

# "Integrating CUMULVS into AVS/Express"

Torsten Wilde, James A. Kohl and Raymond E. Flanery, Jr.  
Oak Ridge National Laboratory<sup>1,2</sup>

Keywords: Scientific Visualization, CUMULVS,  
AVS/Express, Component-Based Design

**Abstract.** This paper discusses the development of a CUMULVS interface for runtime data visualization using the AVS/Express commercial visualization environment. The CUMULVS (Collaborative, User Migration, User Library for Visualization and Steering) system, developed at Oak Ridge National Laboratory, is an essential platform for interacting with high-performance scientific simulation programs on-the-fly. It provides run-time visualization of data while they are being computed, as well as coordinated computational steering, application-directed checkpointing and fault recovery mechanisms, and rudimentary model coupling functions. CUMULVS primarily consists of two distinct but cooperative libraries - an application library and a viewer library. The application library allows instrumentation of scientific simulations to describe distributed data fields, and the viewer library interacts with this application side to dynamically attach and then extract and assemble sequences of data snapshots for use in front-end visualization tools. A development strategy will be presented for integrating and using CUMULVS in AVS/Express, including discussion of the various objects, modules, macros and user interfaces.

## 1. Introduction

Scientific simulation continues to be a field replete with many challenges. Ever-increasing computational power enables researchers to investigate and simulate more and more complex problems on high-performance computers, to obtain results in a fraction of the time or at a higher resolution. The data processed and created by these simulations are huge and require much infrastructure to manipulate and evaluate. Scientific visualization and interactive analysis of complex data during runtime provides a cost-effective means for exploring a wide range of input datasets and physical parameter variations, especially if the simulation runs for days. It can save time and money to discover that a simulation is heading in the wrong direction due to an incorrect parameter value, or because a given model does not behave as expected.

---

<sup>1</sup> Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing research, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

<sup>2</sup> This research was supported in part by an appointment to the ORNL Postmasters Research Participation Program which is sponsored by Oak Ridge National Laboratory and administered jointly by Oak Ridge National Laboratory and by the Oak Ridge Institute for Science and Education under contract numbers DE-AC05-84OR21400 and DE-AC05-76OR00033, respectively

A proper visualization environment allows scientists to view and explore the essential details of the simulated data set(s) [1]. For 2-dimensional (2D) data sets, a 2D visualization environment is sufficient. But for 3-dimensional (3D) problems, a 3D visualization environment is required to provide access to all the detailed information embedded in the data set.

This paper describes work to integrate CUMULVS[2,3] into the AVS/Express[4] viewer environment, which provides a framework for data visualization, including both 2D and 3D capabilities. AVS/Express is unique in the way that it allows changes to the application structure and functionality during runtime. Applications are constructed by “drag & drop” of modules from the component library. The user can add and/or delete components dynamically, to change the application behavior on-the-fly. This work is important in the sense that the integration of CUMULVS into AVS/Express will enable the user to use runtime scientific data sets for visualization instead of file or static data sets.

## **2. Background**

### **2.1. CUMULVS**

CUMULVS (Collaborative, User Migration, User Library for Visualization and Steering) [2,3] provides an essential platform for interacting with running simulation programs. With CUMULVS, a scientist can observe the internal state of a simulation while it is running via online visualization, and then can “close the loop” and redirect the course of the simulation using computational steering. These interactions are realized using multiple independent front-end “viewer” programs that can dynamically attach to, interact with and detach from a running simulation as needed. Each scientist controls his/her own viewer, and can examine the data field(s) of choice from any desired perspective and at any level of detail. A simulation program need not always be connected to a CUMULVS viewer; this proves especially useful for long-running applications that do not require constant monitoring. Similarly, viewer programs can disconnect and re-attach to any of several running simulation programs. To maintain the execution of long-running simulations on distributed computational resources or clusters, CUMULVS also includes an application-directed checkpointing facility and a run-time service for automatic heterogeneous fault recovery.

CUMULVS fundamentally consists of two distinct libraries that communicate with each other (using PVM[5]) to pass information between application tasks and front-end viewers. Together the two libraries manage all aspects of data movement, including the dynamic attachment and detachment of viewers while the simulation executes. The application or “user” library is invoked from the simulation program to handle the application side of the messaging protocols. A complementary “viewer” library supports the viewer programs, via high-level functions for requesting and receiving application data fields and handling steering parameter updates.

The only requirement for interacting with a simulation using CUMULVS is that the application must describe the nature of its data fields of interest, including their decomposition (if any) across simulation tasks executing in parallel. Using calls to

the user library, applications define the name, data type, dimensionality/size, local storage allocation, and logical global decomposition structure of the data fields, so that CUMULVS can automatically extract data as requested by any attached front-end viewers. Given an additional periodic call to the `stv_sendReadyData()` service routine, CUMULVS can transparently provide external access to the changing state of a computation. This library routine processes any incoming viewer messages or requests, and collects and sends outgoing data frames to viewers.

This manual instrumentation of application data can be alleviated by systems like `DynInst` [10] which do automatic run-time introspection of codes, however this type of analysis is not sufficient to fully describe distributed data decompositions; some user intervention is needed to specify the implied parallel semantics and the context of the local data in the overall global array. Unlike systems such as `DICE` [11], where whole copies of each data field are placed in a globally shared file structure using `DDD` [12] and `HDF` [13], in CUMULVS the data movement is demand-driven and the viewers dynamically extract only requested subregions of data fields from each application task. This reduces the application overhead in most cases and provides more flexible multi-viewer collaboration scenarios.

When a CUMULVS viewer attaches to a running application, it does so by issuing a “data field request,” that includes a selection of desired data or “view” fields (constituting a “view field group”), a specific region of the computational domain to be collected (“view region”), and the frequency with which data “frames” are to be sent back to the viewer. CUMULVS handles the details of collecting the data elements of the view region for each view field. The view region boundaries are specified in global array coordinates, and a “cell size” is set for each axis of the data domain, to determine the stride of elements to be collected for that axis, e.g. a cell size of 2 will obtain every other data element. This feature provides more efficient high-level overviews of larger regions by using only a sampling of the data points, while still allowing every data point to be collected in smaller regions where the details are desired. CUMULVS has been integrated with parallel applications written using `PVM` [5], `MPI` [14] and `InDEPS` [15] and can be applied to applications with other arbitrary communication substrates.

### 2.3. AVS/Express Visualization Environment

`AVS/Express`[4] is a commercial environment for visualizing scientific data. It provides the user with a visual programming interface and includes standard modules for the most common visualization functions. Using `AVS/Express` the user can develop a custom viewer by “drag & drop” of modules (objects) and connecting together specific input and output ports of the objects (see Figure 1). *This concept enables users to create a visualization without the need for programming custom code.* Modules in `AVS/Express` represent single object instances in an object oriented programming language. They are the basic components of any `AVS/Express` program. Modules can be grouped into macros in order to create higher-level hierarchical objects. Macros can be grouped with other macros in order to create even higher-order objects. Ultimately, one macro could represent a complete application. Custom module creation can be done for `AVS/Express` via the following 4 steps:

- Define parameters and values using AVS/Express primitive data types (e.g. integer, real, string) or groups of primitive types and other structures.
- Add methods (functions) for the module processing.
- Define the type of execution for the module methods ~ the user code can be compiled directly into the AVS/Express program or can be compiled as its own distinct program.
- Define execution events for module methods and method behavior.

The developer can specify which parameters are connected to which methods. There are four possible options here:

- notify: The method is called if the parameter value changes.
- read: The method reads the parameter.
- write: The method writes the parameter.
- required (req): The method can only be called if the parameter has a valid value, as checked automatically before event processing.

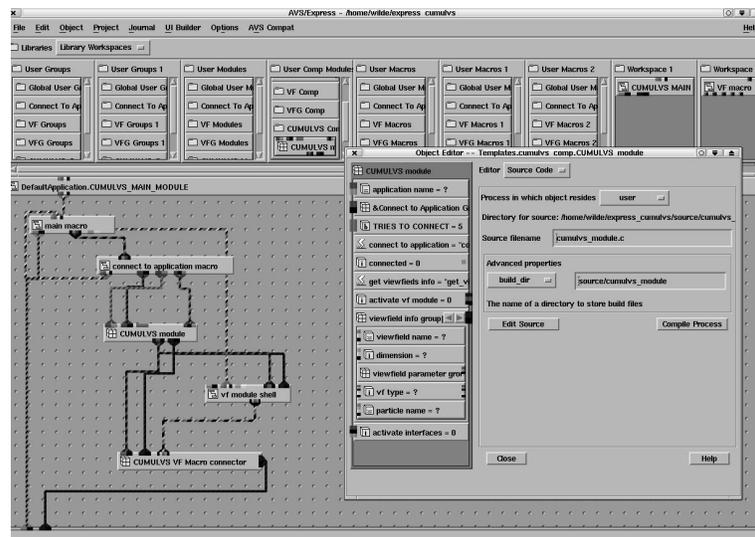
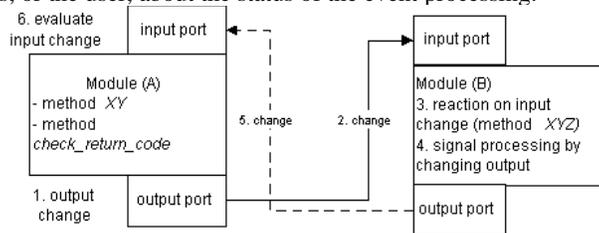


Fig. 1. AVS/Express application creation interface

AVS/Express incorporates a data driven or “Event based” execution paradigm. It responds to events to execute different sets of instructions depending on which event occurs rather than following a pre-defined sequence of instructions. This means a module method can only be executed if a specific parameter has changed. Usually this principle is used where the program states are driven by the graphical user interface (GUI). Figure 2 shows the handling of function return codes in this paradigm. A “hand shake” approach is used, where the caller receives feedback regarding the processing state of the event, such as error-success information. An event change in *Module A* executes method *XY*. Because this event requires some processing in

*Module B*, the method changes the output port of *A*. This change triggers the input port of *B* (connected to the output port of *A*). *Module B* now reacts to this event by executing method *XYZ*. After finishing the execution *XYZ* writes the status information and/or return values to output port *B*. This changes the value of the corresponding input port of *A*, triggering the execution of the method *check\_return\_code()* which evaluates the return code and/or values and can inform other modules, or the user, about the status of the event processing.



**Fig. 2.** Module Execution Paradigm Example

Modules can use this principle to verify if another module has been connected. This is important for verifying the program state at all times, especially when dealing with changing connections or new modules at runtime. AVS/Express allows multiple modules to be connected to the same module port. The GUI also can be dynamically extended as the application changes.

### 3. CUMULVS Interface Design for Integration into AVS/Express

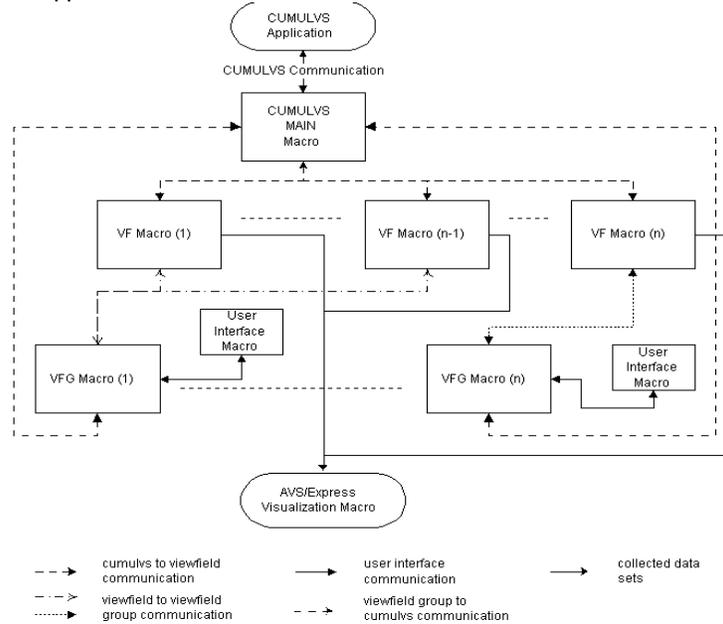
The goal of this work was to integrate CUMULVS into AVS/Express in order to enable runtime data visualization and to create an AVS/Express viewer for the CUMULVS library. CUMULVS already supports several graphical viewers based on AVS5[6], Tcl/Tk[7], VTK[8] and the CAVE[9] environment. The AVS/Express work is especially interesting because of its component-based functionality. It is a complete viewer environment with components for everything needed to view scientific data, e.g. reading, filtering, transforming and visualizing the data. AVS/Express provides a dynamic application structure, e.g. viewers can be customized to an application on-the-fly by adding or deleting components using the visual programming interface. By adding new modules to the AVS/Express module library the user can improve and extend the AVS/Express capabilities and construct custom viewers. The following subsections describe the module structure and GUI for the CUMULVS AVS/Express viewer.

#### 3.1. CUMULVS Module Structure

The CUMULVS functionality is divided into global modules for the AVS/Express viewer by analyzing program functionality paired with the required GUI blocks. Figure 3 shows this global module structure including 4 primary macros.

The *CUMULVS Main Macro* handles all communication with the running application and provides the GUI for specifying the application name and other global parameters. It also provides information about the connected application, like available data fields for viewing, their bounds and data type, and whether the data is particle-based or a mesh decomposition. Because of its central role, this module must communicate with all other modules:

- Sends View Field (VF) information to VF Modules
- Get View Field Group (VFG) information from VFG Module for data collection
- Sends application connection status to VFG Module



**Fig. 3.** main object structure

The *VF Macro* gets the VF information from the *CUMULVS Main Macro* and provides the GUI for selecting one VF from the possible VFs for viewing. Every VF requires one *VF Macro*, which stores all important information about the selected VF and transfers it to the *VFG Macro*. One *VF Macro* can only be connected to one *VFG Macro*, e.g. if the same VF is required for a second *VFG*, an additional *VF* module has to be instantiated for this VF.

The *VFG Macro* combines connected *VF Macros* into one *VFG*. The *VFG Macro* calculates the global bounds and dimension from the connected VF values. For example, the global lower boundary could be the highest VF lower boundary found in the connected VFs and the global upper boundary could be the lowest VF upper boundary found. The *VFG Macro* provides base values for the *User Interface (UI) Macro*, which sets parameters like boundaries and cell size for each dimension, visualization frequency, etc. The *UI Macro* checks all user input for errors before

sending it to the VFG Macro. When input is forwarded to the Main Module, the data collection is started or appropriate changes to the data collection are made.

In addition to data flow through the system, control parameters are also transferred between macros. Initially, only the Main Macro GUI is activated, but after successful connection to the application the VF Macro is then activated. Activation happens by setting a port connection to “true” (an integer value of 1).

The VFG Macro is activated if at least one connected VF Macro has a valid VF selection. After the user connects the VFG to the application, the VFG information is sent to the Main Module together with the “connect” flag. Likewise if the user chooses to disconnect the VFG, the “disconnect” flag is sent to the Main Module. The connection status is transferred back from the Main Module to the VFG Macro and from there on to the VF Macro, and influences the GUI status of these modules.

### 3.2. Graphical User Interface (GUI) Structure

Because the user typically controls the CUMULVS-AVS/Express modules via the visual interface it is very important to design for flexible use and efficient overview of the vital information. The GUI consists of three main windows corresponding to the three main macros and their functionality. The first window allows input of the application name and is created from the Main Macro. A second *View Field Info* (VFI) window is also created by the Main Macro and provides a port for the VF Macro. The VFG Macro creates a third *User Interface* (UI) window. The structure of these latter two windows is shown in Figure 4.

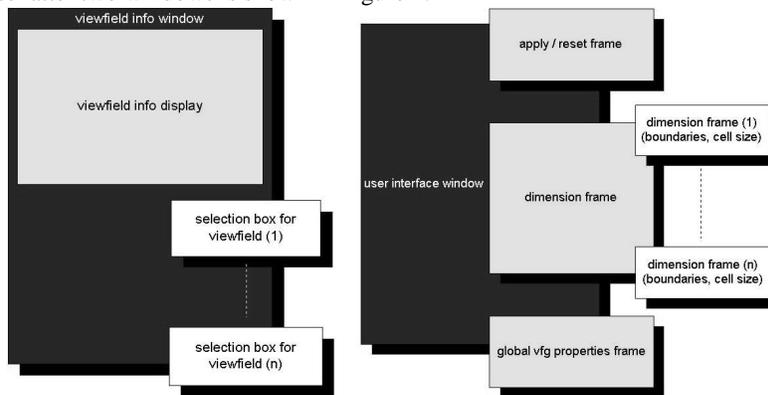


Fig. 4. View Field Info and User Interface Window Structures

The VFI window displays important information about the available VFs, including name and boundaries for each dimension. Each instantiated and connected VF Macro adds a “selection box” for its VF to the VFI window. If the connection from the VF Macro to the Main Macro or the VF Macro itself is deleted, the corresponding “selection box” is also deleted. The VFI window resizes automatically.

The UI window consists of three frames, which are positioned depending on the order of their connection to the window. This portion of the GUI enables the user to

change the parameters for the collected VFG data set. The “apply/reset frame” allows the user to apply the changes to the VFG or reset to previous values. The “dimension frame” sets the boundaries and cell size for each dimension. This frame is constructed modularly with one “dimension property frame” per dimension of the VFG, e.g. for a 3 dimensional VFG, 3 such frames are connected. The overall dimension frame is resized automatically. The user can adapt this interface to different problems (applications) on the fly. All global VFG parameters like visualization frequency can be changed using the “global VFG properties frame”. Each frame has a well-defined port connection to its parent frame or window. It is therefore possible for users to create their own customized GUIs using the provided frames, or by creating new ones implemented with the given input/output port connection specification.

#### 4. Results

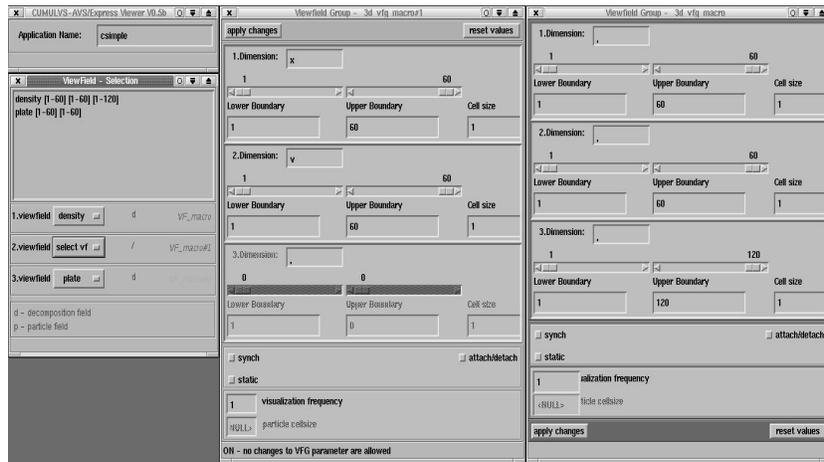
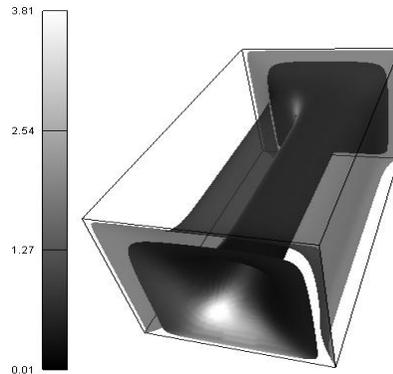


Fig. 5. Screenshot of the CUMULVS-AVS/Express Interface

Figure 5 shows a sample snapshot of the GUI in action. The application name window is in the upper left corner. The next window below that is the “View Field Selection” window. Two data fields from the application are available for viewing: “density” has 3 dimensions with boundaries [1-60][1-60][1-120], and “plate” is 2 dimensional with boundaries [1-60][1-60]. There are three view fields attached. The first two (“VF\_macro”, “VF\_macro#1”, see also Figure 6) are connected to one view field group (right interface window, “3d\_vfg\_macro”). The third view field builds its own VFG (middle interface window, “3d\_vfg\_macro#1”). The VFG windows are color-coded, with the VF\_Macro name in the “View Field Selection” window colored like the VFG to which it belongs. In addition, a special letter code describes the type of the selected view field (“d” for mesh decomposition field and “p” for particle field). The two VFG windows are composed of the three frames described in 3.2. Values are initialized using default VFG parameters. Any user input is validated before changes are submitted to the VFG, after the “apply changes” button is pressed.

Because VF “plate” is two-dimensional in this example, the input for the third dimension is deactivated automatically in the corresponding UI window (left).

Figure 6 shows a 3D visualization of an example application rendering from a simple LaPlace simulation. Surface rendering was used to visualize the data set. The inner surface was solid rendered and the outer surface in 65% transparent.



**Fig. 6.** Visualization of LaPlace Simulation

## 5. AVS/Express Concerns

During this development several problems with AVS/Express for Linux were encountered. As a result, the CUMULVS viewer plug-in is currently only available for AVS/Express on SGI Irix systems. A crucial bug is related to the motif environment, which causes random crashes of AVS/Express if menus or buttons are accessed. AVS support verifies the problem, and there is an updated version of Motif1.2 available for Linux glibc2.1. Unfortunately this update does not seem to work for other glibc versions like glibc2.2. Also some stability problems were encountered with the new AVS/Express5.1 under Linux. The most stable overall environment seems to be Mandrake7.2, with the Motif update and AVS/Express5.0.

The only know problem with AVS/Express5.1 under SGI Irix is that the user interface input field “UIfield” instantiates with a fixed field width independent of the value set in the object (Figure 5, bottom right window, for visualization frequency).

As a developers note, it is important to point out that there is no clear order of module instantiation and method execution if a complex saved program or macro is instantiated. The order could in fact be opposite to the drag & drop program creation order, and the order of connections is also not guaranteed. It is therefore possible to introduce problems that are only visible during instantiation of the whole program, and not during component testing. Feedback loops can occur involving different high-level macros. Also, setting the “required” flag (see Sect. 2.3) for an array doesn’t ensure that all array values will be valid. Missing values inside the array are not detected by AVS/Express. This is especially problematic for pointer arrays where the access of an invalid pointer leads to program termination or critical failure.

## 6. Summary/Future Work

The integration of the CUMULVS functionality into AVS/Express enables CUMULVS users to take advantage of the powerful component-based AVS/Express viewer environment, and similarly AVS/Express users can collect and visualize data from running parallel/distributed scientific applications using CUMULVS. The event-based execution paradigm and the highly scalable module approach make AVS/Express very flexible, but at a potentially high cost in complexity for internal module communication. The plug-in was tested using simple example applications. The next step will be to use it in real world applications. Future plans include solving the problems with AVS under Linux, integrating a steering interface into the CUMULVS plug-in and improving or rearranging the user interface based on user feedback.

## References

- [1] K.J. Weiler, "*Topological Structures for Geometric Modeling*", Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY, May 1986
- [2] G.A. Geist, J.A. Kohl, P.M. Papadopoulos, "*CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications*", INTL Journal of High Performance Computing Applications, Volume II, Number 3, August 1997, pp. 224-236.
- [3] J.A. Kohl, P.M. Papadopoulos, "*CUMULVS user guide, computational steering and interactive visualization in distributed applications*", Oak Ridge National Laboratory, USA, Computer Science and Mathematics Division, TM-13299, 02/1999.
- [4] "*AVS/Express Developer's Reference*", Advanced Visual System Inc., Release 3.0, June 1996.
- [5] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, "*PVM: Parallel Virtual Machine*", *A User's Guide and Tutorial for Networked Parallel Computing* The MIT Press, 1994.
- [6] "*AVS User's Guide*", Advanced Visual Systems, Inc., Waltham, MA, 1992.
- [7] J.K. Ousterhout, "*Tcl and the Tk Toolkit*", Addison-Wesley, Reading, MA, 1994.
- [8] Will Schroeder, Ken Martin, Bill Lorensen, "*The Visualization Toolkit an object-oriented approach to 3D graphics*", 2<sup>nd</sup> Edition, Prentice Hall PTR, 1998
- [9] CAVERNUS user group, CAVE Research Network Users Society, <http://www.ncsa.uiuc.edu/VR/cavernus>
- [10] DYNINST - An Application Program Interface (API) for Runtime Code Generation <http://www.dyninst.org>
- [11] J.A. Clarke, J.J. Hare, C.E. Schmitt, "Distributed Interactive Computing Environment (DICE)", Army Research Laboratory, Major Shared Resource Center, <http://frontier.arl.mil/clarke/dice.html>
- [12] J.A. Clarke, J.J. Hare, C.E. Schmitt, "Dice Data Directory (DDD)", Army Research Laboratory, Major Shared Resource Center, see <http://frontier.arl.mil/clarke/Dd.html>
- [13] "Hierarchical Data Format (HDF)", National Center for Supercomputing Applications
- [14] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, "MPI: The Complete Reference", MIT Press, Cambridge, MA, 1996
- [15] R. Armstrong, P. Wyckoff, C. Yam, M. Bui-Pham, N. Brown, "Frame-Based Components for Generalized Particle Methods", High Performance Distributed Computing (HPDC '97), Portland, OR, August 1997, <http://glass-slipper.ca.sandia.gov/~rob/poet/>