CUMULVS: Collaborative Infrastructure for Developing Distributed Simulations *

James Arthur Kohl Philip M. Papadopoulos G. A. Geist, II †

Abstract

The CUMULVS software environment provides remote collaboration among scientists by allowing them to dynamically attach to, view, and "steer" a running simulation. Users can interactively examine intermediate results on demand, saving effort for long-running applications gone awry. In addition, CUMULVS provides fault tolerance to distributed applications via user-directed checkpointing, heterogeneous task migration and automatic restart. This paper describes CUMULVS and how this tool benefits scientists developing large distributed applications.

1 Introduction

Computer simulation is becoming increasingly important as an alternative to the often expensive task of constructing physical prototypes and experiments for scientific research and development. More and more, scientists are developing software simulations of their work to be run on high-performance parallel or distributed computer systems. Such software experiments provide a cost-effective means for exploring a wide range of variations in input datasets and physical parameters. The on-line nature of these experiments also provides a framework within which scientists can collaborate with other remotely located researchers, a task not possible when physically constructing a prototype or experiment.

There are many issues to overcome when developing a high-performance computer simulation, especially when many scientists must interact with the simulation over a wide-area network. The scientists must be able to observe the progress of the simulation and share and coordinate control over it, and the user environment must be capable of withstanding or recovering from system failures. The efficient handling of these issues requires a special expertise in computer science and a level of effort higher than the typical application scientist is willing to expend.

The CUMULVS system [1, 2, 3, 4] is an infrastructure for developing high-performance computer simulations in a collaborative environment. Using CUMULVS, each scientist in a group can dynamically attach to a running simulation, extract custom views of the intermediate data results on-the-fly, and coordinate control over the progress of the computation by "steering" physical or algorithmic parameters. Each collaborator interacts with the simulation via a front-end program called a *viewer*[1]. Any number of viewers can simultaneously attach and detach from the simulation, and the connections that CUMULVS provides between the simulation and the viewers are tolerant of network faults

*Research supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corp.

†Oak Ridge National Laboratory, Oak Ridge, TN

[1] "Viewer" is a generic term to describe a program for visualizing or steering an application.

and host failures. CUMULVS also provides a platform for integrating fault-tolerance into the simulation program itself, for saving user-directed checkpoints, migrating the simulation program across heterogeneous computer architectures, automatically restarting failed tasks, and reconfiguring the simulation while it is running.

The remainder of this paper briefly describes the features of the CUMULVS system, and discusses specific scenarios in which it can be of assistance to collaborating scientists.

2 CUMULVS Overview

CUMULVS is a freely-available library middle-ware system that bridges the gap between existing application codes and commercial visualization / user interface packages. The CUMULVS libraries can be used with simulation programs written in C, C++ or Fortran. The CUMULVS communication protocols can be utilized on top of any complete message-passing system, and the viewing functions can be linked to any front-end visualization system. The current implementation uses PVM [5] as a message-passing substrate, and several visualization systems are supported including AVS [6] and Tcl/Tk [7]. Porting CUMULVS to a new system requires only creation of a single declaration file, to define the proper calling sequences for CUMULVS. The CUMULVS distribution includes several complete viewer programs: a standard text-only viewer, a standard Tcl/Tk slicer, a standard AVS-compatible viewer, and a sample custom Tcl/Tk viewer for a particle-based simulation. These viewers are sufficient for typical user needs; however, if a special viewer is desired, CUMULVS provides a library interface for building custom viewers.

To use CUMULVS with an existing simulation code, the computational data fields and algorithmic or physical parameters of the simulation must be described. These special declarations consist of the data type, the size and cardinality of any arrays, and any distributed data decompositions. Several data decompositions are supported, including generic block-cyclic decompositions as described by High-Performance Fortran (HPF) [8, 9][2], and particle-type decompositions with user-defined particle accessor functions. CUMULVS needs declarations only for the data that is to be viewed and the parameters that are to be steered, not the entire symbol table. At the point in the simulation's calculation where the data field values are deemed "valid," a single call is made to the "send-to-front-end" library routine, to temporarily pass control to CUMULVS. Here, CUMULVS handles any pending viewer requests and processes any steering parameter updates. If no viewers are attached, the "sendToFE" routine carries only the overhead of a single message probe, so that negligible intrusion to the simulation is experienced.

Some of the details of CUMULVS's data field viewing and computational steering features are described in the following two subsections, respectively. A third subsection describes the fault tolerance features that CUMULVS provides for user simulation programs.

2.1 Data Field Viewing

When a CUMULVS viewer attaches to a running simulation, it does so in terms of a *data field request*. This request includes some set of data fields, a specific region of the computational domain to be collected, and the frequency of data *frames* which are to be sent to the viewer. CUMULVS handles all of the collection of the sub-region, or *visualization region*, of data from the simulation. For a distributed or parallel simulation, each task in the simulation need only identify its logical processor position in the data field decomposition,

[2] In fact, the built-in decomposition accessor functions in HPF provide for potentially automatic instrumentation of a simulation for use with CUMULVS.

and then CUMULVS can determine which data elements are present in each task and extract the data automatically.

In addition to the boundaries of the sub-region, a full visualization region specification also includes a "cell size" for each axis of the computational domain. The cell size determines 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 for more efficient high-level overviews of larger data 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. To reduce network traffic CUMULVS downsizes the datasets within the parallel simulation tasks themselves, instead of at the viewer. CUMULVS-capable viewers are sent only the data that they have requested. Another performance gain is achieved by sending the data asynchronously, as is discussed below.

Once CUMULVS has collected the data for a given visualization region at each local task, the data is sent to the viewer task where it is automatically assembled into a coherent frame for animation. CUMULVS ensures that each viewer has a time-coherent view of parallel data. An internal sequence number is maintained to determine where each simulation task is in the computation, and any attached viewers loosely synchronize with the simulation for each data frame they receive. The loose synchronization means that a viewer can determine on which iteration tasks are computing to within some desired number of iterations (as controlled by the frame frequency) without using an explicit barrier. The frequency of data frames can be dynamically set from the viewer to control the overhead and the loose synchronization effects.

2.2 Computational Steering

Aside from collecting data frames from running applications, CUMULVS viewers can also remotely "steer" a simulation's computational parameters on-the-fly. Often this is a useful capability when the scientist desires to explore "what-if" experiments with various parameters in a simulation, or perhaps viewing the intermediate results of a computation can reveal a problem or a new opportunity to manipulate the simulation. Such interactive control can save countless hours of wasted computation time waiting for final simulation results that might have been corrupted in the first few iterations.

CUMULVS supports simultaneous coordinated steering of simulations by multiple collaborators. A token scheme prevents conflicting adjustments to the same steering parameter by different users. Consistency protocols are used to verify that all tasks in a distributed simulation apply the steering changes in unison. So scientists, even if geographically separated, can work together to direct the progress of a computation, and can do so without concern for the consistency of steering parameters among potentially distributed tasks.

The simulation program can determine whether parameters have been "steered" by CUMULVS via the return value from the main "sendToFE" library routine. This value indicates the number of steering parameters that have been updated for the next computation phase. The simulation program can, if necessary, poll individual parameters with an additional library routine to see specifically if CUMULVS has changed a given parameter's value. Steering parameters in CUMULVS can be either scalars or vectors, with each element in a parameter vector being of a potentially different data type. Steering parameters can also be "indexed," to allow multiple users to steer individual elements of parameter arrays simultaneously.

CUMULVS does not require explicit synchronization with the simulation program to manipulate steering parameters. Instead, the same loose synchronization used for visualization data frames is applied for steering parameter updates. As a result, the precise iteration for each concurrent simulation task is not known, so changes to steering parameters can only become uniformly effective at some future predetermined iteration.

2.3 Fault Tolerance

Much of the same infrastructure that is used for visualization and steering can be used for *user-directed* checkpointing. The descriptions of the simulation data and decompositions can be applied for efficiently extracting checkpoint data. By taking advantage of the user's knowledge of a simulation program, the minimal necessary program state can be identified and saved, along with detailed structural information, to produce a very flexible checkpointing system. Not only can this checkpoint data be used to automatically restore the simulation in the event of a failure, but these checkpoints can be used to interactively migrate simulation tasks, even across heterogeneous machine architectures, for improved load balance / performance. Such heterogeneous migration is not possible with environments that utilize system-directed checkpointing, where full core images are saved. Also, because the user precisely defines the minimal program state, the amount of checkpoint data can be significantly smaller.

The well-defined nature of CUMULVS checkpoints also leads to some innovative capabilities in terms of dynamically reconfiguring an application. Because CUMULVS understands the various data decompositions in a simulation program, it can re-arrange the data stored in a checkpoint and actually restart the simulation using a different processor configuration. So, a checkpoint saved from an execution on a 256-node Intel Paragon could, for example, be restarted instead on a 32-node IBM SP system, to continue from where it had left off. Likewise, if the computational resources available to the simulation program were to change, CUMULVS could be told to reconfigure the simulation program on-the-fly to use more or fewer processing nodes. This type of automated flexibility could greatly improve the productivity of simulation-based research. Of course, for this to be possible the simulation program must be able to handle certain changes in data decompositions. This may, however, be handled already in terms of normal checkpoint restarting, where a restarted simulation task must likely adapt itself to oversee an externally selected set of data elements within a decomposition.

3 Usage Scenarios

This section outlines some sample usage scenarios that a group of collaborating scientists might encounter over the course of a simulation experiment. These scenarios encompass the development of an initial sequential simulation prototype, its evolution to a distributed simulation, and its subsequent analysis by the group using CUMULVS.

3.1 Initial Sequential Prototype

Consider a team of collaborating scientists who are designing an airplane wing. It is likely not cost effective to design physical prototypes initially, so these scientists, spread across the country at different sites, decide to start with a computer simulation of the air flow over the wing. The simulation will explore the stability and performance of the wing given different configurations and air flow inputs, with the goal of designing the best wing given various constraints.

```
ndim = 3;
axis_type = { stvCollapse, stvCollapse, stvCollapse };
axis_info = { stvDefault, stvDefault, stvDefault };
global_lower_bounds = { 1, 1, 1 };
global_upper_bounds = { 250, 250, 500 };
processor_rank = 1;
processor_shape = { 1 };

decomposition = stv_decompDefine( ndim, axis_type, axis_info, axis_info,
        global_lower_bounds, global_upper_bounds, processor_rank, processor_shape );

array_declare = { 250, 250, 500 };
processor_address = { 0 };

field_id[0] = stv_fieldDefine( pressure, "pressure", decomposition,
        stvNoFieldOffsets, array_declare, stvFloat, processor_address, stvVisOnly );

field_id[1] = stv_fieldDefine( pressure2, "temp pressure", decomposition,
        stvNoFieldOffsets, array_declare, stvFloat, processor_address, stvVisOnly );
```

FIG. 1. *Serial CUMULVS Instrumentation*

The first step is to write a computational fluid dynamics (CFD) simulation program that computes the pressure of the air flowing around the wing. This first simulation is a simple sequential one that will serve primarily to verify the model – it will check for the expected results in a few basic cases. At this stage, CUMULVS is applied to interactively view the air pressure data as the CFD algorithm slowly converges to a solution. The sequential simulation is CUMULVS-instrumented to define the pressure field and a few special parameters, such as the Mach number and angle of attack. The decompositions at this point are trivial, as the data is not actually decomposed, but all resides in a single computational task. The instrumentation does not take long (see example in Figure 1). With CUMULVS in place, the group runs a few simulations. Each scientist connects a viewer from their site to the lone running simulation program and observes the progress of the computation. Some scientists observe the overall computation domain at a low granularity, while others "zoom" in to look at the details in a few specific sub-regions.

Although the entire simulation would take days to converge, several simple program bugs are seen immediately by viewing the intermediate pressure values. After a few partial runs, having cleaned up some program bugs, the basic simulation appears to be working and a few serious test runs are made. Each of these runs takes a very long time to execute, and not much progress is made toward the ultimate goal. Several scientists use their CUMULVS viewers to coordinate steering of parameters to modify the physical model characteristics, but the response time is poor and each change takes many iterations to be seen in the viewed data.

3.2 Distributed Simulation

The scientists decide to parallelize the simulation code to improve its performance. The pressure field around the wing is separated into smaller grid regions which can be computed

axis_type = { **stvBlock**, **stvBlock**, *stvCollapse* };
axis_info = { *stvDefault*, *stvDefault*, *stvDefault* };
. . .
processor_rank = **2**;
processor_shape = { **32**, **32** };
. . .
array_declare = { **8**, **8**, 500 };
processor_address = { **i, j: i=0,31; j=0,31** };

FIG. 2. *CUMULVS Instrumentation Changes for Distributed Data Decomposition*

field_id[2] = stv_fieldDefine(resid, "residuals", decomposition,
stvNoFieldOffsets, array_declare, *stvFloat*, processor_address, *stvVisOnly*);

FIG. 3. *Debugging Residual Field Instrumentation*

in parallel. At the end of each iteration, the boundaries among these grid regions are exchanged to keep all the pieces "glued" together numerically. The CUMULVS instrumentation is extended to describe this new data arrangement – a simple block decomposition – and now the group is ready to try another run. The corresponding changes to the CUMULVS declarations are shown in Figure 2 – the decomposition for the two pressure fields now uses a two-dimensional processor configuration with standard "block-block" data distribution on the first two axes.

Using CUMULVS viewers, the group connects to the now parallel simulation, which runs on a large 1024-node multiprocessor. Again the scientists independently explore the ongoing simulation, looking for indications that the computation is proceeding properly. But in fact it is not – the new parallel version does not seem to converge at every point in space. So one of the scientists goes into the program source code and adds a CUMULVS declaration for the computation "residuals" data field, to see precisely where the convergence problem is coming from (see Figure 3). This data field, already present in the computation, represents the error in the solution, and is used to direct the simulation towards convergence.

A new simulation run is started from scratch, with the residual field now available for CUMULVS viewing. It quickly becomes apparent that one of the columns of data elements in the residual field is not changing, as this column sticks out like a huge spike in the viewer display while the rest of the field settles. The array indices were not being traversed correctly, and the last elements along the two main axes were not being computed. The bug is quickly fixed, and now the real development work can begin.

3.3 Collaborative Analysis

The scientists start setting up many different simulation runs to explore the solution space for their wing design. Each of these runs still takes many hours to complete, even in parallel, but some answers are beginning to become clear. Occasionally, one of the big multiprocessor computers fails, but having added some CUMULVS checkpointing to the simulation program, things are always restarted to continue forward.

At first, all of the scientists work together on a few specific simulation experiments, all watching them progress from their independent viewers. Then some of the scientists get some ideas on their own, and begin exploring some alternate simulations. One scientist

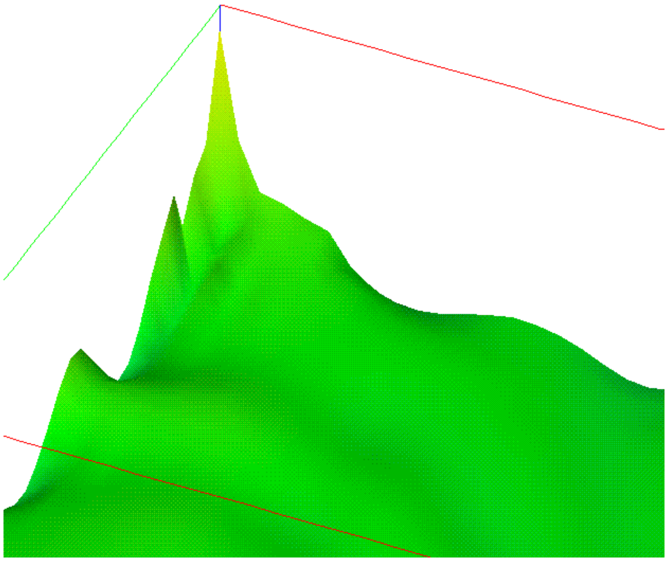

FIG. 4. *CFD Residuals View Using CUMULVS*

takes a checkpoint from an almost completed run on the "big" machine, and reconfigures it, using CUMULVS, to run on a local workstation cluster. The scientist steers a few parameters to push the simulation in a slightly different direction, and gets some good results. Excited at the new information, the scientist contacts a colleague from across the country, who proceeds to hook up a viewer to the new simulation. The new configuration looks very good, and soon the two scientists migrate the simulation again, reconfiguring it for the "big" multiprocessor machine again. Everyone checks it out, and "bingo" – a major breakthrough!

3.4 Real-Life Example

A scenario much like the one above, but on a smaller scale, was experienced by a researcher here at ORNL in preparation for the Supercomputing '96 High-Performance Computing Challenge demonstration, in which ORNL received a Silver Medal for Innovation with CUMULVS. The residuals data field example above was extracted directly from that development effort, as shown in Figure 4.

CUMULVS proved to be a valuable tool over the course of that project, and was utilized repeatedly for debugging the simulation software and exploring the resulting software model. An image of the final simulation in action is shown in Figure 5.

4 Conclusions

The CUMULVS system is an infrastructure for developing high-performance computer simulations in a collaborative environment. Using CUMULVS, scientists can dynamically attach to, extract custom views from, and coordinate control over a running simulation from around the world. CUMULVS also provides a means for integrating flexible fault-tolerance into simulation programs, and allows saving of user-directed checkpoints, migration of the simulation program across heterogeneous systems, and automatic restarting and reconfiguring of an ongoing simulation. There are many scenarios in which CUMULVS can be applied to expedite collaborating scientists' development of simulation experiments.

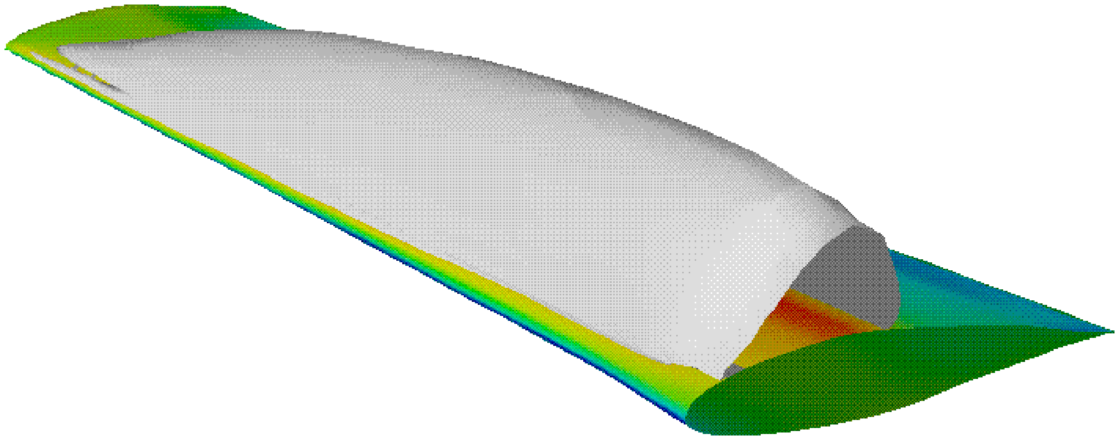

FIG. 5. *View of Running Air Flow Simulation Using CUMULVS*

CUMULVS can be used to help debug a simulation, and to explore "what-if" hypotheses interactively among collaborators. CUMULVS has already been applied to real scientific projects with great success.

5 Acknowledgments

The authors wish to thank Dave Semeraro at Oak Ridge National Laboratory for his work on the air flow simulation, and his help as guinea pig in really trying out CUMULVS.

References

[1] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos, *CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications*, International Journal of Supercomputing Applications (to appear), also available via http://www.epm.ornl.gov/cs/cumulvs96.ps.

[2] P. M. Papadopoulos and J. A. Kohl, *CUMULVS: an Infrastructure for Steering, Visualization and Checkpointing for Parallel Applications*, 1996 PVM User's Group Meeting, Santa Fe, NM.

[3] J. A. Kohl and P. M. Papadopoulos, *The Design of CUMULVS: Philosophy and Implementation*, 1996 PVM User's Group Meeting, Santa Fe, NM.

[4] J. A. Kohl, P. M. Papadopoulos, *A Library for Visualization and Steering of Distributed Simulations using PVM and AVS*, Proceedings of the High Performance Computing Symposium, Montreal, Canada, 1995, pp. 243–254.

[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] Advanced Visual Systems, Inc. *AVS User's Guide*, Waltham, MA, 1992.

[7] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994.

[8] C. Koebel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, MA, 1994.

[9] *High Performance Fortran Language Specification, Version 1.1*, Rice University, Houston, TX, November, 1994.