of high cost simulations, we need more computation than rendering nodes to ensure the real-time constraint of VR application. Then the NetJuggler choice to replicate the rendering computation on all the nodes may become a handicap to performances. In this sense, we present three series of benchmarks to illustrate the advantages of our approach with regard to the cluster technology. In the first serie, we simply use NetJuggler as a SPMD platform, i.e. all instances are performing both simulation and rendering computations. In this serie, only two processes are really displaying on a screen. In the two last series, we have distinguished processes which are doing graphical computations from those which are computing simulation steps thanks to

some tricks with MPI rank inside NetJuggler. The only difference between the two last series is the number of displaying nodes.

For example, this wiliness allows to win above 30% in frames rate when the number of simulation nodes is at least eight as illustrated by the two first series. Furthermore, our implementation leads to a 23 frames rate on six simulations nodes as in [4] but with a grid size 56% larger. Finally, we successfully model a 225x225 grid with this 23 frames rate on 16 simulations nodes and two graphical ones.



number of nodes	*All rendering nodes, 2 visualization nodes*	*separation rendering/pre-rendering, 2 visualization nodes*	*separation rendering/pre-rendering, 4 visualization nodes*
2	7,3	13,1	16,3
4	12,4	18,5	17,8
6	17,5	22,1	22,6
8	21,3	27,0	25,3
10	24,6	32,4	27,9
12	27,7	36,8	29,1
14	30,2	38,2	31,5
16	31,1	42,8	32,1

Tab. 1 the frames per second with respect to the number of simulation nodes for a 160x160 grid.

These results show that as the scientific domains can draw advantages to develop in Virtual Reality environment, it becomes necessary to perfect the development tools in order to manage sophisticated simulations. A first way consists in reinforcing the link between simulation and visualization to reach new optimizations.

4.3. Visualization and simulation optimizations

In the scientific computations domain, numerous libraries of parallel implementations (ScaLapack or PETSc for instance) are available in order to easily develop efficient simulations. For example, in [4] the fluid motion is based on a PETSc solver in the place of our multigrid method. This approach is proved efficient in term of development facilities in the NetJuggler environment and in term of frames per second performances. We extended this work with our own multigrid implementation in order to reinforce the simulation control on the data-flow. This evolution allows us to precisely test the gain induced by interactions between the visualization and the simulation parts.

Our example consists in differentiating two grids discretization of the fluid finite representation from the simulation computation and the visualization. As the simulation can use an increasingly fine grid to optimize its accuracy, the rendering step conserves always the same discretization of the grid of interest. A new operation is defined which approximates the fluid solver results with a pertinent average of the density on a coarser rendering grid. This allows to reduce the communications to the rendering nodes. The simulation is always accurate but the rendering is simplified to a sufficient level for the user interpretation. So the communications are reduced while ensuring the rendering quality.

All the performances given in the tab.1 are a direct consequence of this optimization where the visualization uses an 80x80 grid to draw the fluid whereas the simulation solves the fluid motion on a 160x160 grid. In this configuration, the communications have been decreased by a factor of four.

This optimization is easy to design because of the control parallelism that we explicitly introduce in the NetJuggler platform. We can now plan to develop many interactivities with the user to reinforce the physical phenomena interpretation. This can be done with a return from the user to modify on-the-fly some visualization parameters or simulation parameters. The user has the possibility to specify an area of interest by a zoom and a new grid size at the same time for the fluid solver or the visualization to improve its simulation quality and its interpretation.

5. Conclusion

We experimented a fluid motion as a VR application on an original VR environment using cluster technology and some software tools as NetJuggler. To be in adequacy with the scientific domain, the NetJuggler platform has

been used with the integration of a parallel program based on control parallelism. Furthermore, to optimize the frames rate, we realized an explicit separation of the rendering and the simulation nodes.

The performances of the fluid motion using multigrid solver demonstrated the great advantage to this development approach. But it has two drawbacks: the separation of rendering and simulation nodes comes from a trick in NetJuggler and the control parallelism has to be entirely managed by an advanced programmer. We need an extension of NetJuggler in order to provide a development environment which can ease the implementation of such sophisticate simulations. This new platform would help any user to build such applications by proposing a structured environment to express the control parallelism and the interaction between the different simulation and visualization components.

If we have illustrated the gain of an ad-hoc coupling between the simulation and the visualization, a second important aspect which has to be developed consists in offering a very rich interaction with the user. Therefore the interpretation of the physical phenomena is improved and the simulation and the visualization can be directly influenced by a return information.

In this sense the multigrid algorithm offers a lot of possibilities to take into account solvers of different complexities according to the fluid turbulences. The *M* elementary method, the iteration numbers of these different *M* elementary methods in the multigrid algorithm or the deepness level in the error estimation are parameters which can be modified by a return information of the user. The fluid motion application we studied here could really benefit from the integration of these improvements in the proposed NetJuggler extension.

References

1. C. Elliot, G. Schechter, R. Yeung, and S. Abi-Ezzi: "TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications," *proceedings of ACM SIGGRAPH '94*, pp.421-434 (1994).

2. J. Allard, V. Gouranton, G.Lamarque, E. Melin, and B. Raffin "SoftGenLock: Active stereo and Genlock for PC cluster," *IPT/EGVE'03 Workshop* (2003).

3. J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin "NetJuggler Running VRJuggler with Multiple Displays on a commodity component cluster," *IEEE VR* (2002).

4. J. Allard, V. Gouranton, E. Melin, and B. Raffin "Parallelizing Pre-Rendering Computations on a NetJuggler PC Cluster," *Immersive Projection Technology Symposium, Orlando* (2002).

5. A. Bierhaum, C. Just, P. Hartling, A. Baker, and C. Cruz-Neira "VRJuggler: A virtual platform for virtual reality application development," *IEEE VR* (2001).

6. W. Hackbusch, "Multigrid methods and Applications", *Springer, Berlin*, (1985).

7. G. Humphreys, and P. Hanrahan "A distributed graphics system for large tiled displays," *IEEE Visualization'99* (1999).

8. S. Madougou, and J. Vairon, "Fast deployment of a commodity VR cluster", *VR-Cluster'03 Review* (2003).

9. J. Stam "Real-time fluid dynamics for games," *Proceedings of the game developer conference* (2003).