Science & Technology Facilities Council

Technical Report

Computational Steering Toolkits for Grid Environments

J.V. Ashby

11 Jan 08

RAL-TR-2007-018

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44 (0)1235 44 6403
Email: Library@rl.ac.uk

ISSN 1358-6254

Computational Steering Toolkits for Grid Environments

J.V. Ashby

Computational Science and Engineering Department
STFC Rutherford Appleton Laboratory
J.V.Ashby@rl.ac.uk

Abstract

Computational steering is a netural extension of interactive computing in which the parameters of a program may be altered as the program is running. We discuss two software libraries for Computational Steering, gViz and RealityGrid, which are targeted at Grid Environments. The architectures and functionality of these two libraries are outlined. We also present a discussion of the desirable features an application should have to make use of these libraries to achieve the desired goal of more and better scientific understanding.

1. What is Computational Steering?

The basic notion of Computational Steering is, to a large extent, inherent in interactive computing. With Computational Steering, one alters the parameters of a program to achieve some desired solution or output state, *while the program is running*. At one level, a command line interface to an operating system is "steering" the OS to perform particular tasks and hence to produce certain outputs. At a much more intimately bound level, a flight simulator is steered to respond in near real time by altering the controls of the simulated aeroplane, whether these are a physical reproduction of the real controls, keyboard shortcuts or mouse events.

For the purposes of this report we shall consider a more restrictive view of Computational Steering. We shall consider the case where the state of the program is constantly (or regularly) monitored, usually with some visualisation. The command line interface only presents the system state when prompted and so fails this test. A graphical representation of the filestore, however, comes closer since there is immediate visual feedback when changes are made such as deleting or renaming a file. The OS remains responsive, however. There is no sense in which left to its own devices the computer would evolve its state in one direction, but intervention from a user can redirect that evolution. This is the essence of computational steering in the sense we wish to discuss.

We also rule out the flight simulator since for the most part such applications are tightly coupled to their visualisation and interaction loops. Of more interest to us here is the case of a scientific application code, perhaps developed for batch running, where it can make sense to intervene during a program run to change the program's trajectory. For example, a program might have a parameter, α, whose value for best performance or scientific outcome may not be known, and which may be problem specific. By monitoring the solution process it may be possible to capture poor behaviour such as divergence of iterative methods or the generation of unphysical regions, and change α, steering the program to the desired state.

Wright [1] distinguishes between two different types of Computational Steering. A computer program can be thought of as a map between parameter space and a solution space. Usually one thinks of this as a mapping of a single point in the parameter space (the set of inputs) to a single point in solution space, the output of the completed program. However, in reaching its final state the program follows a trajectory in the solution space, and by interrogating intermediate results this trajectory can be visualised. In Type 1 steering we are interested mainly in exploring the parameter space, possibly with a desired solution in mind. By monitoring the trajectory we can, for example, prematurely abort simulations with parameters which are leading away from the desideratum. Alternatively we could spot interesting regions of parameter space and revise the exploration strategy accordingly. Type 1 steering is thus akin to process farming, but with the added advantage of being able to interactively intervene in processes which appear to be exploring parts of the solution space which is uninteresting. In Type 2 steering, on the other hand, the intervention is more immediate and one can change parameters "on-the-fly" to control the solution towards a desired (previously determined) goal. Type 2 steering is looking for the set of input parameters which map to a particular solution..

Both Type 1 and Type 2 forms of Computational Steering rely on human interaction with and analysis of the data. Computational Steering is thus closely connected with visualisation and data presentation systems. It is usually to be found embedded within a Problem Solving Environment. For

more information on general Computational Steering systems, see [2,3].

In the next section we shall briefly discuss the capabilities of some available Computational Steering systems. In section 3 we consider what features of a scientific application make the use of computational steering worthwhile and finally we present some conclusions.

2. Available Software

In 1998 Mulder, van Wijk and van Liere [4] surveyed Computational Steering Environments, and in 2001 Allan and Ashworth [5] surveyed tools for high performance computing, including computational steering systems. Between them they identified nine systems of which one, VASE, was already no longer under development in 2001. Of the remaining eight one, COVISE, is available commercially and is being developed actively, with a focus on Virtual Reality as a visualisation technique, another, SciRun, is available as open source and has a most recent release in February 2007 while the remaining six appear to have become defunct. Both COVISE and SciRun based themselves originally on AVS, although they have both branched out to interface to other visualisation systems such as MATLAB. There is clearly a high historical attrition rate amongst Computational Steering tools.

There are two recent projects which were unmentioned in either [4] or [5], gViz and RealityGrid. Both of these embed their Problem Solving Environments in a computational Grid and use their computational steering libraries as part of the middleware for the Grid.

gViz

The gViz Project [6] was a collaboration between the Universities of Leeds, Oxford Brookes and Oxford, CCLRC, NAG Ltd, IBM UK and Streamline Computing. It was funded under the UK eScience Initiative to develop the middleware needed to support visualisation in a Grid environment. The major output was thus the gViz library, gViz Lib, [7] which allows application developers which can be executed on remote resources but controlled and interacted with by user interface (UI) components running on a local desktop. The level of abstraction encapsulated in this library is such that it can be used as a more general model of communication between remote and local processes. It has been designed to have minimal impact in terms of changes to a simulation code and on performance. It is also independent of any particular visualisation system, and it has been used with both IRIS Explorer and MATLAB.

The library is in two parts: gVizLibC for routines appropriate to simulation and gVizLibP for routines called from the User Interface. The simulation must initialise the gViz library at start-up; this spawns a thread to listen for the UI and initialises internal datastructures. It also registers a connection with an external agent and passes information about steerable and viewable parameters, the services it provides. The UI retrieves this information and uses it to establish a connection. Routines are provided to establish the connection and handle requests for various services from both UI and simulation sides.

Parameters for the simulation are divided into two classes, steerable and viewable. Steerable parameters are a subset of viewable parameters which can be changed in a simulation by an external process. These parameters can be any tunable parameters in the code – they may be physical boundary conditions, the “constants” of various physical models or purely numerical and algorithmic

quantities. Examples include wind speed and direction in the simulation of a pollution incident, the diffusivity of the pollutant in the atmosphere and the size of the residual to use as a stopping criterion in an iterative linear algebra routine. The simulation process determines which ofits parameters are steerable, but the UI uses this information to build a graphical interface to the program, showing the current state and providing interactors. The set of steerable parameters is fixed at the time of connection. If the simulation decides to change which parameters are steerable, perhaps as the result of a change of state which makethis sensible, any steering processes already connected are not updated.

Viewable parameters also include parameters within the simulation which can be viewed by the UI (and hence by the user) but which cannot be altered. They are Read Only as opposed to the Read-Write of the steerable parameters. Viewable parameters can be updated at any time during the simulation, not just at the end of a time step or iteration, or when the steerable parameters are updated. It is expected that all steerable parameters will also be viewable. The UI is free to drop intermediate changes to a viewable parameter if changes are being made faster than it can respond.

The simulation can also pass data to connected processes, either as data objects in their own right or by reference to data files which can then be accessed by other retrieval tools such as GridFTP.

Because of the Grid environment in which gViz is expected to work, some tools are provided to assist in managing simulations which are run in batch queues and/or on cluster nodes not immediately visible to the outside world. GvizDS is a simple directory service which provides a connection point for simulations which register their details with the service, UIs which interrogate the service to find out what and how to connectto, and proxies which can use the service to verify details of a process to which they are being asked to connect. It would normally run on the head node or a cluster, visible to both the cluster and the external network. gVizProxy also runs on such a node and mediates the transfer of information between simulation and UI.

The API for gViz is written in C and currently only a C binding is available.

RealityGrid

RealityGrid [8] was also a project funded by the UK eScience Initiative and involved University College, London, the Universities of Edinburgh. Manchester, Loughborough and Oxford and Imperial College as Academic Partners in collaboration with Computation for Science Consortium, Schlumberger Cambridge Research Ltd, Edward Jenner Institute for Vaccine Research, Silicon Graphics Inc., Advanced Visual Systems Inc., Fujitsu Ltd and BTexact. The project also drew on resources in US universities and national laboratories as well as supercomputing centres. The project's aim was to produce a framework in which experimental facilities, databases, computational resources and visualisation can be coupled in a transparent, high-throughput fashion to produce scientific results that would not otherwise have been possible.

The Computational Steering component, the RealityGrid Steering Grid Service (SGS) [9], was one middleware component among several which addressed different issues of job scheduling, resource discovery, and application coupling. It is written in Perl and uses the OGSI::Lite hosting Environment. The architecture is very similar to that used by gViz: there is a Registry to which the simulation supplies details of what parameters can be steered or what commands it accepts. Standard

Grid Services methods are then used by the UI to discover this data. Before launching the UI, the launching script queries the registry for the Grid Service Handle of the simulation SGS, then queries the SGS itself for a list of IOTypes. This is returned as an XML document which the UI can then parse and present to the user in whatever way is deemed appropriate. An SGS associated with the UI client is created (if necessary) and initialised. The addresses of the two Steering Grid Services are passed to their respective programs by means of an environment variable.

On reaching a call to Register_IOTypes, the UI queries its SGS and uses the IOType list to populate its datastructures. This list contains not only the name of the parameters available, but also their Grid Service Handles from which the UI can discover how to connect to it.

Once connected the UI and simulation communicate through a Steering Protocol which supports various methods. These include Attach, Detach, GetNotifications, GetNthDataSource, GetStatus, Pause, PutControl, Resume and Stop. Many of the communications initiated by these methods are mediated by XML, according to a specific schema.

3. What makes Computational Steering worthwhile?

Computational Steering is a powerful tool, but like all such tools it must be used with respect. We have seen in section 1 that there are two basic modes of operation for steering, and here we discuss their advantages and pitfalls. There are some commonalities as well. The clearest is that for either form of steering to be applicable to a simulation, the timescales involved must be appropriate. There are four such timescales: the time for a simulation to alter its state sufficiently to warrant an update of its output, normally a few timesteps or iterations in an iterative process; the communication time needed to pass data between simulation and UI; the rendering time in which the UI processes the raw data coming from the simulation and presents it to the user; and by no means the least important the time taken for the user to make any decision based on the presentation of the data. If the simulation time is very short in comparison with any of the others, the simulation will have moved on to a very different state by the time any user intervention is received. If, on the other hand, it is very long, the element of interactivity is lost – steering may still be appropriate, but will be more a case of checking in every hour or so to see how the simulation is progressing. In some cases it may be better to perform a full parameter sweep, save all the data one possibly can and use off-line datamining techniques to explore it. However, in some cases the quantity of data will preclude this. Again a judgement has to be made on the most appropriate strategy to follow. Note that this may change over time as storage or compute costs alter.

When Computational Steering is being used in Type 1 mode, for sweeps of the parameter space we should consider firstly whether abandoning apparently uninteresting simulations is the best thing to do, and if so whether steering is the only (or best) way to achieve it. By selecting out "uninteresting" solutions we may be biassing the simulation results towards the expected, and missing genuinely interesting results because they do not fit our pre-conceived notions. This is, of course, a perpetual problem in interpreting the results of simulations, but as with all technology which makes things apparently easier, steering has the potential to make us forget. In those cases where the decision is clear about aborting a simulation it may be that it can be made automatically by, for example, a simple algorithm examining the results as they are created. The generation of non-physical results or numerical divergence are possible scenarios where this can apply, but these may not require human

intervention. There are cases where an algorithm might be too expensive to apply and human judgement more applicable, and here steering may be of use.

In Type 2 steering the same problems of bias apply as one alters parameters to look for the desired solution. In time dependent simulations steering can be useful to investigate the impact of specific interventions, particularly if the steerable parameters relate directly to controllable physical parameters. For example, in the case of a pollution release while it is interesting to steer the wind speed and direction to investigate various "what if?" scenarios, it may be of more consequence to be able to control the aperture through which the pollutant is released to see whether a fast release or a slow release is preferable, and at what point in the simulation any such changes should be made. With such experiments a mechanism to backtrack to a point in the simulation history and change how the parameters were altered is highly desirable.

It cannot be too strongly stressed that the results of a steered simulation are only as good as those of the unsteered simulation, and that well-validated programs must be a priority. Having said that, steering can assist in the validation process, particularly in assisting the developer to define the regions of parameter space over which the simulation can be trusted. Once this is done, then a steered simulation can be handed over to less expert hands for use. This is particularly important as simulations start to be used in clinical contexts to assess the efficacy of surgical or other interventions, and in similarly important areas.

4. Conclusions

We have examined the potential of Computational Steering for controlling scientific applications, both for simple desktops and on High Performancesystems. Two pieces of middleware which assist Computational Steering in a Grid-based system have been described. Some of the pitfalls of Computational Steering have been discussed. In spite of these, steering is a relevant and potentially powerful tool, providing it is used wisely and judiciously. In that case it is capable of enabling the discovery of good science more speedily than would be possible by traditional methods, and of generating improved understanding, both of the simulated phenomena and of the simulation process itself.

References

1. Wright H., Introductory remarks to the CompuSteer Workshop, Hull (unpublished) (2007)
2. Kreylos O., Computational Steering of CFD Simulations, http://graphics.cs.ucdavis.edu/~okreylos/ResDev/CFDSteering/index.html
3. Avis N., Computational Steering, www.wesc.ac.uk/resources/presentations/pdf/vc-lecture-feb04.pdf
4. J. Mulder, J. van Wijk, and R. van Liere. A Survey of Computational Steering Environments. Future Generation Computer Systems, 13(6), (1998) http://citeseer.ist.psu.edu/article/mulder98survey.html
5. Allan R.J. And Ashworth M., A Survey of Distributed Computing, Computational Grid, Meta-computing and Network Information Tools, UKHEC Technical Report (2001)

http://www.ukhec.ac.uk/publications/reports/survey.pdf

6. Brodlie K. and Wood J., gViz: Visualization and Computational Steering on the Grid, All Hands Meeting 2004 http://www.comp.leeds.ac.uk/vvr/gViz/publications/AHM04_wshop_paper.pdf
7. gViz Lib, http://www.comp.leeds.ac.uk/vvr/gviz/gViz1.1.tgz
8. RealityGrid: moving the bottleneck out of the hardware and back into the human mind http://www.realitygrid.org/
9. The RealityGrid project http://www.rcs.manchester.ac.uk/research/realitygrid