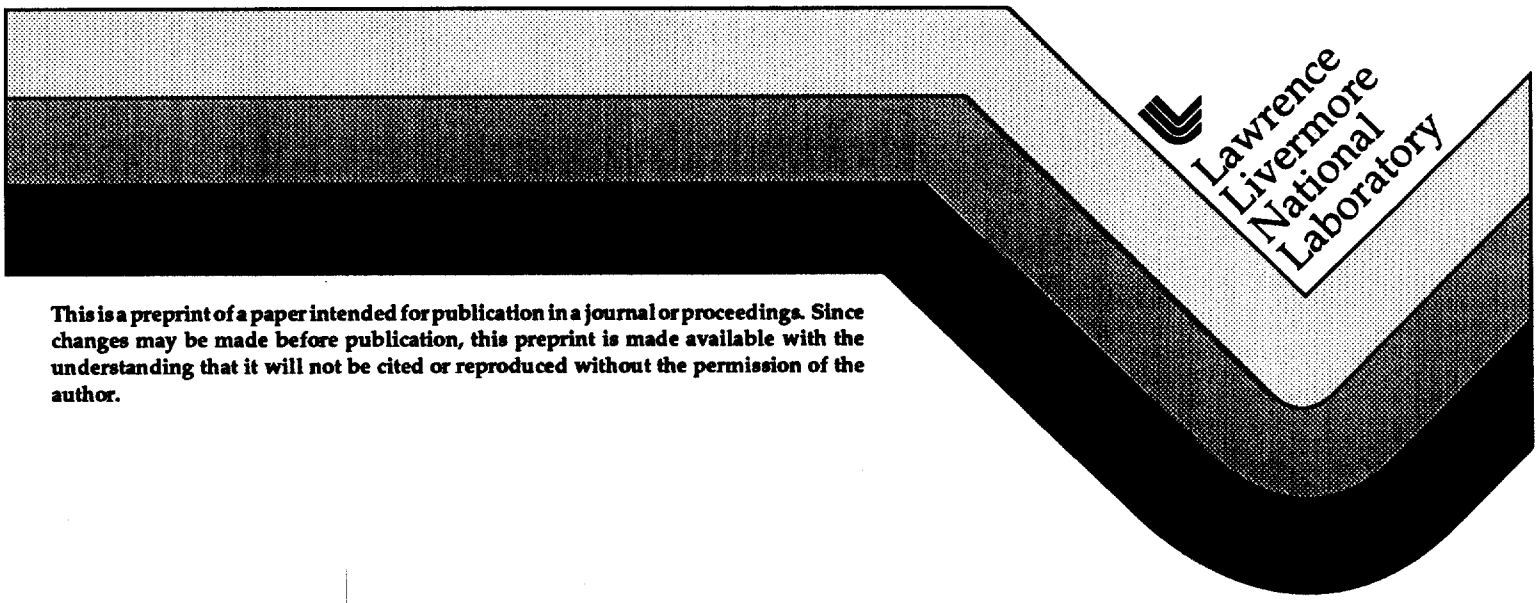
UCRL-JC-127501
PREPRINT

A Production Code Control System for Hydrodynamics Simulations

D. M. Slone

This paper was prepared for submittal to
The O'Reilly Perl Conference
San Jose, CA
August 19-21, 1997

August 18, 1997



A Production Code Control System for Hydrodynamics Simulations

Dale M Slone[1]

Introduction

We describe how the Production Code Control System (PCCS), written in Perl, has been used to control and monitor the execution of a large hydrodynamics simulation code in a production environment.We have been able to integrate new, disparate, and often independent, applications into the PCCS framework without the need to modify any of our existing application codes. Both users and code developers see a consistent interface to the simulation code and associated applications regardless of the physical platform, whether an MPP, SMP, server, or desktop workstation. We will also describe our use of Perl to develop a configuration management system for the simulation code, as well as a code usage database and report generator.

We used Perl to write a backplane that allows us plug in preprocessors, the hydrocde, postprocessors, visualization tools, persistent storage requests, and other codes. We need only teach PCCS a minimal amount about any new tool or code to essentially plug it in and make it usable to the hydrocode. PCCS has made it easier to link together disparate codes, since using Perl has removed the need to learn the idiosyncrasies of system or RPC programming. The text handling in Perl makes it easy to teach PCCS about new codes, or changes to existing codes.

Hydrodynamics

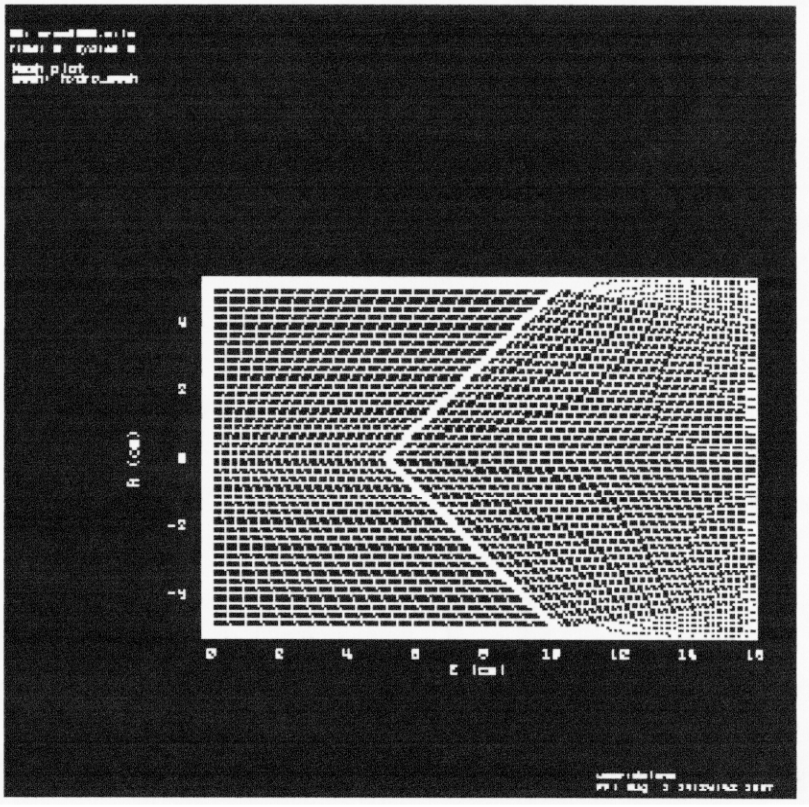
In a basic description, hydrodynamics codes compute material properties such as velocity, density, pressure, or temperature as a function of time and space in response to external forces. Such codes first create a grid or mesh that approximates the geometry of interest. The grid is initialized at either the grid points or cells (a cell is created

---

1. Computer Scientist, Scientific Computing Applications Division, Lawrence Livermore National Laboratory, Livermore, CA, 94551. slone3@llnl.gov

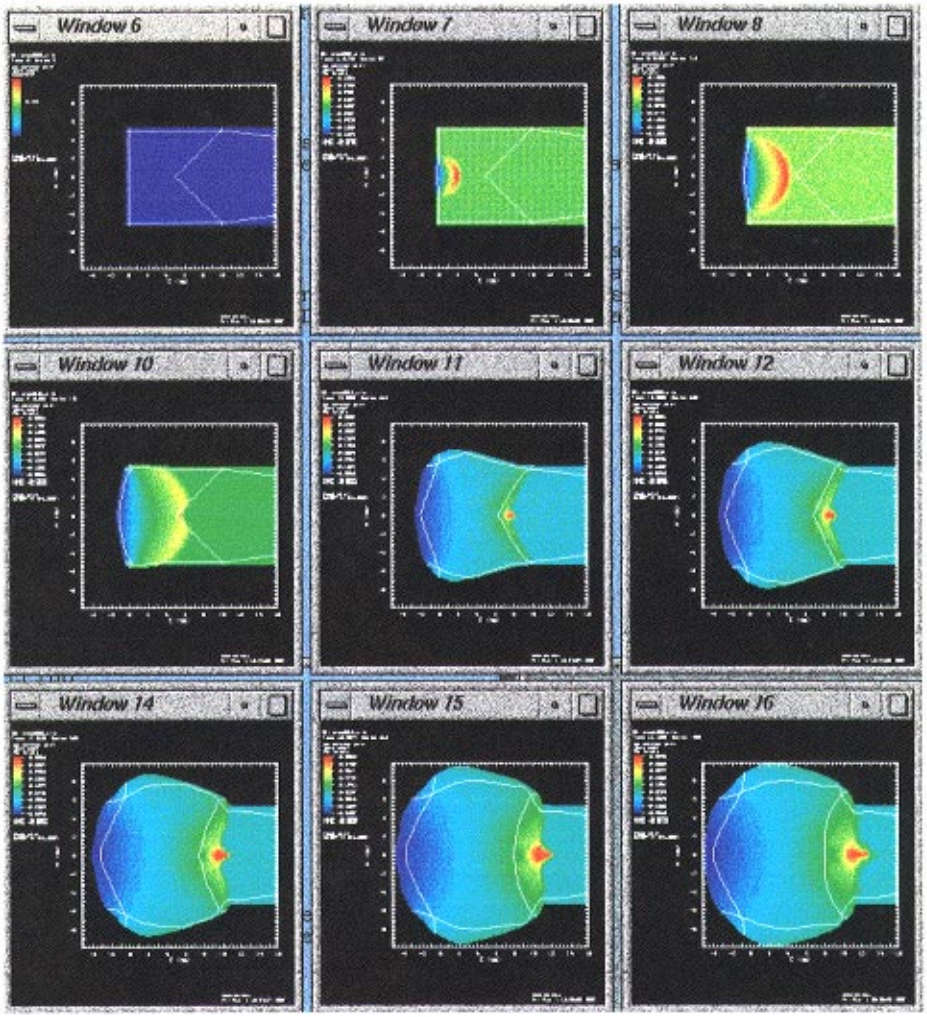
This work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under the contract number W-7405-Eng-48

by joining adjacent grid points) with the appropriate material properties. The figure below show a typical grid. At



each step of the simulation, approximations to the differential equations that describe conservation of mass, momentum, and energy are solved. These solutions allow the matcrial to rcspond to the external forces. The only other piece of the system is something that describes the change in density and internal energy (or volume and temperature) with pressure. This is known as an equation of state. We know from experience with water in its other two main forms (ice9 doesn't count,) that material properties can change quite dramatically over even a small temperature range.

As an example of problems that we might run with a hydrocode, the figure below show a series of snapshots from a shock jet simulation. The pictures are a slice through a cylinder that contains explosives (to the right of the rotated V in the upper left) and metal (to the left of the explosives). The colors represent the speed of the material, with blue being slowest and red fastest). Time proceeds left to right and top to bottom.



A legacy code

The code has its origins more than twenty-five years ago as a magnetohydrodynamics (MHD) code to study stars. We are currently on the tenth major release, fifteenth minor release. There are no extant lines of code dating back to that first version. However, there are some from the last major release in the mid-80s. We expect to have the major release around the beginning of the new year. Presently, there are four physicists and two computer scientists working full-time on the code. In addition, there are three physicists and two computer scientists who work part-time on the code.

There are two parts to the code: a generator to generate the mesh and allocate memory for the material properties, and a main code to perform the physics. The code is written in standard f77 with Cray-style pointers. Only within the last 5-7 years has the code been ported to run on non-Crays.The code consists of ~530K loc of Fortran. The generator and main code are ~14Mbytes and ~27Mbytes respectively. There is no graphics capability within the code.

Platforms

There is one Cray-YMP still running, but it's slated to die in September (but it has been more than two years since the first time the plug was going to be pulled.) We have three Cray J90s, one Meiko CS-2 MPP, one IBM SP2 SMP, and one DEC Alphaserver. These machines are managed by the Livermore Computing center and all run

some flavor of Unix. In addition, B division has a few SGI R10000 workstations that can also run the code. The platform situation is quite dynamic. The CS2 is less than four years old, and except for the YMP, it is the oldest of the machines at the center. The DEC is not quite a year old; the center is in the process of installing a cluster of five more DECs. The SP2 is also less than a year old. It is part of the ASCI program that is building teraflop computers. Each new platform requires a port of the entire code and all of the libraries, which is getting easier now that we've done a few.

We use a Perl tool called cave to handle all of the configuration management issues associated with a long-lived code with many developers. Cave sits on top of RCS and allows developers to keep separate work spaces and then merge their resulting changes. We are in the process of adding regression testing to cave.

PCCS

An individual run can last as long as two weeks on a Cray-YMP - continuously! The machines are tuned to provide the best system response for just such long-lived jobs. A batch queue system is used to keep the machines running near their tuned optimization. In addition, there are interactive cpu limits (as little as ten minutes) to prevent users and developers from degrading performance with interactive jobs that could be better run on local workstations. Unfortunately, a problem that runs two weeks may need to notify the user of something important. The Production Code Control System (PCCS) was written in Perl to facilitate running long batch jobs.

PCCS works by scanning a user's input deck for syntactical errors, stopping the run and pointing out the error before the job is started in the batch queue (this is very important when users are charged by cpu cycles.) PCCS then fires off the job in the batch queue using forks to be able to communicate with the running job. Messages from the code are captured by PCCS and relayed to the user if necessary. The users may talk to the code at any time to get status information. PCCS understands the messages given by the code and reacts as required (or as the user would in an interactive session.) PCCS will start a graphics session if desired, or link a mesh created elsewhere with the code, or
switch running modes for the hydrocde itself.

PCCS has been instrumental in allowing long-lived problems to run without interference from the user even as the operating system gets bogged down or even hiccup. Because the hydrocode is so big, and users want to run as large a problem as possible, we can quickly fill up disk space (on some machines, each user has only a small allowed partition of memory and/or disk space to work with.) A typical dump, which contains all the information needed to restart the problem, might be 25Mbytes large. PCCS watches the user's space quota and moves intermediate restart files to a tape storage system behind the user's back. The user can specify how many files to keep on disk, but most let PCCS deal with the whole issue. In addition. PCCS can force the hydrocode to make a restart file if it catches a signal from the system that tells users to save their running job if possible. PCCS has allowed many users to have a running job come Monday morning when the machine had to be restarted sometime over the weekend.

PCCS also is used to run parallel jobs on the MPPs without the user having to understand the subtleties involved without getting a job running on the different parallel schedulers.

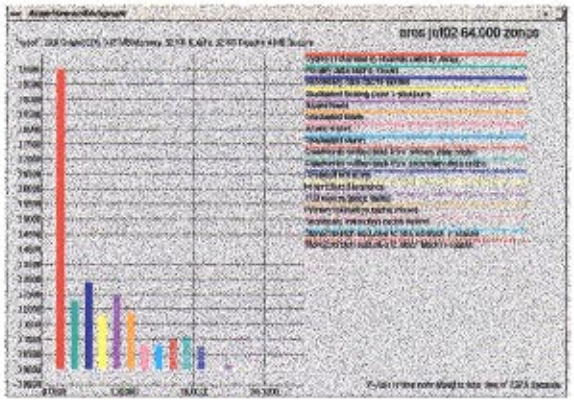
Adding capability

One of the first capabilities that we added to PCCS was an interface to a portable visualization tool call MeshTV. Before MeshTV, visualization was performed on the Crays with a tool that was designed for this code to run only on the Crays and display on only one type of terminal in the user's office. With the advent of powerful workstations and X-terminals, a new tool was developed that used a common data format, rather than one specifically for our code. We changed the hydrocode to use the common data format for graphics files but then added routines in PCCS to talk to MeshTV in the same manner as hydrocode. We now control the visualization from within PCCS; users can execute different sets of graphics commands at different times during the simulation by specifying to PCCS what is to be done when. PCCS uses the current cycle or time to decide what graphics to be performed.
There are other graphics codes that PCCS can run by executing postprocessors and starting up the other graphics codes. This too is specified by the user in the input deck. As new graphics capabilities have been developed, we have needed only to teach PCCS about them, the hydrocode hasn't need to be changed since we started writing in a

common data format.

Within the last few years, another group at Livermore developed an optimizer that runs a particular code or set of codes until some figure-of-merit is maximized. This code, GLO, was not written with hydrocodes in mind, but we used Perl to develop a driver to let users determine which variables should be considered relevant to the figure of merit. We then used PCCS to run GLO with the hydrocode. Neither the hydrocode nor the optimizer had to be modified to run the combination.

We have recently begun to analyze our code's performance using system monitoring software. The picture below is the result of a run using the SGI R10000 perfex tool that monitor various hardware conditions (*i.e.* cache miss, floating point instruction, etc.) We used Perl to grab the output from perfex and plot the results. In addition we are keeping statistics of all monitored runs to see what we can learn about the production performance of the code.



We also use Perl to maintain code usage statistics not only for the hydrocode, but on all codes within the division. We either use system accounting tools (on the Crays,) or monitor jobs using ps to gather cpu use and memory sizes. We can generate textual or graphical reports on either a code or a user basis. The reports help bolster our case when we ask for money to buy more memory or more cpus, or decide whether a particular code needs to be ported to new machines.
We are in the process of webifying the statistics gathering and analysis so that more people will be able to make use of them.

What's next

We are now developing a new hydrocode called ares (see below for the initial configuration of the demonstration problem,) written in C. We hope to use Perl as the backbone of ares itself, rather than the backplane that PCCS is. We would then be able to perform true code steering, where users could not only interact with the code to get status information, but could dynamically interact to change values in the code or redefine abnormal conditions on the fly.

Summary

As our legacy hydrocodes were ported to Unix and off Crays, and specialized tools couldn't be developed any more, Perl provided a means to develop a system that allowed users to run long-lived, large jobs in a production environment. The string handling ability has proved instrumental as we have added capability to PCCS to run codes developed elsewhere and for other purposes. We need teach PCCS only a small amount about the new codes and fork them off, allowing users to interact with the codes.

We hope that Perl will allow us to create a more integrated code environment, where Perl would be intimate with the hydrocode as associated tools. Users would then be able to dynamically control running jobs, providing immediate feedback to their problems.