



REVIEW OF TECHNOLOGIES FOR  
INTERACTIVE, DISTRIBUTED AND  
PARAMETER STUDY SIMULATIONS ON  
A GRID

WP5

---

Document Filename: **CG-5-TAT-IntersimRev-001-FINAL.doc**

Work package: **WP5**

Partner(s): **Cyfronet**

Lead Partner: **Cyfronet**

Config ID:

Document classification: **Confidential**

---

**Abstract:** The aim of this document is to review the approaches which were used to put distributed and interactive applications on the Grid. It presents the existing tools and solutions like Cactus or Nimrod and evaluates their appropriateness for CrossGrid WP1 applications.



---

**Delivery Slip**

	<b>Name</b>	<b>Partner</b>	<b>Date</b>	<b>Signature</b>
<b>From</b>	Katarzyna Zając, Marian Bubak, Maciej Malawski	Cyfronet	10.03.2002	
<b>Verified by</b>				
<b>Approved by</b>				

**Document Log**

<b>Version</b>	<b>Date</b>	<b>Summary of changes</b>	<b>Author</b>
1-0-DRAFT-A	10/3/2002	Draft version	Katarzyna Zając, Marian Bubak, Maciej Malawski
1-0-DRAFT-B	21/5/2002	Proofreading	Katarzyna Zając, Marian Bubak, Maciej Malawski, Piotr Nowakowski

## CONTENTS

<b>1. INTRODUCTION</b>	<b>5</b>
<b>2. EXISTING INTERACTIVE SIMULATION &amp; VISUALISATION ENVIRONMENTS</b>	<b>6</b>
2.1. CSE - COMPUTATIONAL STEERING ENVIRONMENT	6
2.1.1. <i>Short description</i>	6
2.1.2. <i>Conclusions and ideas useful for Grid environment</i>	6
2.2. CUMULVS	6
2.2.1. <i>Short description</i>	6
2.2.2. <i>Conclusions and ideas useful for Grid environment</i>	7
2.3. VIPER	7
2.3.1. <i>Short description</i>	7
2.3.2. <i>Conclusions and ideas useful for Grid environment</i>	8
2.4. VISIT	8
2.4.1. <i>Short decription</i>	8
2.4.2. <i>Conclusions and ideas useful for Grid environment</i>	8
<b>3. INTERACTIVE SIMULATION IN DATAGRID</b>	<b>9</b>
3.1. SHORT DECRPTION	9
3.2. CONCLUSIONS AND IDEAS USEFUL FOR GRID ENVIRONMENT	10
<b>4. GRID SOLUTIONS FOR INTERACTIVE ENVIRONMENTS</b>	<b>11</b>
4.1. PARALLELIZATION FOR METACOMPUTER VS HIGH PERFORMACE COMPUTING.	11
4.2. EXAMPLS OF APPLICATIONS BENEFITTING FROM GLOBUS TECHNOLOGIES	11
4.2.1. <i>NASA's overflow-d2</i>	12
4.2.2. <i>Numerical relativity (Cactus application)</i>	12
4.2.3. <i>Remote access to climate simulation data</i>	12
4.2.4. <i>X-Ray CMT application</i>	12
4.3. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS	13
<b>5. CACTUS PROBLEM SOLVING ENVIRONMENT</b>	<b>14</b>
5.1. OVERVIEW	14
5.2. PORTABILITY	14
5.3. DISTRIBUTED PARALLEL SIMULATIONS	14
5.4. COMMUNICATION OPTIMIZATION USED IN CACTUS	15
5.5. MULTILANGUAGE SUPPORT	15
5.6. MODULARITY	15
5.7. CHECKPOINTING DISTRIBUTED SIMULATIONS ON THE GRID	16
5.8. UNIFIED I/O INTERFACE	17
5.9. REMOTE ONLINE DATA STREAMING AND VISUALIZATION	17
5.9.1. <i>HTTP control interface</i>	18
5.9.2. <i>Socket connection</i>	18
5.10. REMOTE MONITORING AND STEERING	18
5.10.1. <i>Thorn HTTPD</i>	18
5.10.2. <i>Stream driver</i>	19
5.11. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS	19
<b>6. NIMROD- A TOOL FOR PERFORMING PARAMETERISED SIMULATIONS</b>	<b>20</b>
6.1. LIMITATIONS OF NIMROD AND MIGRATION TO NIMROD/G	20
6.2. NIMROD/G ARCHITECTURE	20
6.3. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS	21
<b>7. HOW TO USE THESE IDEAS FOR THE CROSSGRID APPLICATION ARCHITECTURE</b>	<b>22</b>
7.1. REDESIGN EXISTING PARALLEL CODE	22

---

7.2. USE EXISTING GRID SOLUTIONS FOR CONNECTING SIMULATION AND VISUALISATION	22
7.3. USE EXISTING GRID SOLUTIONS FOR PARAMETER STUDY	22

## 1. INTRODUCTION

The aim of this document is to show the ideas for making better use of Grid technologies by the CrossGrid applications. CrossGrid applications can be divided into two groups:

- Interactive parallel simulations
- Parameter study simulations

Both types of applications are connected to a visualization kernel.

Most of the applications consist of parallel code designed for clusters and high performance computers and few of them are going to be parallelized.

The problems are:

- How to parallelize simulation algorithms so that good performance can be achieved in a distributed Grid environment (is there a need for running parallel simulations in a Grid distributed environment)?
- How to make applications fault tolerant? (we cannot assume that the Grid is faultless.)
- How to effectively connect simulation with visualisation?

The first Chapter presents existing tools used for connecting visualizations and interactive simulations via a network. Their architectures are presented and the possibility of porting them to a Grid environments is evaluated.

The second Chapter presents DataGrid efforts regarding interactive simulation. Since the focus is on using large data sets, this aspect is not a main subject which needs to be developed soon.

The third Chapter presents existing applications that have already made use of Grid technologies. We also describe techniques for porting parallel code, so as to make it efficient in a distributed environment, with high latency times and relatively low bandwidth.

The next two Chapters presents two tools build over Globus. The first one is Cactus, which is a flexible problem-solving environment. The Cactus code was originally developed to provide a framework for the numerical solution of Einstein's equations, but it has since evolved into a general-purpose, open-source environment that provides a unified, modular and parallel computational framework for scientists and engineers. It supports a mechanism useful in fault-prone environments like the Grid – namely, checkpointing. The other tool discussed here is Nimrod/G; a tool for performing parameterized simulations over networks of loosely coupled workstations.

The last Chapter proposes some application ideas for the solutions presented earlier.

---

## 2. EXISTING INTERACTIVE SIMULATION & VISUALISATION ENVIRONMENTS

This Chapter presents the environments that were developed in order to interactively connect scientific simulations with visualisations. They are not explicitly designed for Grid computing, but some of the ideas may be useful when designing the architecture of CrossGrid applications.

### 2.1. CSE - COMPUTATIONAL STEERING ENVIRONMENT

#### 2.1.1. Short description

The CSE [Cse] is developed at the Department of Interactive Systems, Centrum von Wiscunde en Informatica, Amsterdam. The project team remains in close cooperation with the visualization group at the Netherlands Energy Research Foundation, ECN. One aim of the CSE is to provide scientific end users with an environment in which they can easily define interactive interfaces to ongoing simulations.

The CSE's architecture is implemented as a set of processes - called *satellites* - which implement standard visualization operations. The simulation is also packaged as a satellite. Satellites cooperate by sending and receiving data from a central *data manager* which, in turn, notifies all interested satellites about data mutations. The most predominate satellite is the *PGO* graphics editor, which allows the end user to sketch out visualizations. A two-way binding between visualization and data is achieved by binding the sketch to data within the data manager. CSE uses the TCP/IP protocol for Ethernet transport.

#### 2.1.2. Conclusions and ideas useful for Grid environment

The main disadvantage of the CSE is its centralization, which hampers its scalability in Grid environments. However, the idea of a data manager and satellites can be somehow extended (i.e. by building hierarchical or distributed data sets).

## 2.2. CUMULVS

### 2.2.1. Short description

CUMULVS [Cum] is a project at the Oak Ridge National Laboratory; part of the Advanced Computational Testing and Simulation Toolkit.

CUMULVS is an infrastructure library that allows a programmer to add interactive steering and visualization to an existing parallel or serial program. From the application code perspective, the setup is very simple. After describing how data fields are spread across processors, and which parameters are adjustable via the front-end, the program makes a single subroutine call in the body of the main loop. This call directs when CUMULVS can communicate with and send data to attached viewers. The return status from this subroutine can be queried to determine if the viewer program has adjusted (steered) any parameters. CUMULVS guarantees that all nodes in the parallel program will receive updated parameters at the exact same timestep, as long as all nodes call the interface subroutine at each timestep. The subroutine consumes very little overhead when no viewers are attached (the cost is then reduced to a single message probe). CUMULVS "understands" standard HPF style block and block or cyclic decompositions as well as particle decompositions. The system allows a viewer to examine data in a parallel program as if it were available as a large monolithic array. The software manages all the details of which data to pack and send to a front-end and all the details of attaching a viewer to a running program.

CUMULVS also allows an application program to perform user-directed checkpointing and automated restarts of parallel programs using checkpointing, even across a heterogeneous cluster of machines. A single user library interface routine passes control to CUMULVS periodically, to transparently handle

the viewer attachment/detachment protocols, the selection and extraction of data, and the updating of steering parameters. CUMULVS allows each front-end viewer to interactively select the granularity and extent of data that it desires to view.

### **2.2.2. Conclusions and ideas useful for Grid environment**

Currently, CUMULVS uses PVM as its message passing substrate; it allows for pairs of anonymous tasks to communicate with each other without both tasks being started at the same time. MPI does not allow these dynamics, so porting CUMULVS ideas to MPI would not be easy.

In the past (1999), several experiments were conducted to explore the feasibility of porting CUMULVS to the Globus/Nexus environment as an alternative to the PVM message-passing substrate. There were plans to extend CUMULVS to provide better support for MPI applications which use Mpich-G on Globus. However, the most recent version of CUMULVS supports only the PVM library.

Although the software itself is quite new (the modifications of several source files date to April), its documentation has aged (a 1996 user guide).

The obvious advantages of this tool are its fault tolerance and checkpointing ability. One major disadvantage is the lack of support for MPI.

## **2.3. VIPER**

### **2.3.1. Short description**

VIPER (Visualization of massively Parallel simulation algorithms for Extended Research) [Tum] is a conceptual prototype for on-line visualization for parallel simulation algorithms developed at the Technical University of München.

VIPER propagates an object-based view of the whole application cycle. The so-called objects are built from distributed and replicated data structures of the massively parallel simulation algorithm. These data structures are based on parameters of the mathematical model and parameters of the numerical methods.

VIPER breaks down the structure of an application cycle into three distinct logical components: computational units, connectivity units and visualization units, which compose a pipeline architecture. With regard to this pipeline architecture, VIPER follows a new DUAL SERVER model, which is based on a client/server/client architecture. One client is the massively parallel simulation algorithm, the other client is the visualization system. On request, DUAL SERVER executes different kinds of services for both clients. The processes involved are split into three process spaces. The processes of the massively parallel simulation algorithm run on a remote parallel computer system which is a multiprocessor system or a cluster of workstations connected via WAN or LAN. The processes of the visualization system run on a powerful graphical workstation. The processes of the distributed application DUAL SERVER run on processors which reside in a LAN and on a remote parallel computer system in a WAN or LAN. To enable online visualization, the DUAL SERVER offers an infrastructure to extract data out of the computational unit during simulation time, to transfer data across a WAN (or LAN) and to hand out data to the visualization unit.

RPCs (remote procedure calls) and the XDR protocol are used to implement interprocess communication between WAN- and LAN- distributed processes of the DUAL SERVER. Again, RPCs are used to hand out the transferred data to the visualization unit. .

Because of the object-based view of VIPER, interaction is similar to execution of operators on objects. To steer the simulation process, a modification operator is applied to the parameters of the mathematical model or to the parameters of the numerical methods. In order to control the data being

---

extracted, transferred, and handed out by the DUAL SERVER, relevant parameters need to be switched on or off.

To process and re-process scientific data, the visualization unit maintains a database which holds data from the distributed and replicated data structures of the massively parallel simulation algorithm, handed out by the DUAL SERVER.

All parameters are typed (e.g. scalar, matrix, physics, grid) and built as objects by declarations, via source code instrumentation in the simulation algorithm. Each object (parameter) is associated with so-called synchronization points (SPs). These synchronization points are executed during simulation time. Executing a synchronization point in a computational unit means sending a signal (request) to the DUAL SERVER. As already known, the DUAL SERVER extracts data out of the computational unit, provided that the synchronization point and the relevant objects are switched on.

### 2.3.2. Conclusions and ideas useful for Grid environment

The architecture of the DUAL SERVER is quite interesting for development in a Grid environment. Using VIPER-based concepts allows application developers to achieve good scalability. However, according to VIPER Web pages, only a prototype implementation exists and it appears that the project has been discontinued.

## 2.4. VISIT

### 2.4.1. Short description

VISIT [Vis] is a project developed at the Juelich Research Centre, Central Institute for Applied Mathematics, Germany

VISIT is a library for point-to-point communication between two independent applications (like a simulation and a visualization) using a client-server model. Bundled with VISIT is a simple name service called *seap* (service announcement protocol). A visualization can register its service(s) on a *seap*-server. The simulation can then query this information (consisting of a hostname and a port number) and use it to connect to the visualization. The current implementation of VISIT uses TCP/IP sockets for connecting simulations and visualizations. Besides that, it is also possible for the simulation to write data to a file, which a visualization can then read, using the same send/receive calls as when a socket connection is established. This is intended to be used for offline visualizations, where the simulation data is recorded in advance. The simulation-related API is reduced to a few function calls. Language bindings are available for C, FORTRAN, and Perl. On the visualization side, VISIT has language bindings for C and Perl. The distribution contains demo clients and servers written in all the supported languages. A complete server for AVS/Express is also included.

### 2.4.2. Conclusions and ideas useful for Grid environment

VISIT is a very simple tool. The data transfer is over IP. No parallel libraries are supported, no security mechanisms included and no sophisticated error-recovery protocols provided for. In addition, VISIT only supports peer-to-peer connections.

However, the idea of thinking about a visualization module as a server which can be registered as a service is very similar to OGSA. Some solutions can be borrowed from VISIT while designing the CrossGrid architecture.

### 3. INTERACTIVE SIMULATION IN DATAGRID

#### 3.1. SHORT DECRPTION

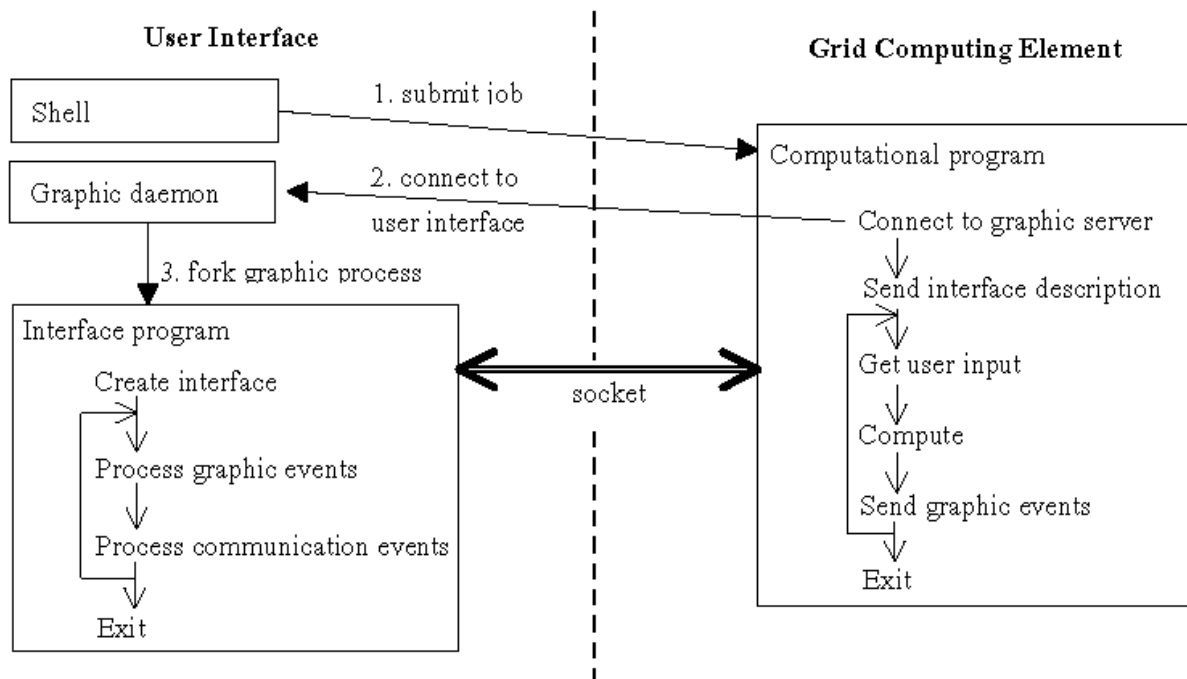
Although concentrated on HEP computing, the DataGrid application layer also contains biomedical and earth observation applications, some aspects of which correspond to long-range objectives that are outside the scope of this project.

The brunt of the effort involved in developing DG applications goes towards enabling intensive use of data produced from spaceborne or medical instruments. [Edg] proposes a design of an interactive simulation architecture on the Grid, destined for the biomedical application.

The proposed architecture is very general - its body consists of an infinite loop:

1. the user may enter some interactive input;
2. the algorithm progresses depending on its current internal state and the user input;
3. the user visualizes the current state through a graphical interface.

Figure 1 shows the general architecture of an interactive application running on the Grid. The User Interface machine operates a graphic daemon which awaits connections on a given port. The user first submits an interactive job using the DataGrid middleware interface (1). Once in the execution phase, the remote process connects back to the graphic daemon of the User Interface (2). The graphic daemon then spawns a new graphic process (3) to handle interaction with the computational process. A socket is opened between the two processes. The interface program creates a graphic interface and then enters a loop where it successively processes graphic events and communication events until it receives an exit message from the computational program. The computational program enters an interactive loop already described above.



**Figure 1 DataGrid architecture for interactive simulation [Edg].**

### **3.2. CONCLUSIONS AND IDEAS USEFUL FOR GRID ENVIRONMENT**

As can be seen, the connection between visualization and simulation is arranged through a socket library which is a low-level mechanism that does not provide advanced functionality. The idea is very general, since it does not relate directly to the main subject of the DataGrid project. Hence, it may be considered a good starting point, but the CrossGrid project should ultimately present techniques that would be more sophisticated.

---

## 4. GRID SOLUTIONS FOR INTERACTIVE ENVIRONMENTS

This Chapter presents various techniques that deal with the problems of bandwidth and latency in Grid environments. It shows examples of Grid applications already making use of these techniques.

### 4.1. PARALLELIZATION FOR METACOMPUTER VS HIGH PERFORMANCE COMPUTING.

The Grid can be viewed as a large distributed-memory parallel computer consisting of multiple (groups of) processors which exchange data across communication links. In parallel computers, a proprietary interconnection network is provided by a vendor; in clusters the links may be commodity networks such as Fast Ethernet. However, in distributed supercomputing the communication links are relatively slow, with high latency times and high potential for failure.

In [Fost] a range of decomposition techniques are presented to construct a distributed supercomputing application:

1. pipelining or dataflow decomposition
2. functional decomposition – it is useful when the simulation can be divided into parts, which execute more efficiently on different types of architectures.
3. domain decomposition – this approach is particularly difficult to parallelize in Grid environments, especially with a tightly coupled simulation.

In the last case, one can think of dividing the problem into parts which do not need to communicate with each other very often (i.e. a climate model can be divided into an atmosphere-related model and an ocean-related model). Another useful technique is overlapping communication and computation. (i.e the exchange of data between different tasks is carried out in smaller subdomains).

One example of an application ported from a classic parallel architecture to a distributed grid environment is the SF-Express[Fost] project which uses the Distributed Interactive Simulation (DIS) technique to model the behaviour and movement of hundreds or thousands of entities for military training, analysis and planning.

In order to effectively run the simulation in a distributed environment, the entities were distributed over a large number of simulation nodes. SF-Express uses a technique called *region of interest management* to reduce the amount of communication. The entities are assigned to nodes in a way that preserves physical locality as much as possible. The nodes are then organized into groups and a *router node* associated with each group is charged with propagating to other routers only those events, which may be of interests to their groups. For example tanks in Germany do not require information about naval vessels conducting training exercises in the Pacific. To summarise, SF-express consists of entity simulators, routers and data servers (to achieve uniform IO performance).

SF-Express makes use of the Globus infrastructure, namely it used the Globus Resource Allocation Manager to initiate an SF-Express computation so that the computation can be started from a single point. It also uses the Globus Heartbeat Monitor, which provides a mechanism for tracking the state of an SF-Express computation. The SF-Express developers have also experimented with the use of the Globus communication mechanisms, including Nexus/MPI. According to the appropriate Web pages, the project is over now, but it is nevertheless a good example of parallel simulation on the Grid.

### 4.2. EXAMPLES OF APPLICATIONS BENEFITTING FROM GLOBUS TECHNOLOGIES

#### 4.2.1. NASA's overflow-d2

This application simulates airflow around an airborne vehicle. It is an example of an experiment in which a large scale scientific application, developed for tightly-coupled parallel machines, is adapted to the distributed execution environment of the Information Power Grid (IPG). The core of the application is a computational fluid dynamics (CFD) algorithm. Globus is used as the enabling device for the geographically-distributed computation. Its performance was tested on three separate SGI Origin2000 machines: two located at the NASA Ames Research Center and the third at the Argonne National Laboratory. A maximum of 8 processors were used on any one machine, each running Globus version 1.0.0 and MPICH-G was used as the message-passing library.

In order to achieve good performance in the distributed environment the following techniques were used:

- Advanced partitioning - a graph partitioner was used to assign grids to processors in a way that balanced load and minimized communication;
- Overlapping communication & computation - latency tolerance was obtained by delaying the boundary value update by an additional timestep.

#### 4.2.2. Numerical relativity (Cactus application)

This application was described in [Cag] as an example of dealing with latency and bandwidth in a Grid environment. The article presents future directions in Cactus development. The present state of Cactus is described in the next Chapter of this document. Here, the techniques that are tested and going to be used in Cactus are presented. The test were performed on four supercomputers (three SGI Origins at NASA and one IBM Power-SP at SDSC). The measured bandwidth between the two sites was 3MB/sec, while the bandwidth between machines at each site was 100MB/sec.

The following techniques are described:

- increasing the number of z-direction ghostzones in parallel simulations, so they do not need to be synchronized very often,
- communication & computation overlapping,
- data compression for transmission across a WAN.

#### 4.2.3. Remote access to climate simulation data

In [Clim] an application dealing with climate simulation is described. It focuses on dealing with large data sets (utilizing the Globus Replica Manager) while interactive analysis and simulation are not distributed and are performed directly on the user's computer.

#### 4.2.4. X-Ray CMT application

[Xray] describes a pipelined data acquisition and on-line reconstruction system for Computer-Managed Tomography (CMT). The ability to reconstruct data rapidly and provide a medical scanner user with immediate feedback is valuable for user-shared facilities, since each user receives a limited operations quota. The inability to examine reconstructions derived from newly acquired data makes it difficult to undertake crucial decisions during the experimental period, resulting in poor use of scanners.

This application is designed for performing the reconstruction on-line. The data acquired from the scanner is first written to a local hard disk and then transferred via a 100 Mb/s network to the SGI Origin, where the preprocessing and reconstruction calculations are performed. The reconstruction results are saved to a hard disk and then loaded into the Visualization Engine. The Engine provides rendered images interactively to the scanner feedback monitor via a high-speed network, while also periodically writing rendered images to a web server, so they can be accessed with a WWW browser.

The paper mentions that Globus is used for performing calculations. Although these calculations were done on a single supercomputer the authors claim they are going to make the application Grid-compatible.

#### **4.3. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS**

This Chapter presented techniques used to redesign existing parallel code so that it can achieve good performance in a distributed environment. The reviews presented here show that various parallel applications are in fact being ported from classical high performance architectures to the distributed Grid architecture, using methods like:

- region of interests management (SF-Express)
- communication and computation overlapping (Overflow-2D, Numerical relativity)
- advanced load balancing (OverFlow-2D)
- data compression (Numerical relativity)
- infrequent synchronization (Numerical relativity)

Similar techniques can be used when parallelizing CrossGrid applications.

---

## 5. CACTUS PROBLEM SOLVING ENVIRONMENT

### 5.1. OVERVIEW

Cactus is an open-source problem solving environment designed for scientists and engineers. The Cactus code was originally developed to provide a framework for the numerical solution of Einstein's equations; one of the most complex sets of partial differential equations in physics.

Now, Cactus has evolved into a general purpose, open-source environment that provides a unified, modular and parallel computational framework for scientists and engineers.

The name Cactus comes from the design of a central core, which connects to application modules - or thorns - through an extensible interface. Thorns can implement custom-developed scientific or engineering applications, such as the Einstein solvers, or other applications such as computational dynamics. There exists a standard computational toolkit, consisting of thorns which provide a range of capabilities, such as parallel I/O, data distribution, or checkpointing.

The main features of Cactus, which might be useful, are:

- portability,
- a mechanism for building distributed parallel simulations,
- communication optimization techniques,
- multilanguage support,
- modularity that allows for collaboration between many code developers,
- a mechanism of checkpointing distributed simulations on the Grid,
- a unified IO interface,
- remote monitoring and steering of the simulation,
- remote online data streaming and visualization.

### 5.2. PORTABILITY

Cactus forms an additional layer on top of the Grid, providing a programming interface which allows the user to be ignorant of the nature of the machine or machines that the simulation runs on. The code enables access to Grid resources such as distributed I/O and parallelization across any number of supercomputers with precisely the same interface as it does to the resources of a single machine.

### 5.3. DISTRIBUTED PARALLEL SIMULATIONS

Parallelism is achieved by hiding the driver layer and features such as the communication and calling interface under a simple general-purpose API. The exact method used to provide parallelization (e.g. MPI, PVM) is not usually important for the thorn writer since the thorn driver provides routines, which are called by standard interfaces from the flesh. Sample routines are shown below:

- CCTK\_nProcs - returns the number of processors being used
- CCTK\_MyProc - returns the processor number
- CCTK\_SyncGroup - synchronizes a group of variables by exchanging the values held in each processor's ghostzones with the physical values of its neighbors
- CCTK\_Barrier - waits for all processors to reach this point before proceeding.

Different thorns can be used to implement different parallel paradigms, such as PVM, Pthreads, OpenMP, CORBA, etc. Cactus can be compiled with as many driver thorns as required (subject to availability), with the one actually used chosen by the user at runtime through a parameter file.

---

The standard thorn driver is currently PUGH (from the CactusPUGH package), which is a parallel Unigrid driver. PUGH provides distribution of computational grids over a number of processors, as well as grid decomposition, and parallelization.

By default, PUGH distributes the computational grid evenly across all processors. The computational grid can be manually partitioned in a regular way. Cactus also provides a thorn called PUGHInterp, which implements interpolation operators working on regular, uniform Cartesian grids. There is also a PUGHReduce thorn that registers a number of reduction operators within the flesh (such as the maximum or minimum of a computational grid variable). One disadvantage of the way in which parallelization is achieved in Cactus is its limited flexibility. The application programmer has the ability to parallelize problems using domain decomposition on a computational grid, which does not cover all parallelization techniques (such as parameter study or functional decomposition).

To the best of our knowledge, Cactus can be suitable for Lattice-Boltzmann methods (WP1.1). It is not clear, however, if this functionality is sufficient for WP1.1 as well as for other applications (i.e. the flood control application).

In order to perform distributed Cactus simulations on the Grid, the PUGH thorn is simply linked to the Grid-enabled MPICH-G implementation of MPI, which comes bundled with the Globus toolkit. Thus, preparing a Grid-enabled version of Cactus is a compilation choice, and it is completely transparent for application thorn programmers who wish to add their own code to a Grid-enabled Cactus. Using the Globus job submission tools, Cactus users can start their Cactus runs in a Grid environment just as easily as they do on a single machine.

#### **5.4. COMMUNICATION OPTIMIZATION USED IN CACTUS**

When operating in a metacomputing environment, one has to deal with different types of networks (shared-memory, distributed-memory, high-speed network, LAN, WAN/Internet) with different latency/bandwidth characteristics. Cactus has the ability to distinguish among these different types of network connections in one single distributed run and it can tune communication patterns adequately. For example, if a network has high latency times but also high bandwidth, many small messages can be coalesced into fewer larger ones.

MPICH-G2 already provides some of this functionality, as it can distinguish between processors located on one host (with native MPI installed) and processors separated by a LAN or WAN. Then, according to the locations of the computing nodes, MPICH-G2 can choose different protocols (TCP or vendor's MPI) for communication in one single distributed parallel run.

Another aspect is load balancing. Since different architectures provide different types of processors at different speeds, Cactus has the ability to decompose the whole computational problem into subproblems of different sizes, each of which matches a local processor's power.

#### **5.5. MULTILANGUAGE SUPPORT**

Cactus API supports C/C++ and F77/F90 programming languages for the thorns. This makes it easier for scientists to turn existing code into thorns, which can then make use of the complete Cactus infrastructure, and in turn be used by other thorns within Cactus.

#### **5.6. MODULARITY**

A thorn is the basic working module within Cactus. All user-supplied code goes into thorns, which are, by and large, independent of each other. Thorns communicate with each other via calls to the flesh API, plus, occasionally, via custom APIs of other thorns.

One of the key concepts which relates to thorns is the concept of implementation. Relationships among thorns are all based upon relationships among the implementations they provide. In principle, it should be possible to swap one thorn providing an implementation for another thorn providing that implementation, without affecting any other thorn. An implementation defines a group of variables and parameters, which are used to implement some functionality.

For example, the CactusPUGH/PUGH thorn provides an implementation driver. This implementation is responsible for providing memory for grid variables and for communication. Another thorn can also implement a driver, and both thorns can be compiled in at the same time. At runtime, the user can decide which driver thorn is used. No other thorn should be affected by this choice. When a thorn decides it needs access to a variable or a parameter provided by another thorn, it defines a relationship between itself and the other thorn's implementation, not explicitly with the other thorn. This allows the transparent replacement (at compile time or at runtime), of one thorn with another thorn providing the same functionality (with respect to other thorns).

## 5.7. CHECKPOINTING DISTRIBUTED SIMULATIONS ON THE GRID

The Cactus framework provides a cross-platform checkpointing mechanism. In general, checkpointing technology allows the user to freeze the state of an application by writing a checkpoint file to a disk, from which the application can be restored and continued at a later time.

In Cactus, a checkpoint is not just a memory image of the application written to disk (as in several other checkpointing systems), but rather the total set of user-defined objects (variables, scalars, etc.). While memory images tend to be quite large and are only compatible within the same class of operating systems and architectures, the Cactus approach allows for smaller, architecture-independent checkpoints. The crossplatform checkpoints of Cactus can be transferred between arbitrary architectures, operating systems and numbers of processors, while still making it possible to restart and continue simulations.

The checkpointing mechanism is transparent to the user, who can request checkpoints to be written at regular timestep intervals, at the end of the requested compute time allocation, or, through a steering interface, immediately at the current timestep. All of the internal technicalities of parallel I/O are concealed from the user.

The user can control checkpoint behavior (such as frequency of parallel I/O) by means of steerable parameters.

The checkpoint mechanism can create a single, global checkpoint file as well as multiple checkpoint files for each of the distributed machines. The mechanism makes use of parallel I/O wherever possible. When restarting, the multiple checkpoint files can be recombined into a single file, which can then be used to restart the simulation on an arbitrary set of machines. The parallel restart operation from multiple files is currently restricted to the same topology of machines. However, it is going to be extended in the future.

With respect to distributed simulations, a Cactus user has the ability to perform a distributed run and checkpoint this simulation even if it is being run on a heterogeneous machine set. A checkpoint file can then be transferred to a new configuration of machines to continue the simulation. The new pool

---

of machines can differ from the previous one in type and number of machines involved as well as the number of processors.

## 5.8. UNIFIED I/O INTERFACE

The flesh provides a runtime interface for arbitrary I/O thorns to register their own, specific I/O methods. These methods can then in turn be invoked by the user or by any application thorn to read external data into Cactus variables or dump their contents to a storage medium for postprocessing analysis and visualization purposes.

The I/O thorns currently available in the computational toolkit provide methods for writing simulation data in different formats (1D traceline plots, 2D slices and JPEG images, full N-dimensional arrays, arbitrary hyperslabs of N-dimensional arrays, reduction scalars (e.g. minimum/maximum values), isosurface geometry data, particle trajectories, runtime standard output), also using different I/O libraries (FlexIO, HDF5, JPEG, ASCII). Further methods or libraries can easily be added by thorn programmers. Cactus uses the Hierarchical Data Format version 5 (HDF5), which defines a flexible file format and provides a software library for managing arbitrary multidimensional datasets of various types. Raw data access is accomplished via a generic Virtual File Driver (VFD) layer in HDF5.

Beneath this abstraction layer exists a set of low-level I/O drivers, which provide different ways of accessing the raw data of an HDF5 file, either located on a local disk or on other storage media. One driver worth mentioning is the GASS (Global Access to Secondary Storage) driver that automatically downloads complete remote files to the local machine and then operates on their local copies via standard UNIX file I/O calls. This method is feasible for small or medium-sized data files.

For accessing data in very large files, as well as for accessing parts of files, Cactus adds its own drivers which enhance the HDF5 VFD layer with a driver that builds on top of the Data Grid software components from the Globus toolkit. Currently, the Data Grid client software only supports remote partial file access to Distributed Parallel Storage Systems (DPSS).

Files are uniquely addressed by their URLs, and after they are opened by the appropriate driver, all read and write operations are performed as network transactions on the Grid, transparent to the application. Individual timesteps and zones of interesting data can be read and visualized using the data selection capabilities of HDF5.

Cactus developers claim that remote access to files that are located on the Grid will soon be provided by a GridFTP driver that supports the standard FTP protocol, enhanced with partial file access, parallel streaming capabilities, and Grid security mechanisms.

They also plan to access and transfer distributed data sets as consistent single files, using a global address space with pointers to pieces located in various places (using the Globus DataGrid tools).

## 5.9. REMOTE ONLINE DATA STREAMING AND VISUALIZATION

Cactus also provides the ability to stream online data from a running simulation via TCP/IP socket communications. Currently, it is used for remote visualization. Multiple visualization clients can connect to a running Cactus executable via a socket from any remote machine on the Grid, then request arbitrary data from the running simulation, and display simulation results in real time. This can be done in two ways: by a HTTP control interface or through socket connections.

---

### 5.9.1. HTTP control interface

One of the methods mentioned above is accessing Cactus output files when they are being written by the running simulation. Those files are registered with the HTTP control interface (as described below) and can be downloaded to any Web browser. For example, simple 1D data graphs can be viewed by simply clicking on a downloaded file and firing off a graphical program (such as xgraph or gnuplot). Two-dimensional JPEG images can be viewed directly in a Web browser, and continuous time sequences of JPEGs can be displayed using the auto-refresh option of the browsers which support it.

### 5.9.2. Socket connection

A more sophisticated way is to implement a proprietary communication protocol for sending specific geometry data such as isosurfaces or particle trajectories through a raw socket connection to a visualization program. Precomputing such data at the simulation side not only allows for parallel rendering of images but also reduces the amount of data to be transferred to remote visualization clients (such as OpenDX, AVS or Amira).

The approach for streaming arbitrary data of any type is based on the HDF5 I/O library and its VFD layer. Cactus provides a Stream driver which holds the HDF5 data to be streamed out of the Cactus simulation as an in-memory HDF5 file. Upon activation, the entire file is sent through a socket to the connected client. In the client application, the same driver is used to reconstruct the file which then can be accessed in usual ways to read the HDF5 datasets. Since the VFD layer hides all low-level I/O operations from the upper layers of the HDF5 library and from the application that operates on top of it, applications can use their existing HDF5 file-based I/O methods immediately for online remote data access without changing their I/O interfaces. The Stream driver is capable of sending data simultaneously to multiple clients.

Cactus developers are working on a design for a protocol and data server that will handle multiple clients, while also being able to serve requests individually.

## 5.10. REMOTE MONITORING AND STEERING

Remote monitoring and steering can also be done in two ways: thorn HTTPD and stream driver.

### 5.10.1. Thorn HTTPD

The Computational Toolkit contains a thorn HTTPD that can be added to any Cactus simulation in order to provide an inbuilt HTTP server. Any number of collaborators can connect to monitor and steer the simulation online just by pointing their Web browsers to a URL which identifies a Cactus job running on a remote machine. The (provided) Cactus Web interface allows users to query certain information describing the run, such as the current iteration step, a list of available thorns and variables, and a full description of all parameters and their current settings. Upon successful authorization a user can also interactively change parameters that are marked as steerable. Within each simulation cycle these parameters are checked, and the appropriate thorns may react to changes individually. Most of the I/O parameters are steerable. This enables users to selectively switch on or off specific output at runtime, dynamically choosing which variables are output using which I/O methods.

I/O options such as hyperslabbing or downsampling parameters may also be modified in order to adjust online data streaming to remote visualization clients. The Web interface can also be used to pause the simulation when a chosen condition is satisfied, and to advance the simulation by single timesteps.

The Web interface provided by the HTTPD thorn is dynamically extensible, as any thorn can register and update its own HTML pages at runtime. Besides a download page for Cactus output files there also exists a viewport which embeds dynamically-generated JPEG images.

#### **5.10.2. Stream driver**

Another steering interface which builds on top of HDF5 is the Stream driver. For this interface, data streaming is conducted in a bidirectional way: Cactus writes parameters to an HDF5 file which is then streamed to a connected steering client. After some user interaction, this client sends back a modified version of the parameter file that is read and evaluated by Cactus.

As an example, minimum/maximum values could be assigned to numerical parameters to create sliders for more convenient user interaction.

### **5.11. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS**

Cactus seems to be a useful tool for online visualization and simulation. Its main disadvantage is that the parallelization is limited to domain decomposition. However, because of the modular architecture of Cactus, it appears that adding extended functionality would be quite easy for application developers.

---

## 6. NIMROD- A TOOL FOR PERFORMING PARAMETERISED SIMULATIONS

Nimrod[Nim] is a tool for performing parameterized simulations over networks of loosely coupled workstations. After a user specifies the simulation parameters (by developing a declarative "plan" file which describes the parameters, their default values, and the commands necessary for performing the work), Nimrod takes the cross product of these parameters and generates a job for each set. It then manages the distribution of the various jobs to machines, and organizes the aggregation of results. Rather than utilizing a shared filesystem, Nimrod copies files between systems before and after the execution of the program.

### 6.1. LIMITATIONS OF NIMROD AND MIGRATION TO NIMROD/G

Nimrod contains no mechanisms for scheduling computation on the underlying resources. Consequently, users would not have any idea when an experiment might complete. Hence, Nimrod/G, a Grid-enabled version of Nimrod, has been introduced. Using Globus, it is possible for Nimrod users to specify time and cost constraints for computational experiments. Globus provides mechanisms for estimating execution time and wait delays when using queued networked supercomputers. Nimrod/G in turn uses these to schedule the work in a way which meets user-specified deadlines and cost budgets. In this way, multiple Nimrod users can obtain a quality-of-service element from the computational network.

### 6.2. NIMROD/G ARCHITECTURE

As illustrated in Figure 2, a user (or a process acting on behalf of the user) initiates a parametric study at a local site; Nimrod/G then organizes the mapping of individual computations to appropriate remote sites, according to the Nimrod/G scheduling heuristics (basing on market economy models). On the local site, the origin process operates as the master for the whole system. The scheduling and monitoring logic is encapsulated in an origin process, which exists for the entire length of the experiment and is ultimately responsible for the execution of the experiment within the specified time and cost constraints. This structure has been designed to support the scheduling of computations on resources scattered across the globe with their own administrative policies and control structures. The user is either a person or a process responsible for the creation of the experiment. The user interacts with the origin process through a client process. Neither the user nor the client are required to be available all the time. The distinction between the client and the origin is useful because a client may be tied to a particular display or environment. The user can stop the client, move to another environment and start another client, without affecting the origin process and thus the progress of the experiment. In addition, it is possible for multiple clients to monitor the same experiment by connecting to a given origin process.

Each remote site consists of a cluster of computational nodes. A cluster may be a single multiprocessor machine, a cluster of workstations, or even a single processor. The defining characteristic of a cluster is that access to all nodes is provided through a set of resource managers supported by the Globus infrastructure. Each Globus Resource Allocation Manager (GRAM) implements a method for accessing processing resources. Typically, the method is a queue in a batch queuing system, such as Condor or LSF or a process fork on a shared memory machine. Before submitting any jobs to a cluster, the origin process uses the Globus process creation service to start a Nimrod Resource Broker (NRB) on the Cluster. The NRB is not the same as the Resource Manager. It provides capabilities for file staging, creation of jobs from a generic experiment and process control beyond those provided by the GRAM.

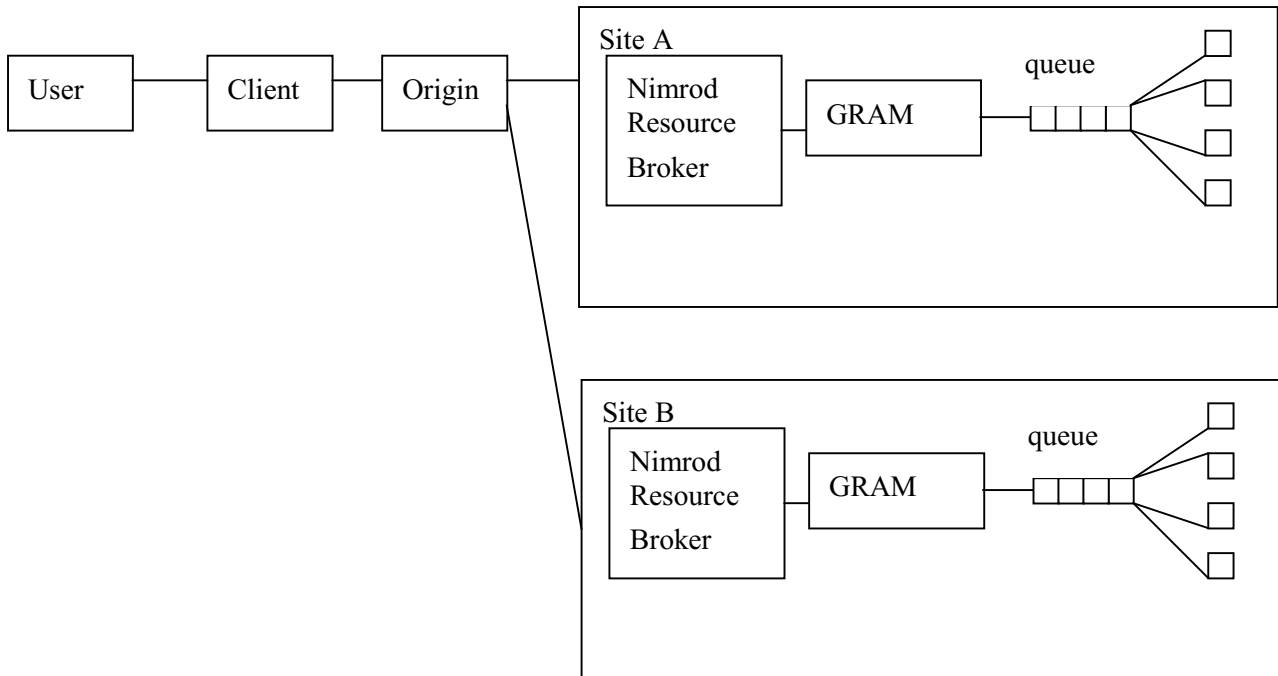


Figure 2: Nimrod/G architecture [Nim].

### 6.3. CONCLUSIONS AND IDEAS USEFUL FOR CROSSGRID APPLICATIONS

The most important features of Nimrod/G are:

- support for parameter study problems (Nimrod/G generates parameter files for each set of parameters from the user “plan” file),
- ability to schedule jobs in a Grid environment,
- support for job deadline control (It is possible for Nimrod/G users to specify time and cost constraints for computational experiments),
- ability to gather results and process the obtained data (i.e. by running data interpretation or visualization software),
- support for multiple clients,
- remote access to running experiments.

Nimrod/G seems to be useful for parameter study problems in a Grid environment. Even though there are some limitations with regard to the flexibility of parameter specifications, it might still be useful for CrossGrid applications. It is worth mentioning that Nimrod/G can gather results and process the data obtained by running data interpretation or visualisation software, which matches the HEP data analysis specification.

## 7. HOW TO USE THESE IDEAS FOR THE CROSSGRID APPLICATION ARCHITECTURE

The aim of this document is to present ideas for making better use of Grid technologies by CrossGrid applications.

### 7.1. REDESIGN EXISTING PARALLEL CODE

The reviews presented here show that various parallel applications are in fact being ported from classical high performance architectures to the distributed Grid architecture, using methods like:

- region of interests management (SF-Express)
- communication and computation overlapping (Overflow-2D, Numerical relativity)
- advanced load balancing (OverFlow-2D)
- data compression (Numerical relativity)
- infrequent synchronization (Numerical relativity)

The authors of Cactus plan to add these features to their tool, so the parallelization of the code in a Grid environment can be done automatically (this is still a long way away, but CrossGrid can nevertheless take advantage of already existing solutions).

### 7.2. USE EXISTING GRID SOLUTIONS FOR CONNECTING SIMULATION AND VISUALISATION

The features of Cactus show that it is a useful tool for porting interactive simulation and visualisation to the Grid environment since it provides mechanisms for sending data between visualisation modules and simulations, and also supports fault recovery mechanism (checkpointing). One solution would be to make the simulation into a Cactus thorn. Visualisations can in fact make use of the visualisation thorns of Cactus (they provide various interfaces to existing visualisation tools like AVS, OpenDX and so on). It would also be possible to compile the HTTPD thorn into the simulation code and control its behaviour through a WWW interface. This would be helpful if application developers wanted to make the simulation or visualisation one of the Web services as described in the OGSA specification.

The modular architecture of Cactus make it possible to use the same code for different purposes.

### 7.3. USE EXISTING GRID SOLUTIONS FOR PARAMETER STUDY

A close analysis of the Nimrod/G capabilities leads to the conclusion that this tool might prove useful. The authors of Nimrod/G focus on schedulers for parameter study problems. Although their ideas will not help in designing the architectures of CrossGrid applications, Nimrod/G could still be used as a tool.

---

**Reference documents**

- [Nim] Abramson, D., Giddy, J., and Kotler, L:  
“High Performance Parametric Modelling with Nimrod/G Killer Application for Global Grid ?”  
<http://www.csse.monash.edu.au/~david/nimrod.html/>
- [Clim] Allcock, B., Foster, I., Nefedova, V., Chervenak, A., Deelman, E., Kesselman, C., Leigh, J., Sim,A., Shoshani, A., Drach, B., Williams, D.:  
“High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies.” SC 2001, November 2001.  
<http://www.globus.org/research/papers.html>
- [Ctga] Allen, G., Bengler, W., Dramlitsch, T., Goodale, T., Hege Gerd, H-C., Lanfermann, Merzky, A., Radke, T., Seidel, E., Shal, J:  
“Cactus Tools for Grid Applications”  
<http://www.cactuscode.org/>
- [Cag] Allen, G., Dramlitsch, T., Foster, I., Karonis, N., Ripeanu, M., Seidel, E., and Toonen, B.:  
“Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus.” In:Proceedings of SC 2001, November 10-16, 2001.  
<http://www.globus.org/research/papers.html>
- [Cfd] Barnard,S., Biswas, R., Saini, S., Van der Wijngaart, R., Yarrow, M., Zechter, L., Foster, I., Larsson, O.:  
“Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus.”  
<http://www.globus.org/research/papers.html/>
- [Ctg] Cactus 4.0 Thorns Guide <http://www.cactuscode.org/>
- [Cug] Cactus 4.0 Users Guide <http://www.cactuscode.org/>
- [Cse] CSE description <http://www.cwi.nl/projects/cse/cse.html>
- [Fost] Foster, I., and Kesselman, C.: “The Grid: Blueprint for a New Computing Infrastructure”
- [Edg] “Grid - Aware Biomedical Applications For DataGrid Testbed Assessment “  
DataGrid-10-D10.2-0109-2-0  
<http://marianne.in2p3.fr/datagrid/wp10/index.html>
- [Cum] Kohl, J.A., and Papadopoulos, P.M.:  
“Cumulvs User's Guide Computational Steering And Interactive Visualization In Distributed Applications”  
<http://www.csm.ornl.gov/cs/cumulvs.html>
- [Tum] Rathmayer, S., and Lenke, M.:  
“A Tool for Online Visualization and Interactive Steering of Parallel HPC Applications “  
[http://wwwbode.cs.tum.edu/Par/appls/res-A/ovid\\_e.html](http://wwwbode.cs.tum.edu/Par/appls/res-A/ovid_e.html)
- [Vis] VISIT description <http://www.fz-juelich.de/zam/visit/>
- [Xray] Wang, Y., De Carlo, F., Mancini, D. C., McNulty, I., Tieman, B., Bresnahan, J., Foster, I., Insley, J., Lane, P., von Laszewski, G., Kesselman, C., Su, M., and Thieboux, M.: “A High-Throughput X-ray Microtomography System at the Advanced Photon Source.” In: Review of Scientific Instruments, April 2001.



<http://www.globus.org/research/papers.html>