

A Chromium Based Viewer for CUMULVS

Dan Bennett

Corresponding Author

Department of Mathematics
and Computer Science
Edinboro University of PA
Edinboro, Pennsylvania 16444
Phone: (814) 732-1177
Fax: (814) 732-1170
Email: dbennett@edinboro.edu

Paul A. Farrell

Department of Computer Science
Kent State University
Kent, Ohio 44242
Phone: (330) 672-9060
Fax: (330) 672-7824
Email: farrell@cs.kent.edu

Cara Stein

Department of Mathematics
and Computer Science
Edinboro University of PA
Edinboro, Pennsylvania 16444
Phone: (814) 732-1184
Fax: (814) 732-1170
Email: cstein@edinboro.edu

Abstract—As cluster computing becomes increasingly more accessible the need for tools to simplify the development and use of parallel simulations grows proportionally. One class of tools, steering and visualization middle-ware, promises to assist in code development as well as increase the value of the final simulation produced.

We are interested in employing a steering and visualization middle-ware package known as CUMULVS. By creating a distributed visualization client, or viewer, we have been able to produce images of the running application at a much higher data rate than a similarly configured single processor viewer. In this paper, we describe the design of such a viewer and present preliminary performance data.

I. INTRODUCTION

Beowulf clusters have begun to deliver the promise of high performance computing (HPC) to a much wider range of scientific and industrial computational problems. Applications built upon libraries such as the Message Passing Interface (MPI) can be developed on small clusters and later executed on the much larger machines at the nation's supercomputer centers. Beowulf clusters have become common in even the smallest science departments.

Ahalt [1] states that HPC software is hard to use, and that the lack of robust software tools in this area is the biggest barrier to a more widespread adoption of HPC. Dongarra [2] points out that such tools are usually the first item abandoned by vendors, and that open source projects need to fill this gap. The

distributed nature of such environments presents a challenge for the development of tools from parallel interactive debuggers to data visualization environments. Furthermore, to be truly useful, such a tool must have a minimal impact on the performance of the simulation with which it is interacting.

We have developed a distributed viewer for molecular dynamics simulations which interacts with steering and visualization middle-ware to extract data from a running simulation. This viewer is Chromium aware and employs this library to produce visualizations on devices ranging from single monitors to wall type displays.

II. PACKAGES INVOLVED

A. Scientific Steering and Visualization and CUMULVS

Scientific visualization and steering middle-ware tools for distributed environments can be employed to extract data from a running application for visualization by an external program called a viewer, or to receive data from such a viewer to provide interactive manipulation, or steering, of the application. Sunderam[3], provides a set of design goals, and CUMULVS [4] is an example of such a package. Originally designed to provide interaction with programs built to run in a batch queue environment, this software contains a library of routines which a programmer can use to instrument, or add data extraction routines, to an existing computation. Once instrumented, multiple users can employ one of the supplied viewers to attach to a running computation for interactive steering and visualization. The

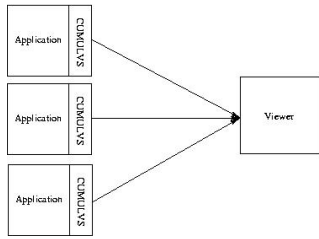


Fig. 1. A Typical CUMULVS Configuration.

middle-ware software is responsible for extracting data only when it is requested, and transporting, assembling, and delivering data from the distributed nodes in the computation to the viewer. A typical CUMULVS application is pictured in Figure 1, where data from a distributed application is extracted via CUMULVS and delivered to a viewer.

CUMULVS was designed to have a minimal impact on the distributed program, both in terms of the amount of code that is inserted during the instrumentation process and the performance degradation when employing the software. Instrumentation, which depends on the nature and number of data fields to be extracted, typically requires the insertion of several hundred lines of code. Performance is effected only when viewers are attached, and depends on the frequency that data is extracted, as well as the number of attached viewers. We have documented this behavior in [5].

A number of viewers are available for use with CUMULVS. These include a two dimensional viewer named TK-slicer, a cgi based viewer called GIF-o-matic, a viewer to use with AVS and AVS Express [6]. These viewers demonstrate the versatility of the CUMULVS library, and provide examples for creating additional viewers. With display walls becoming more available, there appears to be the need for a distributed viewer which can take advantage of the distributed nature of such a display wall.

One final feature of CUMULVS relevant to this work is the ability to extract data from a selected region. By specifying a sub-region, a viewer requests that CUMULVS only extract and transmit data within this area. This feature allows viewers to focus on a region of interest, while reducing the volume of data to be processed as well as the bandwidth required to deliver this data.

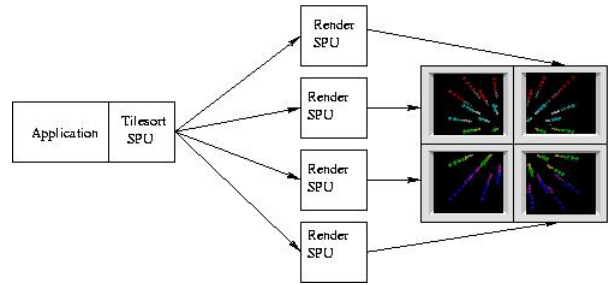


Fig. 2. A Simple Chromium Configuration.

B. Chromium

Chromium provides a framework for parallel computer graphics, based upon the OpenGL graphics system. By replacing the standard OpenGL libraries with those supplied by Chromium, the graphics pipeline can be distributed between multiple machines, with scene generation, rendering, and display possibly performed by different processors. This is achieved by viewing OpenGL commands as a stream. This stream is intercepted, and through a series of Stream Processing Units (SPUs), rendered to produce an image.

A typical scenario is pictured in Figure 2, where a single process generates the OpenGL commands for a scene. These commands are intercepted by a tilesort SPU, that is responsible for examining the geometric objects and determining where they will be processed. In this example, the final display is to be divided between four processors, each responsible for one quarter of the scene. The tilesort SPU is aware of the geometry of the final scene, and sends objects to the SPUs which will render the geometric region of the scene in which those objects lie. A render SPU is running on each of the target processors, where it receives the graphics stream and renders, or draws the final image in a window. In this scenario, the work of rendering is split among four processors, and the final image is suitable for display on a “wall” type display [7].

In a parallel application, multiple tile sort SPUs can be employed to merge the data from parallel graphics streams to multiple rendering SPUs. In this scenario, data, generated by multiple processes in the computation, is transformed locally into graphics primitives, or the OpenGL calls to create the visualization of this data. These streams are then sent, via Chromium’s tilesort SPU, to the

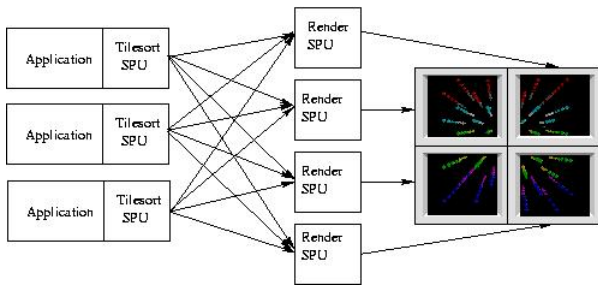


Fig. 3. Parallel Scene Generation in Chromium.

correct render SPU, and the scene is generated and displayed as described above. This scenario is illustrated in Figure 3.

Chromium provides support for programs that were written for a traditional OpenGL environment. Such a program is wrapped by an application faker, which replaces the OpenGL libraries and allows such a program to run in Chromium’s distributed environment. An application is known as chromium aware if it does not require the support of an application faker, but directly performs the tasks necessary to interact with Chromium. These tasks include loading the proper Chromium enabled OpenGL library and providing calls to frame synchronization routines. Chromium aware applications need not be concerned with the final display destination or properties, this information is provided by other elements of the Chromium package. Thus a distributed viewer can be used to display on a single screen or a display wall with no changes to the code.

C. ALCMD - An example application

In this paper, we consider the use of CUMULVS and Chromium to create a parallel viewer for a molecular dynamics code, the Ames Lab Classical Molecular Dynamics simulation (ALCMD)[8]. It simulates the motion of bodies (atoms or molecules) in space by computing the forces between them and applying Newton’s second law of motion to update the locations of these bodies at each time-step. ALCMD is a publicly available FORTRAN77 code built using MPI to implement distributed communications. The code is portable to any platform with MPI support. As a traditional distributed computation it was originally intended to be run in batch mode. We have enhanced the code by utilizing CUMULVS to provide data extraction from the computation for the purpose of visualization. In

addition, we have, as described in the next section implemented a parallel viewer using Chromium to drive a multiple screen display.

III. A PARALLEL VIEWER

A. Justification

The advent of display walls, and visualization clusters, provides the opportunity to produce a new type of CUMULVS viewer. Such a viewer will exploit both the distributed nature of the data and the final scene, while maintaining a minimal code footprint in, and performance impact on the distributed simulation.

B. Design and Implementation

Our parallel CUMULVS viewer consists of an MPI based program, which employs OpenGL and is Chromium aware. This computation consists of an interaction process and a number of visualization processes. To CUMULVS the application appears as a number of independent viewers, each requesting a different region of the data. Each visualization process produces an OpenGL stream, which Chromium distributes and renders. The overall design is illustrated in Figure 4.

In this viewer, the interaction process initially connects with CUMULVS to obtain information about the data fields within the simulation. After extracting information about size and geometric properties of the field of interest, this process divides the field into visualization regions, and assigns each region to a visualization process. After communicating this information to the visualization processes, the interaction process receives viewer requests for geometric transformations such as scaling and rotation, which it communicates with the visualization processes; simulation control requests such as pause and step, which it performs through interaction with CUMULVS; and status change requests such as disconnect and exit.

Visualization processes connect to CUMULVS after receiving information regarding the fields and region bounds from the interaction process. These processes then receive updated data frames from the application and produce the OpenGL calls necessary to render the data, as well as receive and process transformation requests from the interaction node.

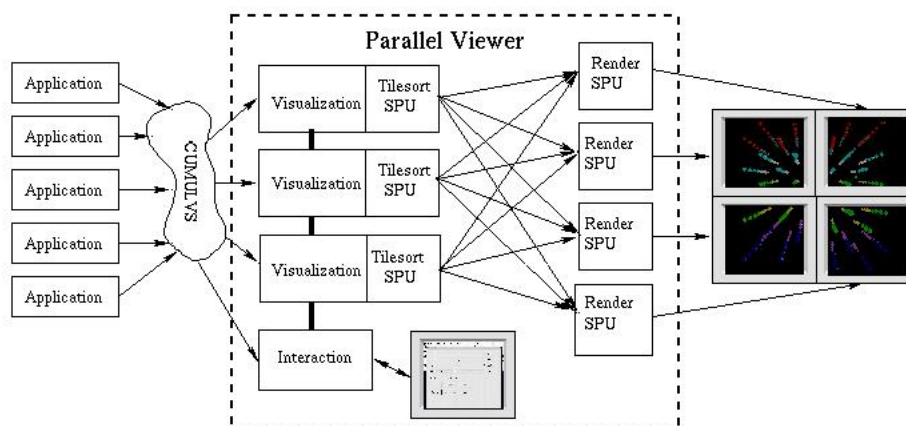


Fig. 4. A Chromium Aware Parallel Viewer for CUMULVS.

Synchronization within the viewer is maintained through interaction with both CUMULVS and Chromium. Viewers and applications within CUMULVS are coupled via an XON mechanism. The application is not permitted to proceed through the next data exchange call until an XON has been received from all attached viewers for the previous call. The interaction node assures that all visualization processes are initially on the same iteration through a MPI barrier. Once initially synchronized, CUMULVS ensures that it is maintained. On the other side of the process, Chromium, through internal barriers, assures that all images are synchronized at the frame level.

The Chromium configuration file contains information about the geometry of the display device, including window geometry and the machine by which each portion of the window will be displayed. Therefore the visualization processes need only deal with scene specific initializations and transformations. All visualization processes make the same calls to establish properties such as the projection models to be used. As mentioned earlier this decoupling of tasks allows a single chromium aware viewer to be used on multiple display media without altering the viewer.

Our prototype viewer is designed for molecular dynamics, where each data point is rendered as a sphere. Sphere location corresponds to molecule location, with one additional parameter displayed by color. The current viewer is built to deal with a parameter representing either the processor of origin or the processor which performed the drawing call

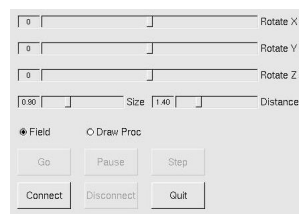


Fig. 5. The Prototype Distributed Viewer Interface.

for each molecule, but this is easily extend to parameters such as Embedding energy, force, or charge density. User interaction is accomplished through a graphical user interface implemented via the Fast Light Toolkit (FLTK), and is currently limited to rotation, scaling, and application control requests. The user interface is pictured in Figure 5.

As a control, a single process visualization viewer was created from the same code base, using compile time directives to remove the superfluous code. This viewer utilizes the same user interface, CUMULVS communications, and OpenGL based visualization routines. It relies on Chromium for rendering and display, and is behaves much like the system illustrated in Figure 2.

C. Performance

The Ames Lab Classical Molecular Dynamics simulation (ALCMD)[8] contains roughly 6,000 lines of code, and works on a variety of platforms including PC (Beowulf) clusters. Parallelization is accomplished by decomposing space into rectangular regions each of which are assigned to a processor. Each process is responsible for all molecules

within its assigned region of space. This code has been instrumented using CUMULVS [5]. It contains code to calculate performance based on the number of floating point operations per second, based on counting the computations performed by the code, rather than using system specific timers on the target architecture.

We have run a number of experiments on our 16 node Beowulf cluster. This cluster is composed of 3.2 GHz P4 processors, with 1.5 GB of main memory, connected with Gigabit Ethernet. Six nodes are connected to flat panel displays forming a 3x2 panel wall with a resolution of 3840x2048. In the trial runs, the molecular dynamics simulation was executed on four nodes, with the number of molecules varying from 2^8 to 2^{20} . Note that this means that the largest simulations involves over a million molecules, and requires the transmission of the positions of these, and the rendering of over a million spheres corresponding to the molecules. Chromium renderers were each given their own processor with attached display, and each drawing node in the visualization was assigned to a different node. In the most complex case, 13 different nodes were employed, four to run the simulation, five to run the CUMULVS viewer (one for interaction and four to generate drawing requests), and four nodes to render and display.

Performance measurements were conducted using a custom CUMULVS frames-per-second (FPS) viewer constructed as part of a toolkit to accompany CUMULVS. This viewer connects to an application via CUMULVS and requests a single scalar parameter. By recording the frequency with which CUMULVS updates the value of this parameter, the viewer is able to compute the number of iterations through the main loop of the application per second. If the application is tightly tied to a visualization client, this is also equivalent to the number of frames per second achieved by the visualization client. See [9] for more details.

The results of our experiments are summarized in table I. Each column in the table represents running the simulation with 2^n atoms, where n is the column label and entries are recorded in frames per second, as reported by the frames-per-second viewer. The first row, labeled *FPS*, represents the application running on four processors, with only the FPS

viewer attached, and can be used as a baseline for performance. The next four lines are labeled with a *D* for tests involving the distributed CUMULVS viewer client, and *S* for tests involving a single processor client. The number of rendering/display processors is indicated by the number following the dash. So, for example, row *D-4* contains the frames per second rate, using the distributed CUMULVS viewer client (five processors - one for interaction and four to generate drawing requests), which is connected to four different processors for rendering, much like the configuration pictured in figure 4, while *S-4* represents a situation similar to that in figure 2, where the Chromium viewer consists of a single processor for both visualization and interaction. Thus the Parallel Viewer component of Figure 4, utilizes 9 processors for *D-4* (4 for visualization, 1 for interaction and 4 for rendering), *D-1* utilizes 6 processors (4 for visualization, 1 for interaction and 1 for rendering), *S-4* utilizes 5 processors (1 for visualization and 4 for rendering), and *S-1* utilizes 2 processors (1 for visualization and 1 for rendering).

The final three lines of the table represent simple analysis of the results. The first line, *%Dec FPS to D-4*, represents the percent decrease from the base case to the *D-4* case. *% Dec D-4 to S-1* represents the percentage decrease in performance between the *D-4* and the *S-1* case. The final line, is calculated by dividing the frame rate for the parallel viewer by that for the sequential viewer. It indicates the improvement in frame rate achieved by using the distributed CUMULVS viewer and four display nodes.

As one might expect, the configuration *D-4* performs better than the other configurations of the parallel viewer, except for the smallest case with 2^8 (256) molecules. We also note that, with the same exception, *D-1* performs better than *S-1*. That is, it is more effective to parallelize the viewer component than the render/display component. As might also be expected, *S-1* performs worse than the other configurations, except for the smallest two cases. As the number of molecules increases the frame rates using *D-4* approaches 50% of those achievable without any viewer, compared with only 3% for the *S-1* viewer, and 8% for the next best viewer (*D-1*). This indicates that the use of both CUMULVS and Chromium components does produce a signif-

TABLE I
PERFORMANCE RESULTS (FPS)

Atoms (Log Base 2)	8	9	10	11	12	13	14	15	16	17	18	19	20
FPS	392	321	237	153	80	48	22	11.5	6.4	3.3	1.7	0.9	0.45
D-4	97	96	93	85	48.3	25.8	13	6.5	3.2	1.5	0.8	0.4	0.2
D-1	97	96	58	38	21.9	11	5.5	2.7	1.4	0.65	0.33	0.17	0.08
S-4	128	47	28	15.2	7.8	3.5	1.8	0.8	0.4	0.2	0.1	0.05	0.03
S-1	128	49	20	10	5.8	3.8	1.5	0.8	0.4	0.2	0.1	0.05	0.03
% Dec FPS to D-4	0.75	0.7	0.61	0.44	0.4	0.46	0.41	0.43	0.5	0.55	0.53	0.56	0.55
% Dec D-4 to S-1	-0.32	0.49	0.78	0.88	0.88	0.85	0.88	0.88	0.88	0.87	0.88	0.88	0.85
Speedup: D-4/S-1	0.76	1.96	4.65	8.5	8.33	6.79	8.67	8.13	8	7.5	8	8	6.67

icant improvement in the frame (computation) rate achievable.

IV. FUTURE WORK

We intend to consider in more detail the various contributions that impact the performance of such parallel viewers. In particular, we will consider the amount of time spent in communication, and in computations on the CPUs and the GPUs of the processors. We expect that the tuning suggested by such considerations will lead to improvements in the performance of distributed viewers. Finally we wish to explore the scalability of our system by interacting with a much larger computational cluster and increasing the size of the display wall.

Furthermore we intend to extend our prototype to build a more generic and robust viewer, capable of displaying multiple parameters for each molecule. Ultimately we wish to build a distributed viewer capable displaying both molecular data and the regularly gridded data formats supported by CUMULVS. This viewer, when complete, will become part of our proposed toolkit to accompany CUMULVS.

REFERENCES

[1] S. C. Ahalt and K. L. Kelley, "Blue-collar computing: Hpc for the rest of us," *Cluster World*, vol. 2, pp. 10–18, November 2004.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds., *Sourcebook of Parallel Computing*. Morgan Kaufmann, 2003.

[3] B. Topol, J. T. Stasko, and V. S. Sunderam, "Integrating visualization support into distributed computing systems," in *International Conference on Distributed Computing Systems*, 1995, pp. 19–26.

[4] J. Kohl, P. Papadopoulos, and G. Geist, "Cumulvs: Collaborative infrastructure for developing distributed simulations," 1997. [Online]. Available: <http://douc54.cs.edinboro.edu/bennett/papers/cs/middleware/kohl97cumulvs.html>

[5] D. Bennett and P. Farrell, "Experiences instrumenting a distributed molecular dynamics program," Kent State University, Department of Computer Science, Tech. Rep. TR-KSU-CS-2006-04, 2006, submitted to *PACISE-06*.

[6] J. A. Kohl, T. Wilde, and D. E. Bernholdt, "Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance," *International Journal of High Performance Computing Applications*, submitted, submitted. [Online]. Available: <http://douc54.cs.edinboro.edu/bennett/papers/cs/middleware/cumulvs/kohl04b.html>

[7] M. Hereld, I. Judson, J. Paris, and R. Stevens, "Developing tiled projection display systems," 2000.

[8] D. Turner and J. Morris, "Alcmd," www.cmp.ameslab.gov/cmp/CMP_Theory/cmd.

[9] C. Stein, D. Bennett, P. Farrell, and A. Ruttan, "A steering and visualization toolkit for distributed applications," Kent State University, Department of Computer Science, Tech. Rep. TR-KSU-CS-2006-03, 2006, submitted to *PDPTA-06*.