WP 5 Project Deliverable D5.2

Interactive Field Simulation Techniques; Solver and Data Flow Parallelization

Project Number	IST-2000-29266
Project Title	Virtual Real Time Fire Emergency Simulator
Deliverable Type	Report
Deliverable Class	Restricted

Deliverable Number	D5.2
Title of Deliverable	Interactive Field Simulation Techniques; Solver and Data Flow Parallelization
Nature of the Deliverable	Report
Contributing WPs	WP 2, WP 5
Contractual Date of Delivery	30. September 2002
Actual Date of Delivery	31. October 2002
URL	www.virtualfires.org
Authors	Christian Redl (CD), Björn Sjögreen (KTH)
Contact Details	Institute for Structural Analysis / SiTu Research Univ. Prof. Dipl.-Ing. Dr. techn. Gernot Beer Lessingstrasse 25/II 8010 Graz / Austria Tel.: +43 316 8736180 Fax: +43 316 8736185 Email: gernot.beer@ifb.tu-graz.ac.at

Abstract	The objective of WP5 Task 2 is to study the feasibility to execute the CFD code concurrent with the visualization
Keywords	CFD, parallelization, one dimensional modelling

1 Introduction

The solver is adapted for execution on parallel super computers with distributed memory. The distributed memory model allows better scalability than, for example, parallelization by threads on a shared memory computer. Modern super computers have a number of processors, which typically ranges from hundreds to thousands.

The distributed memory model also allows the code to be run on clusters of workstations or PCs, in case no super computer is accessible.

The performance of a parallel solver depends both on the computational speed of each of the processors, and on the speed of the network. The performance of the network is usually modelled by two parameters, a fixed communication start-up time and a send time in bytes per second. A super computer has a dedicated network, which gives much faster communication speeds than the typical network connecting a cluster of workstations

To make a program execute on a parallel computer with distributed memory, the programmer has to write the program so that the work is divided between the processors. Doing this is very problem dependent, some problems are better suited for parallel execution than others. At some points in the code where data from other processors are needed, communication is done by inserting calls to library routines. There exists a large number of communication libraries, many of them vendor specific. A well-standardized and portable communication library is the Message Passing Interface, (MPI). MPI is probably the most popular communication library in use today.

2 Parallelization of the ICE code

The ICE code, which will be used in the Virtual Fires project, is a flow simulator built on the Lattice Boltzmann method for solving the equations of incompressible fluid flow with combustion. The code solves the fluid flow equations in three-space dimension. The code advances the flow in time using explicit time stepping, which is well suited for parallelization. During one time step, the data at one point of the computational domain, only depend on the data at the neighbouring points. Thus all communication will be local, and the time it takes to do communication will not increase when the number of processors is increased.

The parallel version of ICE uses the MPI library for communication.

In its present version, ICE solves the equations on a box-shaped domain. The geometry of the tunnel is specified by flagging some points as being inside the tunnel, and other points as being outside. A sketch of the situation is outlined in Fig. 1.

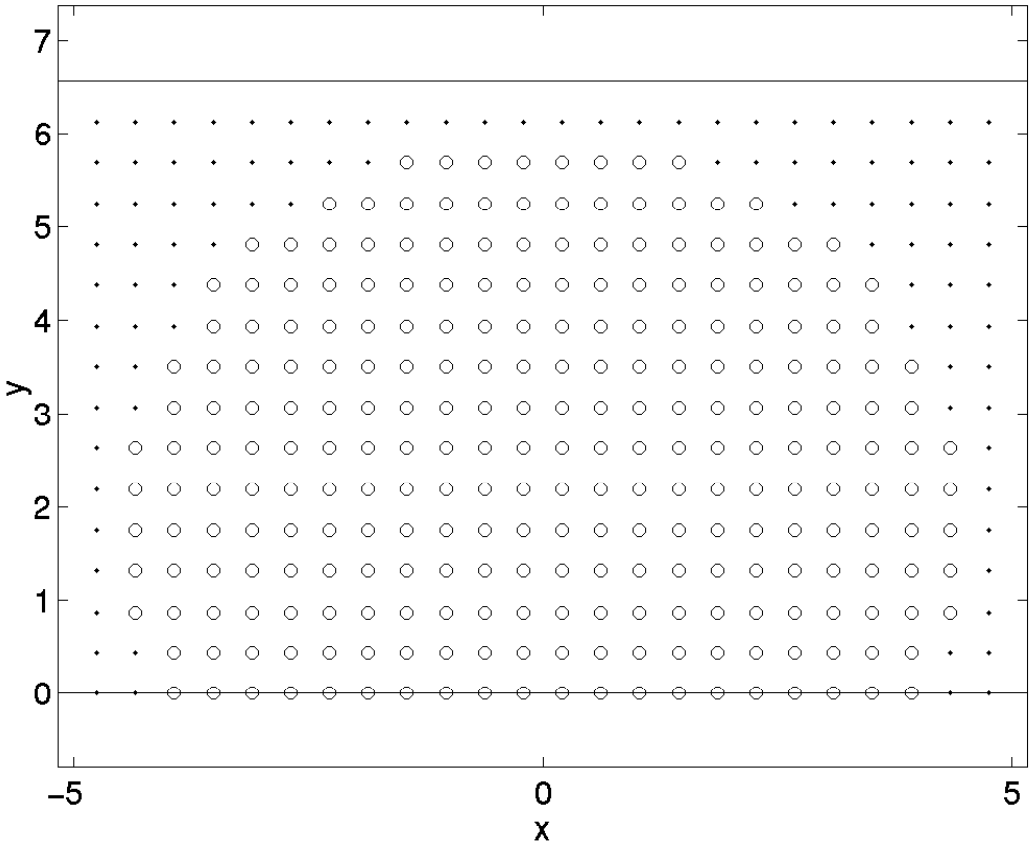

Figure 1. *Cross section of tunnel, as represented in the CFD code. Circles are grid points inside the tunnel, used for the simulation. Dots are inactive points, not used in the computation.*

For execution on *p* processors, the domain is divided into *p* smaller boxes, and one small box is handled by each processor. The sub domains should be of equal size so that the computational work becomes equally distributed among the processors. The division of the domain can be done along all coordinate directions as in Fig. 2, or only along one direction as in Fig. 3. Both ways of dividing the domain are currently implemented in ICE. In the final version, the program will automatically choose the most efficient domain decomposition.

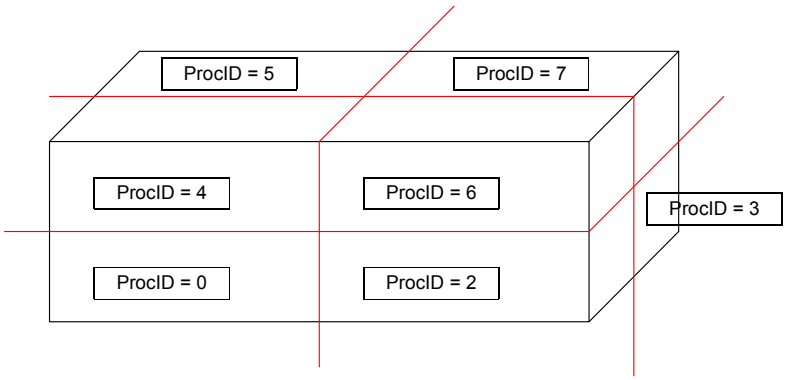

Figure 2. *Domain decomposition along one coordinate direction.*

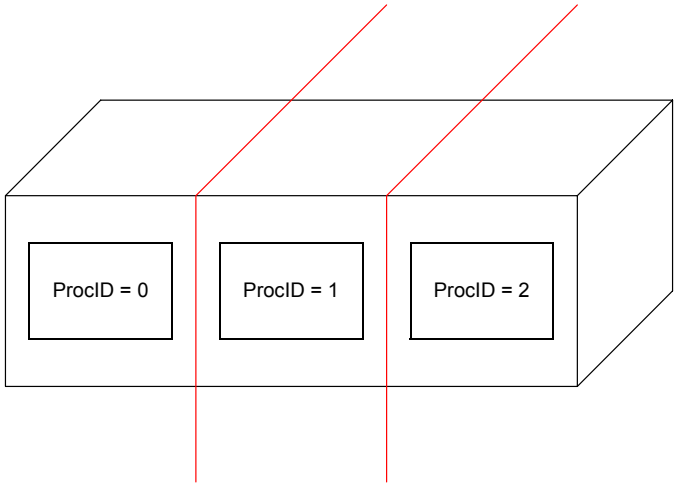

Figure 3. *Domain decomposition along three coordinate directions.*

More advanced domain decompositions could also be thought of by taking into account that the parts of the domain that are outside the tunnel walls carry a smaller workload than the parts that are inside the tunnel.

Algorithm 1 roughly describes the steps executed within the parallel solver. A graphical representation of the program is given in Fig. 4, where we also outline the interaction with the database described in Section 4.

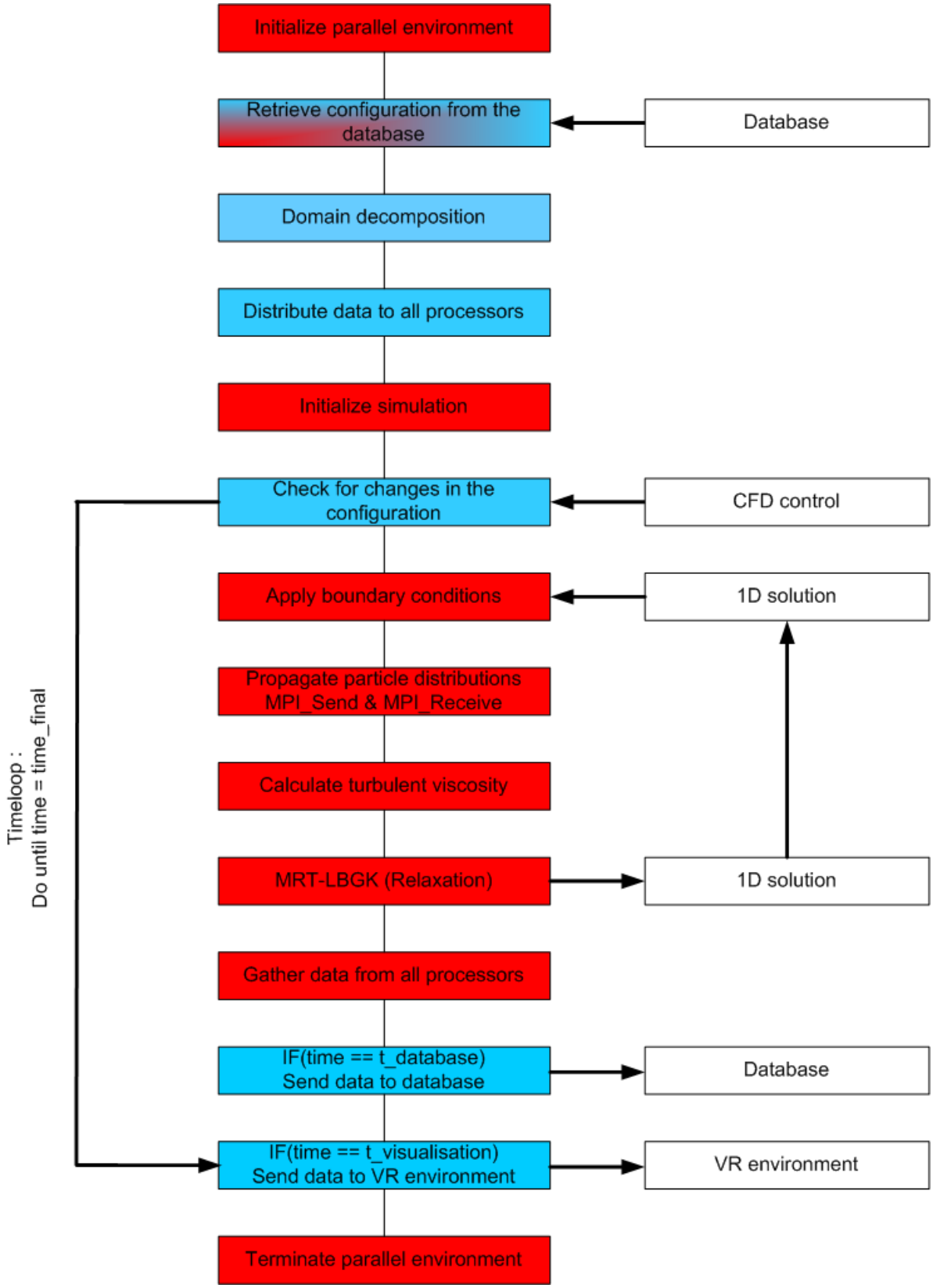

Figure 4. *Flow chart of CFD solver and its interaction with database and CFD control. Red boxes means that only the master processor is involved, whereas blue boxes stresses that all processors are involved.*

```
Program VirtualFires

!---Include the MPI library
INCLUDE "mpif.h"
!---Initialise MPI
CALL MPI_INIT(IERR)
!---Communicate the number of processors
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NP)
!---Every processor finds its ID
CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYID)
!---Processor 0 performs the data input
data_input : IF(MYID == 0) THEN
!---Retrieve geometry data and perform the domain decomposition
     RETRIEVE Geometry FROM database
     DO domain_decomposition
     DISTRIBUTE subdomain_data
!---Read case file, boundary definitions and initial data
     RETRIEVE Case_definition FROM database
     RETRIEVE Boundary_definition FROM database
     RETRIEVE Initial_data FROM database
     DISTRIBUTE case and data
end data_input : ENDIF(MYID==0)
!---All processors perform the calculation
solve_equations : DO time = 1, number_of_timesteps
     CHECK IF devices are activated by the user in the VR environment
     Update the solution at all grid points in the interior of the domain
     Impose boundary conditions at boundary points.
     Communicate points at the interprocessor boundaries.
!---At distinct times the data should be sent to the data base
     dump_results : IF(time==write_result_time) THEN
!---All data are gathered by Processor 0
       CALL MPI_GATHER(data)
!---Processor 0 send the data to the data base via a Unix socket
       IF(MYID == 0) CALL SENT_DATA_TO_DATABASE
     end dump_results : ENDIF(time==write_result_time)
end solve_equations END DO
!---Finalize the parallel execution
CALL MPI_FINALIZE(IERR)

END Program VirtualFires
```

Algorithm 1. *Outline of the VIRTUALFIRES parallel solver.*

In its present version, the ICE code can efficiently solve flows in simple tunnels. More complex situations with several tunnels, some of them intersecting, subway stations, etc, can be computed with the code, but not efficiently. The situation can be improved by introducing multi-block capabilities into the code. This is a major undertaking, which should be considered only after the present version is fully operational.

3 Increased efficiency by one-dimensional modelling

The dimensions of a tunnel are such that it is not realistic to have full numerical resolution all the way from the entrance to the exit. Furthermore, the flow will be approximately one-dimensional in the sections of the tunnel, which are far away from the fire.

It is therefore a natural idea to try to use a simplified description of the physics far from the fire. The simplified description will be such that it can be solved with small computational effort.

In [1], such an idea is presented in the context of computing blood flow in vessels in the human body. In [1], a simplified one-dimensional model is obtained by averaging the three dimensional incompressible Navier-Stokes equations across the sections of the tube. The model resembles the one-dimensional Navier-Stokes equations, but the viscous effects are modelled by a term *K**velocity in the momentum equation, with *K* a problem dependent parameter. The boundary conditions at the interface between three-dimensional and one-dimensional regions are discussed in [1]. Stability at the interface is proved for a simple and straightforward boundary condition. We therefore do not expect any difficulties from the three-d/one-d boundary conditions.

One additional problem for the Virtual Fires project is that the ICE code uses the Lattice Boltzmann method, so that the macroscopic flow variables are not directly available. Instead a number of (nineteen) particle velocities are stored as dependent variables. However, a mapping is available in the lattice Boltzmann model, which can be used to convert between the macroscopic variables in the simplified model and the microscopic variables in the lattice Boltzmann code. We therefore believe that the variable conversion problem can be solved without too much difficulty by use of the above mapping.

Another approach is presented in [2], in the context of simulating fires in buildings. The flow field in one room is represented by two states, one upper hot state near the ceiling, and one lower cold state near the floor. The thermodynamical quantities are constant in space, but time dependent. They are obtained by solving for the combustion as a system of ordinary differential equations. We can think of this as a zero dimensional model, with two zones.

A possible model would be to combine the results in [1] and [2] and use two one-dimensional equations in regions far from the fire. Work with testing one-dimensional models in a simplified setting is underway at KTH.

4 Data Flow Parallelization

A rough outline of the visualization-simulation system is displayed in Fig. 5. The database has a process, which communicates with the solver and with the visualization program. The database reads and writes data from/to a file.

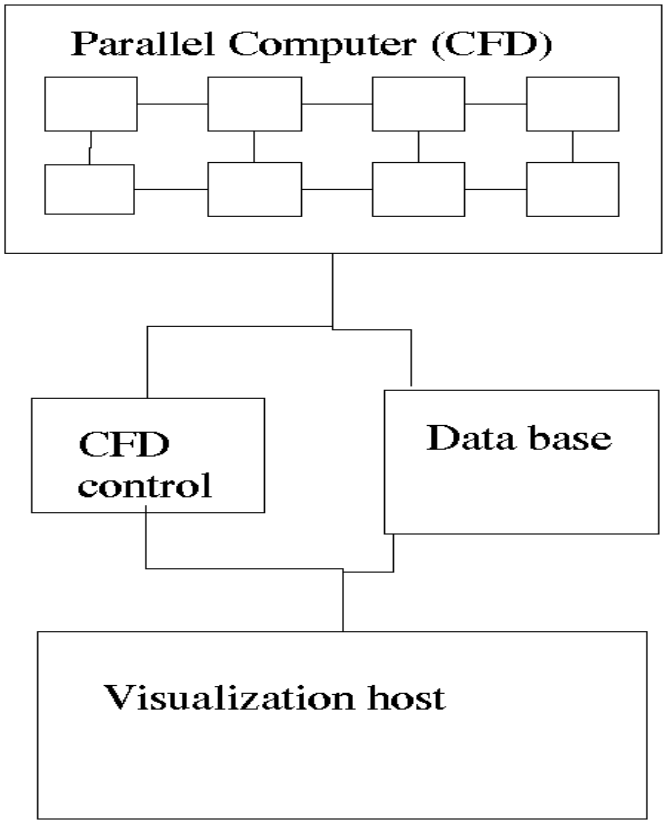

Figure 5. *Basic data flow in simulator*

The solver sends data to the database process at regular intervals. This can be done either by letting each processor send its part of the domain separately to the database, or by collecting the entire flow field in one of the solver processors, and then sending it to the database from that single processor. The database process can normally only receive from one solver process at the time, so there will be no parallel speedup by letting the solver processors send in parallel. We therefore take the second approach, since it gives a cleaner separation between the solver and the database.

For the communication between the database and the solver, MPI is not used. Instead normal networking, such as Unix sockets probably with Corba on top is used. There will then be a complete freedom of where the database should run.

The database keeps the latest received flow field in memory, so that it can be quickly sent to the visualization process.

We have made some preliminary timing on send/receive operations between a CFD program, and a process on the visualization host. The model described above, gathering of the flow field to one process, and sending it out on one socked was used. Corba was not used. The tests are of a very preliminary character. For data of size 9Mbyte to 140Mbyte, we found that

- The time to gather and send is fairly insensitive to the number of processors in the interval 1-40, (40 was the largest number of processors used).

- The wall clock time for the send operation varies considerably from one run to another, and lies in the range from 1 to 10 times the time given by the speed of the network.

- For the CFD code tried, which is not the code that will be used in the Virtual Fires project, the time to gather and send lies in the range 1 to 10 times the cost of performing one time step.

- The time to gather the flow field in one processor is smaller, usually much smaller, than the time to send the flow field to the display host.

It will not be necessary to send every computed time step to the database. Sending every time step will seriously degrade the performance of the solver. The stability limitation of the numerical method will enforce a smaller time step, than what is necessary for the visualization, unless the resolution in space is very coarse. The code saves every *n*th time step, where *n* can be set from the user interface.

Increased efficiency by solving for the flow only near the visualized region is probably not possible. The flow has to be computed at the places where it is changing, independently of whether it is being visualized at those places or not. However, some efficiency will be gained in the visualization by only displaying a part of the computational region. The selection of the subset of points needed for display would be best handled by the data base process. It can be done very simply by selecting a sub cube instead of the entire computational domain.

5 Time evolution

The database stores the time steps one by one in sequence. It will be possible for the user to stop the computation and select a previous time frame, change conditions there (e.g., add water, insert fan), and restart the computation from that point in time. We think about this as a tree of possible time evolutions, where each branch of the tree represents one computation with fixed conditions. See Fig. 6 for an example. For speedy treatment of time histories, each branch should be stored as a unit, in the database.

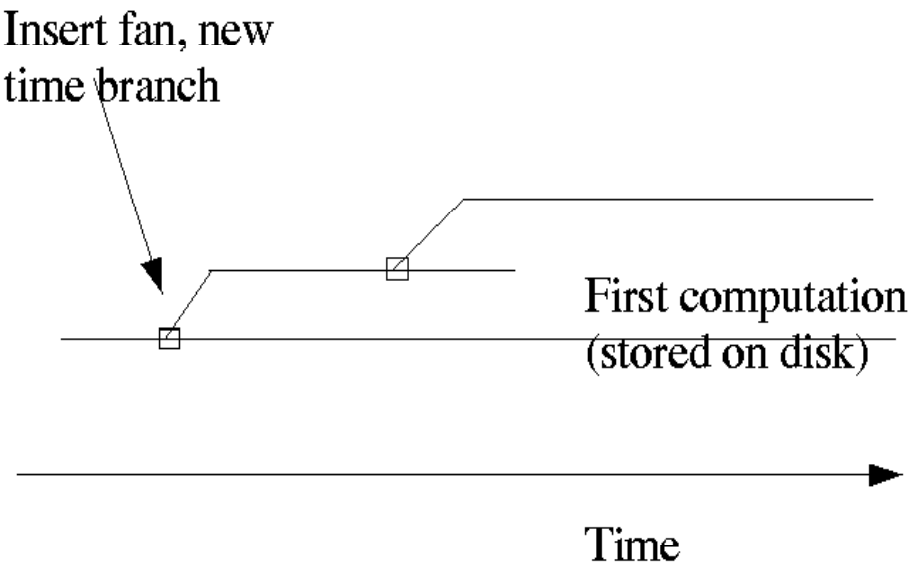

Figure 6. *User actions cause branching in the computational history.*

In order to control the computation as described above the visualization program, which is where the user interface runs, will send both requests (run, stop), and objects (car arrives, fan activated) to the CFD solver. In Fig. 5, this is represented by the CFD-control component.

6 References

[1] L.Formaggia, J.F.Gerbeau, F.Nobile, and A.Quarteroni
On the coupling of 3D and 1D Navier-Stokes equations for flow problems in compliant vessels
Comput. Meth. Appl. Mech. Engrg. 191 (2001), pp.561-582.

[2] W.W.Jones, G.P.Forney, R.D.Peacock, and P.A.Reneke
A Technical Reference for CFAST: An Engineering Tool for Estimating Fire and Smoke Transport
Report NIST TN 1431(2000), National Institute of Standards and Technology, US Department of Commerce.