Experiences Instrumenting a Distributed Molecular Dynamics Program

Dan Bennett
Edinboro University of Pennsylvania
dbennett@edinboro.edu

Paul Farrell
Kent State University
farrell@cs.kent.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. In this paper we discuss this middle-ware and our experiences instrumenting a parallel computation. We present the impact of the instrumentation and some ideas for additional functionality.

KEY WORDS
Cluster computing, scientific visualiation, middle-ware, steering.

1. 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.

2. Software Involved

In this paper, we consider the use of the CUMULVS steering and visualization middle-ware package to instrument the Ames Lab Classical Molecular Dynamics (ALCMD) package. The following sections will describe these packages.

2.1 Software Involved

CUMULVS [3] is an example of a scientific visualization and steering middle-ware tool for distributed environments. Such packages 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, to the application. Originally designed to provide interaction with programs built to run in a batch queue environment, CUMULVS contains a library of routines which a programmer can use to instrument, or add data extraction routines, to an existing computation. Once the application has been instrumented, multiple users can employ one of the supplied viewers to attach to a running computation for interactive steering and visualization. The 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.

In addition to CUMULVS there were a number of steering and visualization packaged developed in the 1990's, including SCIrun [4], CSI [5], MAGELLEN [6] and MOSS [7]. Of these packages, only CUMULVS, developed at Oak Ridge National Labs, and SCIrun, from the University of Utah, appear to be ongoing, supported research projects. SCIrun has become an integrated development environment, with a visual language for integrating modules much like OpenDX or AVS. SCIrun is best used when developing an application from scratch, so that it can be customized to run within this environment. CUMULVS is a more appropriate choice for working with existing packages.

As CUMULVS evolved, two very importation improvements were made to the package: the ability to checkpoint a distributed computation [8] and support for data exchange between multiple computations [9]. A checkpoint is a snapshot of the state of a computation, which can

be used to restart a run which has failed or to migrate a task to a processor with a lower load. The programmer instruments the application to specify data required for a restart, and provides a routine which will perform the restart when supplied with this data.

The task of data exchange between computations is a natural extension to CUMULVS. In this model, a computation acts as a viewer which consumes, as input, data from a second computation. It is not required that the computations be designed to cooperate with each other. In this way an independent simulation predicting the industrial waste emissions of a city can be coupled with a simulation of the flow of water within a lake to produce a larger simulation modeling the production and distribution of pollutants within this body of water.

We are working on a project which will involve instrumenting an existing simulation for visualization, steering, check pointing and process interaction. For this reason, we have selected CUMULVS for further study and possible enhancement. In our initial experiments, we chose to instrument ALCMD, a Molecular Dynamics simulation.

2.2 Molecular Dynamics

Molecular dynamics [10] (MD) computations are concerned with simulating the motion of bodies (atoms or molecules) in space. In such a simulation, the force between all bodies in the model is computed and Newton's second law of motion is applied to update the locations of these bodies at each time-step. While the underlying algorithm for such a simulation is $O(n^2)$, where n is the number of atoms, it is often desirable to run computations with 10^{10} or more atoms for 10^6 or more time-steps. The Lennard-Jones potential describes the interaction of any two molecules, and is a function of the distance r between those molecules

$$u(r) = 4\varepsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \qquad (1)$$

$$f(r) = \frac{24\varepsilon}{r}[2(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}]\hat{r} \qquad (2)$$

In (1) and (2), ε and σ are constants related to the type of atom involved in the simulation, with σ = 3.4Å for liquid argon. It can be noted that r will quickly dominate the computation, and it is normal to assume that for a cutoff distance, r_d, if $r>r_d$ then f(r) = 0. This assumption allows for reasonably efficient decomposition of the task

The Ames Lab Classical Molecular Dynamics simulation (ALCMD) [11] is publicly available FORTRAN77 code built on top of MPI. It contains roughly 6,000 lines of code, and works on a variety of platforms including 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 is an excellent example of a target simulation for instrumentation: it runs in batch queue mode, with data written to a file for post processing; it contains no provisions for interaction with the user; and it has extensive timing measurements built in. This includes the computation of megaflops, which are computed based on actual operations performed in the code

3. Instrumenting the Code

Our goals, in instrumenting the ALCMD code using CUMULVS, were to test the ease of instrumentation of a FORTRAN based MD program with CUMULVS, to check the performance costs involved in such an instrumentation, to look for possible improvements or extensions to the CUMULVS package, and to look for possible additional tools which may be useful when employing CUMULVS.

3.1 CUMULVS Instrumentation Routines

Instrumenting an application consists of a single call to the CUMULVS initialization routine, registering parameters or scalar variables, and registering fields or distributed parallel data, as in (3).

```
stvfinit("appname",TAG,TotalPros
            procID, errorReturn)        (3)
```

After initialization, parameters are defined by a single call as in (4).

```
stvfparamdefine(addres of var,
   "Name", STVtype, ViewType, ID)       (4)
```

It should be noted that parameters are designed for steering. Values that are changed by a viewer are changed in the application by CUMULVS. Finally, one must define the datafield for the application. Data fields in MD simulations are known as particle fields. They have no universally defined parallel decomposition, and thus are the most difficult data type to describe in CUMULVS. Extraction of this data is a two stage process; first a routine to access individual particles within the particle field is defined, (as in (5) then the access routines to characteristic fields of individual particles are specified, as in (6).

```
stvfparticledefine("Name", dim, ub,
   lb,  numPoc, getPart, putPart,      (5)
   pID)
stvfpfielddefine("Serial",pID,
   STVINT, gSerial, 0, pSerial, 0,
   flag, fieldID)
                                       (6)
stvfpfielddefine("Energy",pID,
   STVDOUBLE, gEnergy, 0,
   pEnergy, 0, flag, fieldID)
```

In (6), the first line provides access to a particle serial number, while the second defines how to access the energy of a single particle. It is further required that routines to access individual particles, and the data fields associated with those particles be provided. If check-pointing is desired, routines to restore the values must be provided as well.

Instrumentation is completed by placing a call to allow CUMULVS to extract new data values every iteration through the main loop, as in (7).

```
stvfsendtofe(errorCode)                (7)
```

After updating the data values, CUMULVS provides information about changing parameters hrough a call as illustrated in (8).

```
stvfisparamchanged(paramId,info)       (8)
```

If this routine returns a nonzero value, then a viewer has changed a parameter, and the application should take appropriate action.

In our experiments, we extracted three scalar visualization parameters, a scalar steering variable and three different fields associated with particle data. This added a total of 295 lines of code to the application, including comments and white space. This represents a 4.5% increase in the number of lines of code. This code, however, is for the most part contiguous, and can be removed with three pairs of pre-processor directives.

4 Results

After instumenting ALCMD, we conducted a series of performance runs to determine the impact of the instrumentation on the code. These results are summarized below followed by a discussion of some areas in which we feel that the overall usefulness of the CUMULVS package can be improved.

4.1 ALCMD Performance Analysis

All measurements for this paper were performed on a portion of a 16 node cluster consisting of 3.2 GHz Pentium IV processors, connected by gigabit Ethernet. Each is equipped with 1.5 GB of memory. The cluster runs a generic Fedora Core 3 version of the Linux operating system, with no additional performance tuning. Thus computations are subject to interruption by periodic system processes, and the maximum amount of memory that can be allocated at compile time is limited.

To establish the importance of parallel operation in this code, an initial set of trials were run where the number of processors was varied from 1 to 4, and the number of molecules was varied from 2^{10} to 2^{21}. The upper limit, 2^{21}, is the largest size run which can be accomplished with four processors due to current kernel configuration. As can be seen in Figure (1), for cases larger than 2^{13} molecules, the megaflops rating of the code is directly related to the number of processors employed. Speedup achieved was between 1.4 (for two processors and 2^{10} molecules) and 3.72 (for 2^{19} molecules on four processors). These and all other performance runs were conducted on a quiescent cluster, that is one with no other user processes running, and each trial consisted of 100 time steps of the simulation.

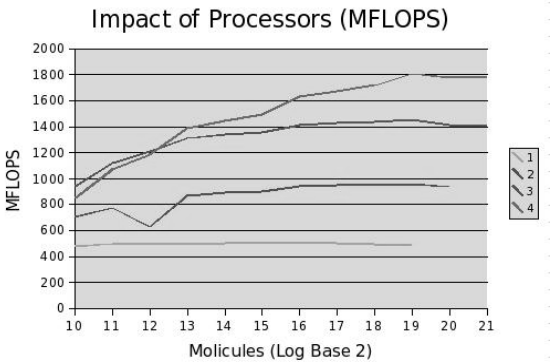

Figure 1: Time Comparison 1-4 processors

A second set of timing runs was conducted, again varying the number of atoms as above, but with different versions of the code. To determine the impact of communications on the base simulation, the simulation was recompiled to employ the MPIP monitoring software. MPIP [12] collects and summarizes information regarding time spent in MPI calls. The timing results are summarized in Figure 2, which displays both total run time and time spent in MPI routines, while the relationship between computation and communications time is displayed in Figure 3.

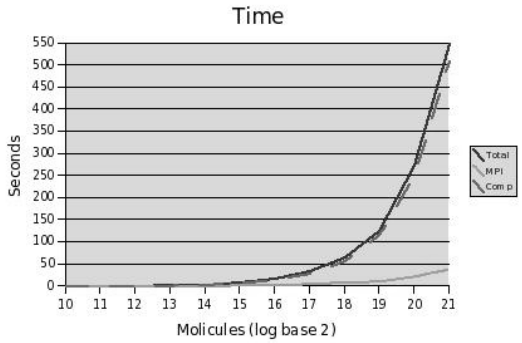

Figure 2: Time Comparison

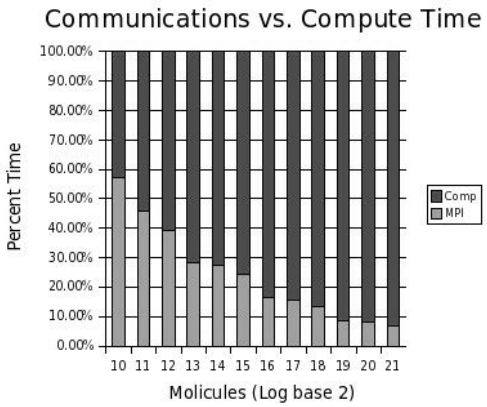

Figure 3: Percentage Time, Communications vs Computaion

The code was run a second time with CUMULVS routines compiled in, but with no viewers attached. This produced variations of less than 0.4% in performance, confirming the expectation that instrumentation alone would have essentially no impact on the performance of the application. This result is well within reason, since instrumentation consists of two calls within the main loop, each of which evaluates a single logical expression to determine that no viewer is present.

Several runs were conducted with single and multiple viewers present. For a single viewer extracting scalar data, the largest impact was 2.5%, which occured when processing 2^{11} molecules. The impact was 0.1% or less for larger sized runs. Multiple connected scalar viewers had a larger impact, as much as 56% for small numbers of molicules, but again the impact of such viewers became negligible (typically 0.3%), when the number of molicules was increased. A summary of run times is given in Table 1.

Performance data was not collected for runs with less than 2^{11} molecules, as the processes had insufficient runtime for viewers to successfully attach.Performance, in megaflops, for all of these runs is illustrated in Figure 4.

Atoms	10	11	12	13	14	15	16	17	18	19	20	21
Raw	0.48	0.79	1.41	2.40	4.70	9.12	16.55	34.13	66.53	126.84	274.29	548.36
MPIP	0.50	0.80	1.42	2.41	4.70	9.14	16.53	34.15	66.53	126.75	274.36	548.96
CUMULVS	0.48	0.79	1.41	2.41	4.70	9.16	16.54	34.09	66.53	126.88	274.46	549.13
1 Client		0.81	1.43	2.43	4.74	9.18	16.57	34.12	66.52	126.74	274.52	548.72
1 client		0.79	1.43	2.43	4.73	9.17	16.57	34.08	66.64	127.26	274.13	548.93
2 Clients		0.94	2.20	3.95	5.18	9.32	16.60	34.24	66.59	126.68	274.13	549.59

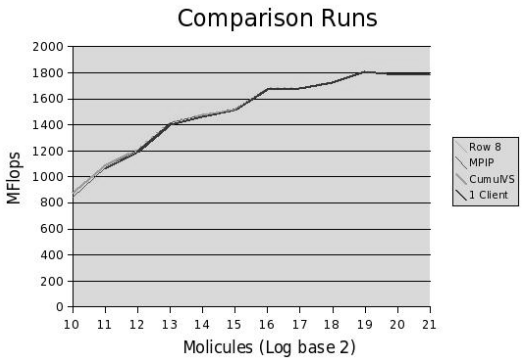

Figure 4: Mflops Comparison

Optimal performance is achieved when the percentage of time spent in communication is minimal. Figure 2 demonstrates that computation time increases dramatically as the number of molecules is increased, while Figure 4 demonstrates that increasing the number of molecules leads to maximum performance as measured by megaflops.

4.2 Viewers

The most apparent area for improvement within CUMULVS is the lack of lightweight viewers. The CUMULVS code comes with a simple "slicer'' viewer that will display a 2-D slice of three-dimensional data, as well as a simple text based steering client. These viewers provide strong examples on how to construct viewers but have a number of features which limit their usability. We have begun creating a toolkit of simple clients to provide easy interaction with CUMULVS instrumented simulations, and believe that this toolkit will enhance the usability of this package. A description of some of these is contained in [13]. Several of these clients were employed in our timing experiments mentioned above.

In addition, we have developed a prototype distributed molecule viewer using the CHROMIUM library. Initial tests show near linear improvement over non-distributed viewers. For more information, please see [14].

4.3 Problems Encountered

The most significant, yet perhaps the easiest problem to overcome is a lack of complete documentation. CUMULVS comes with a partially completed users guide, which can be supplemented with tutorials from several workshops [15, 16] and a large number of working examples. This documentation is focused on instrumenting code, with very limited information about creating viewers.

A second problem discovered was some interoperability issues between FORTRAN and CUMULVS. In several instances, CUMULVS passes variables by value. However, this is easily overcome by using the FORTRAN90 LOC function. A slightly more troublesome problem is that CUMULVS expects all molecule locations to be represented by integers. This was overcome by scaling molecule locations, represented in the simulation as floating point numbers between 0 and 1, by 1000. In each case, a minor alteration of the CUMULVS API would lead to much more FORTRAN friendly code.

5. Conclusions and Future Directions

It is clear from our experiments that instrumentation has very little impact on the performance of applications. Additionally, the minimally invasive nature of the instrumentation makes it easy to remove in environments where it is not supported. Furthermore, we have found that clients monitoring scalar variables also have minimal impact on performance, but greatly enhance the ability to monitor and interact with running simulations. Indeed, we have found ourselves creating new client tools to assist in our work (see [13]).

These experiences have deepened our belief in the importance of a toolkit to interact with instrumented applications. We are in the process of designing and implementing a number of such clients, and will prepare a general open source release of these tools when progress warrants. We believe that the size of this toolkit will increase as more experiments are performed with check-pointing and distributed field manipulation.

Finally, we need to further investigate the performance impact of distributed data field extraction. We have designed and are currently implementing a simple statistics client to extract and analyze a distributed data field. This client will not only provide summary information for distributed fields, but will provide a simple

efficient platform for testing the performance of field extraction.

4. Acknowledgements

This work has been supported in part by NSF ITR 0081324. The braveheart beowulf cluster was provided by funds from the Provost's iniative funds.

References:

[1] S. C Ahalt, & K. L. Kelley, Blue-collar computing: Hpc for the rest of us, *Cluster World 2*(11), 2004 10-18.

[2] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, & A. White, editors *Sourcebook of parallel computing* (Morgan Kaufmann, 2003).

[3] G. Geist, J. Kohl, & P. Papadopoulos, Cumulvs: Providing fault tolerance, visualization, and steering of parallel applications, *SIAM*, August 1996.

[4] Scirun, www.cs.utah.edu/sci/projects/sci/comp.html.

[5] R. van Liere, J. D. Mulder, and J.J. van Wijk. Computational steering, *Future Gener. Comput. Syst*., *12*(5):441-450, 1997.

[6] J.S. Vetter, Experiences using computational steering on existing scientific applications, *Proc. Ninth SIAM Conf. Parallel Processing*, San Antonio, Tx, 1999.

[7] G. Eisenhauer, *An object nfrastructure for high-performance interactive applications* (PhD thesis, Georgia Institute of Technology, May 1998).

[8] J. Kohl, T. Wilde, & D. E. Bernholdt, Cumulvs: Interacting with high-performance scientific simulations, for visualization, steering and fault tolerance, *International Journal of High Performance Computing Applications*, submitted.

[9] J. Kohl, 2nd mxn Working Group Meeting report, *MxN Working Group Meeting*, 2004, Working Group Notes.

[10] D. C. Rapaport, *The art of molecular dynamics simulation* (Cambridge University Press, 1995).

[11] D. Turner, & J. Morris, Alcmd, www.cmp.ameslab.gov/cmp/CMP_Theory/cmd.

[12] J.S. Vetter, & F. Mueller, Communication characteristics of large-scale scientific applications for contemporary cluster architectures, *Journal of Parallel and Distributed Computing*, *63*(9), 2003, 853-865.

[13] C. Stein, D. Bennett, P. Farrell, & A. Ruttan A toolkit for cumulvs, *Technical Report TR-KSU-CS-2006-03, Kent State University, Department of Computer Science*, 2006, Submitted to *PDPTA- 06*.

[14] D. Bennett, P. Farrell, & C. Stein. A chromium based viewer for cumulvs, *Technical Report TR-KSU-CS-2006-02, Kent State University, Department of Computer Science*, 2006, Submitted to *PDPTA-06*.

[15] J. Kohl, Cumulvs hands-on notes - acts toolkit workshop, *ACTS Toolkit Workshop*, August 2003, Workshop Notes.

[16] J. Kohl, Cumulvs tutorial - acts toolkit workshop, *ACTS Toolkit Workshop*, August 2003, Workshop Notes.