Lightweight Computational Steering of Very Large Scale Molecular Dynamics Simulations

David M. Beazley

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112
beazley@cs.utah.edu

November 21, 1996

Collaborators: P. Lomdahl, S. Zhou, B. Holian, T. Germann, W. Kerr, N. Jensen
What is this talk really about?

- Dealing with huge amounts of data
- Total frustration
- We have come up with a solution that emphasizes:
  - Memory efficiency
  - High performance
  - Support for production simulations
  - Simple tools
  - No special purpose hardware

Really, this talk is about how to manage very large simulations (10-100 million atoms) using cheap workstations, slow networks, and simple tools.
Short Range Molecular Dynamics

- Solving $F=ma$ for large number of atoms.
- Finite range of interaction
- Physics incorporated into potential energy functions.

- Our interests:
  - 3D Fracture
  - Dislocations
  - Shock waves
  - Friction
  - Ductile-brittle transition

Fracture simulation with 104,000,000 atoms (1994)
The SPaSM Code

- **Scalable Parallel Short-range Molecular dynamics**
- Runs on CM-5, T3D, SP-2, Fujitsu, Sun, and SGI.
- Written in ANSI C with explicit message passing.
- One of 1993 Gordon Bell Prize winners.
- Has performed production simulations with 100 million atoms, test simulations with up to 300 million atoms (1024 node CM-5).
- Is primarily a materials research code.
Too Much Data!

- Fracture simulation with 104,000,000 atoms:
  - ~13 Gbytes of RAM needed to run the simulation
  - Ran for 180 hours on 512 node CM-5 (in 6 hour increments)
  - Performed over 900 Gbytes of I/O during run.
  - Generated ~40 output files
  - Each output file: 1.6 Gbytes (X,Y,Z,KE)
  - Would have generated more files if we had more disk space.

- Problems:
  - Where do you put the data?
  - How do you get it there?
  - How do you analyze it?
  - What do you analyze it with?
  - Is this even a good idea?

- Consider this:
  - We would like to generate hundreds of files with more data.
  - Oak Ridge ran a 1 billion (1,000,000,000) atom simulation in 1994.
  - On proposed ASCI machines, we could probably run simulations with as many as 10 billion atoms. YIKES!
Total Frustration

• We’re using supercomputers to try and solve large problems and we’re getting crushed by the data.

• Just buy a graphics workstation...
  • Not enough computational power.
  • Not enough memory.
  • Too expensive.
  • Virtually useless if you can’t get the data to it.
  • Why would a sane person think that a workstation can handle data from something run on 1000 workstations?

• Try this “easy to use” visualization package...
  • Not easy to use.
  • Incapable of dealing with large amounts of data.
  • Mostly just a big waste of time...

• The bottom line:
  • Most users have relatively inexpensive workstations (Sun, HP, SGI)
  • Tools don’t work on real problems.
  • Laziness--we want to work from our office.
“Frameworks” and “Environments”

- If we build the perfect computational problem solving environment, they will come...

- This is the “parental” approach...
  - Play by the rules.
  - Share.
  - Be nice.
  - (kind of like living Utah actually...)

- What fantasy-land are these people living in?
  - Parallel computers don’t work.
  - Compilers don’t work.
  - Tools don’t work.
  - Debuggers don’t work.
  - Programs don’t talk to each other.
  - ... and yet somehow your code works. Hmmm...

- Who said chaos is a bad thing?
The Real Problems

- Trying to work with an overwhelming amount of data using underpowered tools and machines.

- Decoupled simulation and analysis.
  - Simulation doesn’t perform any analysis.
  - Analysis tools don’t know much about the simulation.

- Software that is too complicated.
  - A solution that is more complicated than the original problem is NOT a solution.
  - We don’t want a magical black box either.

- Assumption that we know what we want.
  - Scientific applications never die.
  - Problems are always changing.
  - New problems require new approaches.
  - If I only knew what I was going to do next week...
Computational Steering

- Latest buzzword? (well, maybe)

- In a nutshell:
  - Combine simulation, data analysis, visualization.
  - Make all of these things interact with each other.
  - Allow the user to control and manipulate everything.
  - Results of analysis can be used in a simulation.
  - Primarily a diagnostic, debugging, and exploration tool.

- Unfortunately, most steering approaches involve:
  - Special purpose hardware (high-end workstations).
  - High speed networks.
  - Sophisticated visualization (or possibly virtual reality).
  - Complex software.

- However, none of these things are really required...

- In fact, you may not want any of these things when working with really big problems.
The Finite Memory Problem

- Memory is a finite resource.
- Simulation and visualization are both memory intensive.
- Combination of the two must favor simulation.
  - If not, then what is the point?
  - Fortunately, visualization can use data structures in the simulation.
Bandwidth Constraints

- Once your data leaves the machine, you’re dead.
- Even moving data around on the same machine is generally a bad idea.
- The information superhighway? Yeah, whatever...
- Sensible approach:
  - Run simulation on a supercomputer.
  - Visualize and analyze data on the same supercomputer.
  - Send as little data as possible to user’s machine.
Software Complexity

Good software is hard to write, so it should be hard to use...

- Steering systems should emphasize simplicity, modularity, and memory efficiency.
- Physics, not computer science!
- Need to be able to use and modify the code.
- Scientific applications have a way of sticking around for a while.
- User interfaces and “frameworks” come and go so don’t make a simulation code depend on one.
- Don’t force everyone into a rigid programming model.
- Code reuse!
Steering with Scripting Languages

- Use a scripting language to glue stuff together.

- Approach is similar to that used in:
  - MATLAB
  - Mathematica
  - Maple
  - etc...

- User can control code by typing commands interactively or with scripts.
Benefits of Scripting Languages

- Memory efficient.
- Work well over slow networks.
- Portable.
- Easy to extend with new functions.
- Allow both interactive and scripted control.
- Rapid prototyping and debugging.
- Can form the foundation for more sophisticated interfaces (using Tk for instance).

A new idea? **No.**
An time-proven idea that works? **Yes.**
Just a few minor problems...

- How does one find a scripting language that will run properly on a parallel machine?
- How do you go about using it to build a steering system?
- How do you do it without making things too complicated?
Parallel Python

- We have chosen to use the Python language.
  - Small core that’s easy to parallelize.
  - Exceptionally clean syntax.
  - Object oriented.
  - Relatively easy to extend.
  - Portable (runs on UNIX, Mac, Windows).
  - Large collection of modules available.
  - Being used increasingly in physics applications.

- We have modified it to work properly in parallel.
  - Parallel I/O is the problem.
  - Required less than 5 lines of modification to Python source.
  - Runs on top of CMMD, Cray shared memory, and MPI.

- User controls system using Python commands and scripts.

- On workstations, we can use Perl and Tcl as well (but more on that in a minute).
Python Example

A Module

C functions

- Python provides a precise mechanism for setting up and controlling simulations.

- Commands directly map onto underlying C code.

- But how does one go about creating a new command?

```python
alpha = 7
cutoff = 1.7
# Set up potential energy
init_table_pair()
import morse
morse.maketable(alpha, cutoff, 1000)
# Set up initial condition
ic_crack(80, 40, 10, 20, 5, 25, 5, alpha, cutoff)
set_initial_strain(0, 0.017, 0)
set_strainrate(0, 0, 0.001)
set_boundary_expand()
# Run it
timesteps(1000, 10, 50, 100)
```
SWIG

- A compiler that takes ANSI C/C++ declarations and makes them available in scripting languages:
  - Python
  - Tcl/Tk
  - Perl5

- Fully automated (requires no user intervention).

- Supports almost all C/C++ datatypes.
  - Pointers
  - Structures and classes.

- Requires no code modifications.

- Extremely simple to use:
  - Creating an OpenGL module: ~20 minutes.

- Free and fully documented.
// crack.i. Interface file for crack problem.
#define module crack
{%
#include "crack.h"
%
#include SPaSM.i
#include analysis.i

extern int ic_crack(int nx, int ny, int nz, double lc);
extern void init_lj(double epsilon, double sigma, double cutoff);
extern void set_boundary_periodic(void);
extern void set_boundary_free(void);
extern void set_boundary_expand(void);
extern void apply_strain(double ex, double ey, double ez);
extern void set_initial_strain(double ex, double ey, double ez);
extern void set_strainrate(double exdot0, double eydot0, double ezdot0);

SPaSM [182] > SPaSM_geometry(0,0,0,50,50,50,2.5)
SPaSM [182] > init_lj(1.0,1.0,2.5)
SPaSM [182] > set_boundary_periodic()
SPaSM [182] > ic_crack(100,100,20,20.0)
SPaSM [182] >
Simple Parallel Graphics Library

- Have written a **simple** graphics library.
  - 8 bit graphics.
  - z-buffered 3D imaging.
  - Simple API.
  - Efficiency is everything...

- Runs on every node.

- Produces GIF files as output.

- Supports 2D and 3D.

- **Performance:**
  - 35 Million atoms on 16 node T3D: ~ 10 seconds
  - 104 Million atoms on 512 node CM-5: ~ 15 seconds.
Data Analysis

- Have built an object oriented data analysis system in C and Python.
- Supports feature extraction, visualization, etc...
- Allows an unlimited number of simultaneous images and can be scripted.
- Available during simulation and postprocessing.

Example:
- Run a 100,000,000 atom simulation and produce 1000 datafiles.
- Make 10 movies from the data.

1. Set up 10 different views from initial condition.
2. Run the simulation: produces 1000 datafiles and 10,000 GIF images.
3. Look at datafiles later if necessary (easy enough).

VS.

1. Run the simulation: produce 1000 datafiles.
2. FTP 3 Tbytes of data to your workstation.
3. Attempt to set up 10 views and make movies.
4. Give up after alot of frustration.
Remote Data Analysis

How to build a remote data analysis system in an afternoon:

Big Parallel Machine

Supercomputer:
1. Open socket
2. Make image
3. Composite
4. Make GIF
5. Node 0 sends

Workstation:
1. Open socket
2. Listen for image
3. Run ‘xv’
4. Repeat 2-3.

Sounds crazy, but we can interactively visualize 100 million atom datasets over a standard T1 internet connection.
Putting it All Together

Visualization

Simulation

Data Analysis

Graphics

Tcl

Parallel I/O

Python

Perl

Message Passing

CHAOS

Stand-alone C applications. (Batch oriented)

Flexible Parallel applications using C and Python.

Bizarre combinations of everything running on a workstation.
Interactive Message Passing

- Possible to write message passing code in Python scripts.
- Parallel Debugging
- Rapid prototyping of message passing applications.

```python
>>> from pvm3 import *
>>> me = pvm_get_PE(pvm_mytid())
>>> nproc = pvm_gsize("")
>>> if me == 0:
...     a = [1,2,3,4]
...     for i in range(1,nproc):
...         pvm_initsend(PvmDataRaw)
...         pack_list(a)
...         pvm_send(i,1)
...     else:
...         pvm_recv(0,1)
...     a = unpack_list()

>>> pprint(a,range(0,nproc))
PN 0 : [1,2,3,4]
PN 1 : [1,2,3,4]
PN 2 : [1,2,3,4]
PN 3 : [1,2,3,4]
```
Input/Output Problems: Solved!

- Huge amount of time can be spent simply transferring data between codes.
  - Reading initial conditions.
  - Setting up physical parameters
  - Validation of experiments

- Of course, no one uses the same file format...

- But scripting languages are great at processing weird file formats.

- In short, dealing with bizarre files is not much of a problem.
Using Other Packages

Use the best tool for the job. SWIG allows different packages to be glued together with a common interface.

- SPaSM
- Our graphics library
- MATLAB
- Tcl/Tk
- Developed in a single afternoon.
Conclusions

• Steering system has been in operation for last 15 months and has allowed us to work with ANY sized simulation we can run.

• No noticeable impact on performance and minimal impact on memory use.

• Approach encourages better programming:
  • Encourages modular design.
  • Development of well-tested and debugged C libraries.
  • Event-driven programming.
  • Our original physics code actually shrunk by more than 25%.

• Maintains coding simplicity
  • No modifications to C source necessary.
  • Recompiles in about 1 minute with a 10 line Makefile.
  • Doesn’t hide important details.
Conclusions (cont...)  

• No C++.  
• No Virtual Reality.  
• No Graphics Workstations.  
• No Java.  
• No IWAY.  

=> No problem. It is possible to build a powerful steering system that works with huge amounts of data, works with existing software, is easy to modify, and let’s us do real work from any UNIX workstation.  

Of course, nothing in our approach prevents us from using any of the above.
Acknowledgments

- Department of Energy.
- Los Alamos National Laboratory.
- Advanced Computing Laboratory.
- University of Utah, Department of Computer Science.
- Lawrence Livermore National Laboratory.
- Cornell Theory Center.
- The Coca-Cola™ company.
Availability

- Python

  http://www.python.org

- SWIG

  http://www.cs.utah.edu/~beazley/SWIG

- Graphics library and parallel Python

  To be released
Limitations

• Approach may not be suitable for large legacy applications.

• Tools don’t work so well if you go overboard with C++
  • Avoid templates and multiple inheritance.

• Designed for power-users who want precise and powerful program control.

• Fortran not currently supported (but maybe that’s a good thing).

• Use whatever works best for your application.