Chapter 12

Solving Computationally Intensive Engineering Problems on the Grid Using Problem Solving Environments

12.1 Introduction

The present rapidly changing state of Grid computing is that the technologies, resources, and applications used in grid computing all have tremendous diversity [1]. In particular the applications are diverse and encompass many different computing techniques. The common theme between them all is that computing power away from local resources is required, and that these applications envisage the need to use distributed resources. With any application the results achieved should be the most important part, and Grid technologies should be employed to facilitate getting faster results to harder problems.

Grid applications which are computationally intensive and collaborative in terms of the scientific community examining the results lead to two important questions [2]. First, how can knowledge and insight be acquired quickly from an application that may be running on a distributed resource rather than on the users' desktop machine. The second, how can these results be effectively shared between potentially geographically disparate scientists who may well have very different areas of expertise. Both of these questions are particularly relevant when the Grid application is being run by non traditional users who may not have computational science backgrounds, and who may be from a broad range of disciplines.

While a commonly held belief in the Grid community is that the Web Portal will allow applications to be run from any computer, it may also be true that the Web need not be the only portal into the Grid. The Web interface may be simple, but the outputs returned are constrained by the same "bare bones" nature of the environment. Users from all backgrounds are now used to high quality 3-D visualizations, from graphics on news programmes to high quality animations shown during talks. The Grid has the ability to provide these same quality outputs, but the interaction method may need to be driven by the demands of visualization software which provides an important way of extracting knowledge from large datasets. In this work we describe the stages necessary to provide real-time desktop visualization of results for a parallel Grid application computed via problem-solving environments (PSEs). Apart from the visualization, the biggest advantage of using a PSE is the ability to use computational steering. In traditional applications, the input parameters are set and not often changed while the code is running. By making the input parameters available in the PSE, it is possible to change the problem being solved after the

simulation has been started. This means that the visual feedback from the output visualizations can help to guide further inputs, either away from failing computations or toward better investigation of more exciting parameter regions. The benefits of computational steering as a means of improving the solution procedure go back to one of the first demonstrations of steering by Haber and McNabb [3] in 1989.

Recent work by Brodlie et al. [4] shows how collaboration between scientists using PSEs can be achieved through various methods such as (i) sharing the display of the PSE, (ii) sharing the output visualisations once they have reached the PSE, and finally (iii) sharing the access to the simulation allowing steering from any user. The first of these can be done using technologies such as Virtual Network Computing,[1] (VNC) which allows any internet connected computer to view and interact with the desktop on a single machine. This can be useful for shared demonstrations such as in Access Grid Ref here sessions, where conversation may revolve around the computed results. Sharing of data is possible through technologies such as COVISA [5], which has a client-server approach enabling sharing of parameters, formatted datasets, or even geometry within IRIS Explorer. In this work we focus on the third type of collaboration in the final section of the chapter.

Although the example used here is computationally demanding and lends itself to parallel solution techniques, the approach taken here will be to illustrate, through the code development, the use of distributed and shared memory parallelization techniques as will the use of Grid job management. Using the Grid as HPC on-demand for large, complex applications will inevitably involve the use of parallel programming techniques within the application [6]. The Grid will be presented in terms of getting seamless, interactive, on-demand access to high-performance computing (HPC) resources, and how applications may be extended beyond conventional HPC considerations.

The rest of the chapter describes the series of stages necessary to transform a typical application into a fully Grid-enabled one, operating from within a problem-solving environment. These stages will be further explained with a series of example PSEs outlining the technical enhancements necessary for transforming an engineering code into one which fully exploits the benefits of Grid technology.

In Section 12.2, the basic components of a PSE are described in detail, along with some consideration of the options available in choosing the package within which the PSE is to be built. The case studies focus on IRIS Explorer and SCIRun, although other options exist including the use of packages such as Cactus,[2] MATLAB,[3] AVS,[4] and OpenDX.[5] These are a mixture of open source and proprietary software, but the principles for developing in one system transfer well into the others. Other example PSEs using these systems can be found in work by Brodlie et al. [7] for IRIS Explorer; Johnson et al. [8] for SCIRun; Allen et al. [9] for Cactus; Kierzenka

[1] `http://www.realvnc.com` or `http://www.tightvnc.com`

[2] `http://www.cactuscode.org`

[3] `http://www.mathworks.com/products/matlab`

[4] `http://help.avs.com/AVS5`

[5] `http://www.opendx.org`

and Shampine [10] for MATLAB; and, Treinish [11] for OpenDX. More tailor-made solutions include the ICENI project [12], and the RealityGrid project using VTK[6] [13].

In Section 12.2.1, we describe how a traditional code for a demanding mechanical engineering problem has been embedded for use within a PSE. In this study we also consider some of the issues to do with like, how embedded simulations are managed. This example is not necessarily Grid-aware, but sets up the necessary framework for the later applications discussed.

The idea of running the PSE on the Grid and doing the rendering locally, as described in the second example, does not optimise the resources effectively. The greatest leap in Grid computing for PSEs comes when the local machine running the PSE, authenticates with a Grid resource and handles the communication with this separate resource. We describe the mechanisms for doing this in Section 12.4 and in the accompanying PSE Example. Particularly relevant here are the measures for communicating input and output data between the desktop PSE and Grid simulation.

The final stage in the evolution of the Grid-enabled PSE is to remove the dependencies between the desktop and the Grid processes. This is done through the launched process having an extra library attached which handles all the communication with the simulation. This means that once launched, the Grid process need not have any "listeners" but equally it may have many who will be able to see the same results and steer the application. These extensions are described in Section 12.5. The final example describes how such a simulation may be set up, and why such an application is ideally suited for a Grid environment beyond traditional HPC needs.

We conclude in Section 12.6 and consider some of the features which still need implementing. The most obvious issue which we are not attempting to cover in this work is that of brokering. Intelligent brokering, the automatic choice of the best Grid resource for the particular application is still some way off, although many test projects are considering these issues. We have only considered the case that the PSE user can use the standard tools to make the choice of resources, coupled with personal knowledge of the resources required for the application in question.

12.2 Problem-solving Environments (PSE)

One aspect of the advent of Grid computing is that users are now able to access much more powerful resources to perform larger simulations. Since the number of people performing such calculations is not decreasing, then the contention for these resources is also increasing. If the Grid moves to a "commodity computing" phase where each CPU second is chargeable then effective management of simulations will also become an economic factor.

Problem-Solving Environments (PSEs) combine several important processes into one body. The actual application, be it a numerical solver, as used in this chapter, or data–Grid style search, e.g. [14], is only one component in an environment, which has access to visualization tools for the output results generated. It also has the ability to set input parameters for the application, and hence can provide a user-friendly interface to a complex problem. The PSE therefore has synchronous computation and visualization. There are three ways in which even basic PSEs are advantageous: the input parameters can all be set, or adjusted at run time; the solver is included as an important part of the PSE and hence it can be possible to change solution methods, if appropriate; and finally the visualization is an innate component of the package, and results can be visualized and studied as the calculation proceeds. Computational steering gives the PSE

[6] `http://public.kitware.com/VTK`

another advantage over traditional solution methods because this allows the test problem and/or the solution methods to be updated during the calculation. The user, thus, "closes the loop" of the interactive visually-driven solution procedure [15].

The choice of visualization techniques to be used will obviously depend on the data being generated by the application. The number of dimensions of any solution dataset will guide the user toward different techniques. For example one-dimensional results are easily visualized on a graph, but two-dimensional cases allow use of techniques such as contouring or projection into a third dimension. Once three-dimensional cases are considered isosurfacing, slicing, and volume rendering are all standard techniques. Whichever technique is chosen, the visualization system is always more useful with sensible colouring schemes and the ability to rotate the rendered geometry in three-dimensional space. The PSEs should have the ability for the experienced visualization engineer to construct detailed, informative representations of the data generated. For the PSE user it is often the case that they have no desire to learn how to use the intricacies of the chosen environment, but simply to use it as an experimentation tool, pushing buttons and dragging widgets within predefined ranges.

In the introduction, various frameworks for building PSEs were mentioned. In this chapter the examples concentrate on two of these, namely IRIS Explorer [16] and SCIRun [17]. The former is proprietary software from NAG,[7] whilst the latter is open source, available online. The choice of framework is intended to be supportive rather than prescriptive, and the techniques used in this chapter can be extended to any of the other options. The general appearance of PSEs is often as a workflow diagram, with different tasks connected together. In PSE terminology, elements of workflow are typically referred to as *modules* connected by a *dataflow pipeline*. Whilst not all these tasks may be on the Grid it will be seen how the location of the work should feel independent to the user.

Development of the application user interface should not require changes to the application software, but merely the input and output interfaces. The generation of output data structures for the chosen visualization environment is perhaps the most complicated step, although in our experience, once coded, these mechanisms tend to be very portable between different applications. This, therefore, has the important advantage that the simulation can be developed quite independently of the details relating to the PSE, but allows the PSE user interface to be exploited.

12.2.1 Computational Steering in IRIS Explorer for EHL Problems

Computational steering is the procedure by which the user can interact and guide the simulation once it has started. General information regarding the requirements for computational steering is discussed by Mulder et al. [18], who assess a selection of frameworks in regard to a set of criteria they consider desirable for building PSEs. The key part to doing steering within a Grid setting is communicating information to and from applications which are already running. Other examples of this style of working within a Grid setting are provided in the RealityGrid project [19] and in the gViz project [20]. Described briefly, allowing the user to examine the output results enables decisions to be made about future computations. By having the input parameters as accessible widgets on the screen, the user can alter as many or as few of them as they like between runs. Having the application check back to the user interface regularly means that updated parameters can take effect very quickly, hence enabling steering to facilitate learning. The parameters

[7] `http://www.nag.co.uk/`

controlled need not always be computational inputs, but could even be manual dynamic load balancing, for example.

The numerical problem selected motivating our need for using a PSE is that of *elastohydrodynamic lubrication* (EHL) in, for example, journal bearings or gears. This mechanical engineering problem requires sophisticated numerical techniques to be applied in order to obtain solutions quickly. The history of the field is detailed out in papers such as [21]; much information about the numerical techniques currently used to obtain fast, stable solutions is given in both [22] and [23], the latter of which describes in great detail the precise methods used in the code employed in this work. The numerical code used for solving EHL problems used in this chapter is described in detail in [23], and it is used by Shell Global Solutions industrially.

A typical user of this EHL software would be an engineer wanting to establish solution profiles for a particular lubricant under certain operating conditions. Traditionally this would have involved multiple compilations and simulations, with postprocessing of data. With a PSE the instant visual feedback could be quickly used to tune the parameter sets to give the desired results. At this stage, say, a more demanding transient problem could be tackled. The required changes to transform this stand alone software into a PSE application are set out below.

For this example of PSE the framework chosen was NAG's IRIS Explorer [16] product. There has been earlier work employing IRIS Explorer for the development of PSEs, such as Wright et al. [24]. IRIS Explorer is marketed by NAG as a "advanced visual programming environment" for "developing customized visualization applications."[8] Although the IRIS Explorer workflow diagram shows data travelling between elements along "wires," large scale transfer of data is avoided by passing pointers to structures of known types stored in the shared memory arena.

The standard workflow pattern in IRIS Explorer is, normally, a data set either being read in or generated and then control passes to the next module (or modules) downstream. These in turn execute, provided they have all their required inputs and control passes again. If a required input is missing then the module will wait until it is received before executing. If a multiprocessor-shared memory resource is used, then simultaneous module firings will be done on separate processors. This is because IRIS Explorer starts each module as an entirely separate process in the computer. It will be seen how this has both positive and negative consequences, but most importantly will be shown how this can add to the Grid-enabled nature of the software.

The simulation code has been implemented as one module containing the entirety of the numerical solver. The module's control panel is used to set a selection of engineering and numerical characteristics of the problem to be solved. Furthermore, extra information may be provided to the solver through the use of extra input modules, as shown in Fig. 12.1. This has the effect of allowing the user to build up the model of choice through easy interfaces, rather than being faced with large numbers of inputs over which they have no interest. Once the module has completed execution, the datasets of the calculated output profiles are sent down the map for visualization.

The PSE-enabled version of the software has been developed from the original Fortran code by adding an interface routine written in C. The generation of all the IRIS Explorer data structures and communication is done through the *Application Programming Interface* (API) which is well documented for both C and Fortran. The design of the module's user interface is usually done through the Module Builder which allows the widgets to be positioned through a visual interface, rather than by writing code. The Module Builder will also generate the necessary wrapper

[8] `http://www.nag.co.uk/`

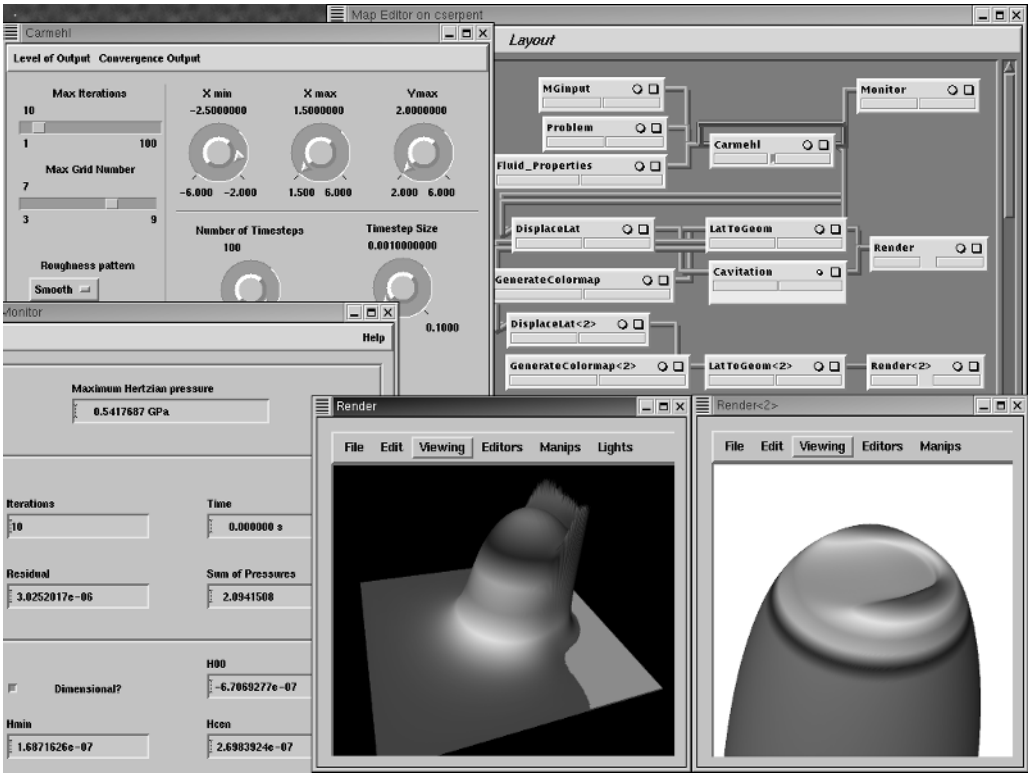


FIGURE 12.1. The PSE running in IRIS Explorer.

codes for complete control of the module's firing pattern and communication of data through the workflow pipeline, and these require no alteration by a developer.

Computational steering is implemented in IRIS Explorer using the looping mechanisms provided. Rather than saving results to disk at the end of a run, the work arrays inside the software can be declared as static and hence the previous results are automatically available for use on the next run. A solution used in this manner may provide a good initial estimate for a differently loaded case, or be interpolated for a change of domain size.

The use of the Hyperscribe module [25] would allow another layer of steering to be included. This module stores datasets or variables on disk for future usage, at the user's discretion. If the entire work arrays, previously saved as static, were stored based on the problem's input characteristics then a suite of previously calculated solutions could be created for future invocations of the PSE on separate occasions, or even by other users.

12.3 Parallel PSE Applications

The style of PSE creation described in the previous section is appropriate for applications which run on a standard PC, where the solution process is sufficiently quick so that the desired steering can produce visible changes instantly, e.g., the pollution demonstrator described in Walkley

et al. [26]. It is, however, the basic building block for the rest of the work described here on constructing Grid-enabled PSEs. Interactivity has been obtained, visualizations rendered, and theoretically trivial parallelism on the PSE level may have occurred since the different simulation and visualization processes should be occurring independently, hence, a multiprocessor machine should enable simultaneous execution. The next stage has to allow the simulation itself to be run in parallel.

Working in the framework of the simulation as an embedded module means that far greater consideration must be given to the actual fabric of the environment in which the PSE is built. For example, as was explained above, IRIS Explorer has each module in the dataflow pipeline as a separate process. Since these processes are launched internally from IRIS Explorer and the processes themselves are wrapped in generated code to communicate with the IRIS Explorer user interface, then launching one of these modules using MPI [27] is not an option currently available. Instead shared memory techniques have been used. In this section the use of shared memory parallelism using SCIRun will be explored.

12.3.1 Parallel Shared Memory Computation within SCIRun

The SCIRun has been developed by the SCI group at the University of Utah as a computational workbench for visual programming [17] and has now been released as open-source software. SCIRun was developed originally for calculations in computational medicine [28], but has since been extended to many other applications.

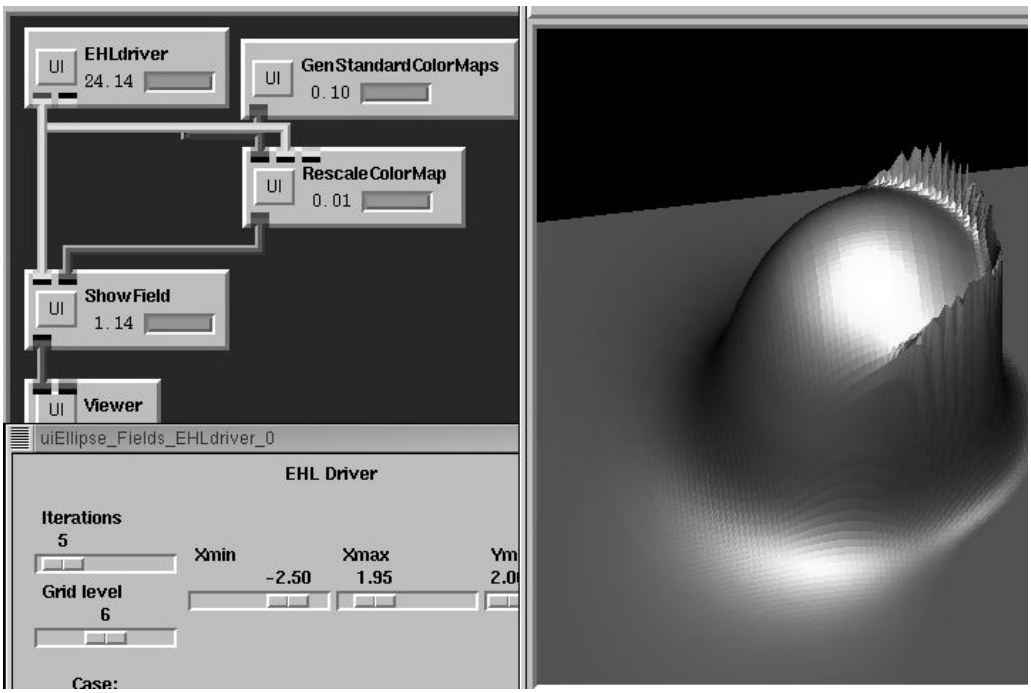


FIGURE 12.2. Parallel threaded EHL PSE running in SCIRun.

The overall appearance of SCIRun is similar to that of IRIS Explorer, as can be seen in Fig. 12.2, where the implementation of the EHL problem explained in Example PSE 1, can be seen working. The module firing algorithm in SCIRun probes the workflow diagram from the desired point of execution so that all elements have all the information they need to run, before then sending the information downstream and firing those modules. This means that upstream elements will be fired if they need to supply information to any other element. Similarly all the downstream modules directly affected by the firing will be made aware that new data will be coming.

A SCIRun is a multi-threaded program, and hence a single process, with (at least) one thread for each launched module. Therefore, every module can have access to all same data without the use of shared memory. This has the advantage that there is more memory available for the generation of datasets to pass between modules, and the disadvantage that any operating system limits on the memory available to a single process apply to the entirety of SCIRun, meaning that calculation and visualization are all included in the same maximum space allocation defined by the system. It also means that any variables declared as static in one invocation of a module will be the same as used in other invocations, since the operating system cannot differentiate between the two.

Parallelism can be easily achieved on SCIRun, thanks to its threaded structure. SCIRun has its own implementation of threads that can be easily incorporated into a user's code. The use of threads means a shared memory machine must be used, but within these constraints the parallel performance for numerical calculations is very good. Next generation packages, such as Uintah [29], use a combination of MPI and threads to achieve massively parallel scientific computations on terascale computing platforms.

Since SCIRun is written as a single threaded process, it has added flexibility with regard to the rewiring of workflow elements during execution. For the EHL problem, when a transient case is run, the output datasets are prepared and released down the pipeline for visualization at the end of each time step. With more than one solution variable being solved for, there is obviously a choice as to what is visualized at anytime. In SCIRun, these changes can be made "on the fly." For example, if the pressure solution was being visualized, then it is possible to change to a surface geometry plot between time steps. This is an important feature since it allows the user to learn and experiment interactively, whilst still making excellent use of the allocated Grid resources.

12.3.2 Shared Memory Parallel Computation Grid Architecture

In developing the shared memory Grid software architecture, it should be noted, however, that running the entire PSE remotely is not always a good idea. The main reason for this is that the final rendering should be done locally whenever possible to allow full utilization of the local graphics hardware. Minimizing the network traffic between simulation and display is another factor which must be considered. For examples, with large datasets for visualization, then doing this work on a Grid resource will be very advantageous. Considering the size of datasets to be transferred over the intervening network between generation, visualization and rendering are very important, as are the connectivity rates. This can be illustrated by the scenario demonstrated in Fig. 12.3. Here we are imagining the computationally intensive part being done on Grid Resource 1, but the visualization is done on Grid Resource 2. Often, all communication with nodes of such resources has to be channelled through a head node. Within the resource the communication will use the fast interconnects, or shared memory, but between resources the

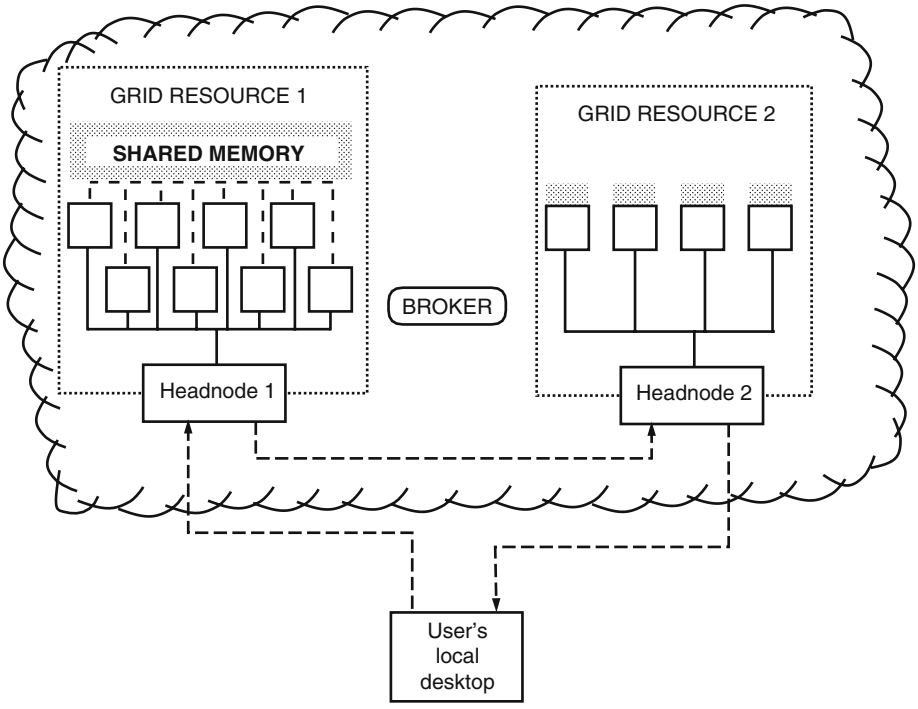


FIGURE 12.3. Example Grid architecture diagram, with different workflow elements on distributed resources.

communication will be at a lower rate. If the native architecture of Grid Resource 1 can be used for the visualization tasks, too, then the overall performance of the application may be substantially enhanced. This scenario emphasizes the difficulties in accurately brokering the entire Grid application, as the broker would need some idea of the data transfer rates between the machines, the size of the datasets to be generated, and an idea of the regularity of the rate at which these are produced.

One way in which manual distribution of work between resources has been implemented in IRIS Explorer is through the use of remote hosts. Work by Wood et al. [20] has extended the standard access methods to include secure Grid-aware authenticated connections.

12.4 Grid-enabled Workflow

To extend the parallelism options available away from shared memory, it must be possible to launch jobs onto remote distributed resources. These resources will now, typically be managed by Grid-aware scheduling software, and hence interaction with this middleware must be done as transparently to the user as possible.

The methods described here use the standard Globus[9] [30] tools for Grid job management, including file input and output. We shall assume that the necessary Globus certification process has already been undertaken before launching the PSE. The brokering is, as described previously, done through user selection based on knowledge of the resources currently available.

The key remaining steps to getting the PSE having elements run on distributed resources are (i) launching the job onto a remote resource, (ii) communicating information back to the PSE detailing the location where the job is running, and (iii) communication of steering and output information to and from this job. These three steps are possible using a variety of approaches, of which we shall describe two.

In order to launch the job onto the resource it is often necessary to have a good understanding of the specification of that resource, and knowledge of the schedulers on that resource. For example, a parallel job launched through Globus requires extra information for the local scheduler to best use the native MPI or shared memory options. Unfortunately, the idea of Globus providing transparent access to heterogeneous resources through a simple command is reliant on the application writer to have enough information about the options to write the user interface and to hide the unnecessary details from the user. A good example of this is shown in Fig. 12.4 below where all the authentication details have been hidden behind the "Use Globus" button.

The communication, the location of the running application from the Grid resource back to the PSE is a surprisingly nontrivial operation. Assuming the job has been submitted using Globus, typically it will be scheduled from the headnode to one or more nodes under its control. This internal node will then be the location with which the PSE will need to communicate information. This location will then need to be passed out of the Grid resource, such that the PSE can obtain the information and commence communication of data. This is typically done either with a direct socket connection, or by using a Web service. The Web service directory approach has the advantage that the location will be available for other users to connect to, or for a central store of running applications with separate input and output streams which the PSE may wish to connect to, and steer, independently. Using a Web service is also useful when the Grid job may be waiting in a queueing system. The advantage of the direct connection is that the connection back to the PSE can be instantaneous on job startup rather than requiring the polling of the Web service. The security of each of these methods is reliant on either the socket connection being encrypted or secure authentication to the web service. The direct socket style of connection is described in Example PSE 3, with the Grid directory style in Section 12.5.

[9] `http://www.globus.org`

FIGURE 12.4. Grid enabling options for an IRIS Explorer PSE.

The final stage is the communication between the PSE and the Grid application. Again it is possible to do this using direct connections or using a Web services approach. The biggest disadvantage of using the latter is that the Web service must be polled to discover if any new output data has been posted, rather than the direct connection method which can wait "listening" for a new dataset to start arriving. It is this latter method which is considered in the rest of the chapter.

One final consideration needs to be given to Grid resources where the internal nodes are not visible to the network outside. In this scenario it is necessary to build a proxy service which manages authenticated connections for services running on that resource. This will run on the headnode, such as shown in Fig. 12.3, and will manage connections for all services on that resource. Such communication bottlenecks are unfortunate, but it is hoped that in future systems, administrators will be able to design more efficient, secure solutions.

12.4.1 Grid-enabled Simulation Using IRIS Explorer

This section describes the expansion of the embedded serial implementation from Section 12.2.1 to a distributed memory, parallel version, as described in Goodyer et al. [31], which is Grid-enabled. To the user, the user interface in IRIS Explorer will still look identical to that shown in Fig. 12.1. The change to the PSE is all implemented within the application module, which is now solely the user interface to the simulation running on the Grid. The additional Grid information required are all confined to only a few extra inputs, such as shown in Fig. 12.4. These are namely the destination, i.e., the selected Grid destination, which we typically have as a user choice from the available resources [32]; the destination executable, to choose which of the available services to run; and, finally the username, which simply acts as an identifier tag in any Web service directory produced.

When the job on the remote machine is started, communication between the launching IRIS Explorer module and the launched Grid process is done through sockets. The launched process knows where to connect to, by means of extra flags passed to it when it is started. Once contact has been established, the launched process then is the dominant communicator, with the launcher as the listener. When the launched process needs data from the PSE, e.g., control parameters for the simulation, it sends a request to the listener who packs the values up into a data array of predefined size and structure, and sends it to the Grid process.

Similarly, output data is packaged by the Grid process and sent to the listener. The received data is then formatted into the relevant output data types which are released down the pipeline for visualization.

Extra input modules are added before the Grid module poses few problems. The incoming data is packed into arrays which are sent to the Grid module, as with the control parameters. Since these input modules need not always be present, then there must be a default set of parameters for cases where they are not connected so that the application can operate accordingly.

Having completed the requested number of job, the Grid process does not terminate but regularly polls the associated PSE module until it is requested to perform the computation. This eliminates the cost of starting up a job on the Grid, and also means that results from the previous iteration can still be stored in local memory or filestore, for future use.

During computationally intensive simulations it used to be the case that the simulation module spent considerable amounts of time "firing." Since the work is being done outside the PSE, this is no longer the case and, so firings only occur when new data is received, be it from changes to input parameters or through the receipt of output data. Whereas input data was only ever available

to the simulation at the start of execution it may now be requested at any time, and hence the opportunity for steering the calculation is increased. This may not always be sensible, so care must be taken when constructing the communication over which parameters can be allowed to change during the solution process.

Part of the rationale for use of the Grid is to gain access to remote-parallel machines. Information about the parallel requirements can be incorporated into the launching mechanism. To accomplish this, two more options must be added to the user interface: one detailing the number of processors and one confirming the launch process for the parallel job. As was explained in the previous section, parallel Grid jobs using Globus do need substantially more information at startup, and sensible communication back to the PSE must only be attempted from one of the processors. The attempt, as ever, in developing PSEs, is to try and hide as much of this as possible away from users.

12.5 Asynchronous Steering of Grid Applications

The method of Grid enabling the PSE described in the previous section with direct socket connections from Grid resource to PSE works well, provided that the simulation you are running is sufficiently fast and that results are available whilst the user watches. The additional demands from Grid-enabled PSEs come when the job has a long initial set up, or when the chosen resource is too busy to schedule the job for immediate execution. This leads to the idea of wanting to launch the job and then having the ability to connect to it later on, potentially allowing asynchronous steering. The functionality required to do this means that the job must be running in a known location, and must be ready to accept new users as well as to continue when said users leave the PSE.

These abilities have been central to the gViz project [20], which is part of the UK e-Science Core Programme. The central theme is that a gViz library is attached to the running simulation. This spawns extra threads for "listening" to incoming communication, and "output" of datasets while the main application generates the results as before. This is intended to act as a much more generic environment for programming Grid-enabled applications.

The PSE modules that attach to the Grid also have the gViz library attached and are able to interact with the simulation by posting and receiving messages via the gViz library. The local-to-remote communication is again done via either sockets or Web services.

Resource discovery and job launching is accomplished using the methods described in Section 12.4. The posting of the socket address or other access information to a Web service provides a central way for any new users to connect to previously running simulations. These options are very useful if there are multiple jobs running or a distributed resource is being used.

Output data from the simulations is expected to be stored centrally with the application. This means that whenever a user joins the simulation they get all the relevant output data as well as the current steering parameters. Since this data is stored in a raw format and the "packaging up" into appropriate formats for output is done at the client end, it has been possible to simultaneously connect a single simulation running on a Grid resource to PSEs in-built using IRIS Explorer, SCIRun, and VTK. The updating of steering parameters, when one side changes a value, can be done transparently and the data is flushed down the map (network) when new data is generated by the application. In this way the simulation and the PSE are now almost disjoint entities but the PSE still retains in full control.

12.5.1 Fully Distributed Grid Applications

In this section we will expand the numerical solver presented earlier to be the full engineering environment used by Shell, as described in [33, 34]. This application is an optimization problem intended to best match experimental results against numerical simulation of lubricant behaviour. This involves typically thousands of evaluations of groups of, between 36 and 100 independent test parameters.

The independent nature of these calculations makes this type of application ideal for parallelism at the solver level. The small amounts of communication necessary between runs make the Grid setting a very appropriate resource. The optimizer itself can be greatly helped by using the PSE to guide the solutions out of local minima and this, in turn, will improve the performance and increase the effective power of the Grid.

The overall schematic of the optimizer is shown in Fig. 12.5. This indicates the distributed nature of the entire application. Whilst only one person needs to authenticate via Globus and start the Grid job, other collaborators may connect direct to the Grid job once they know where it has been launched. The Grid Master process handles all the connections to steering and output information, and is the central point for the distributed application underneath to communicate back to. Each individual instance of the numerical solver can still be running in parallel with communication between instances only between the smaller groups of head nodes. The parallelization of this work is described fully in [34] with just the gViz-enabled PSE described below.

For Grid applications, perhaps the most interesting part comes from the ability to utilize metacomputing techniques. Through the use of MPICH–G2 [35], the Globus-enabled, Grid-aware version of the message passing standard [27], it is possible to exploit the Grid as a collection of smaller resources to accomplish large HPC tasks. The MPICH–G2 passes messages in a manner which reflects the topology of the Grid to maximize efficiency of communication. This means that each instance of the computationally expensive, communication heavy numer-

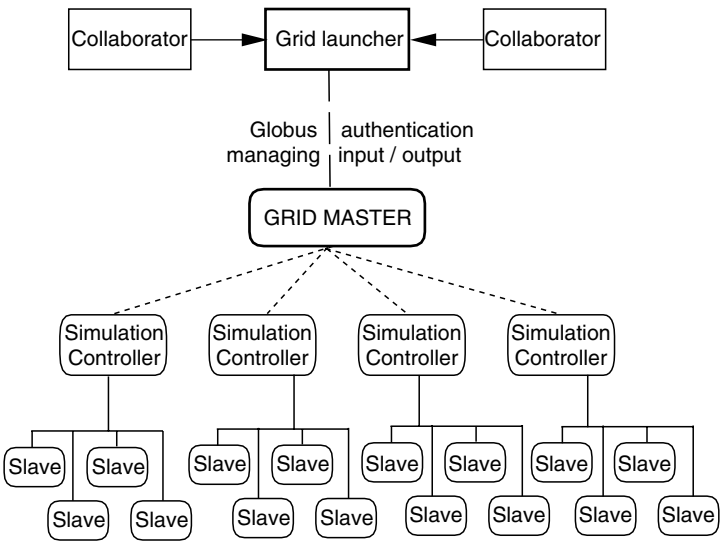


FIGURE 12.5. Schematic of the Grid-enabled Optimisation PSE.

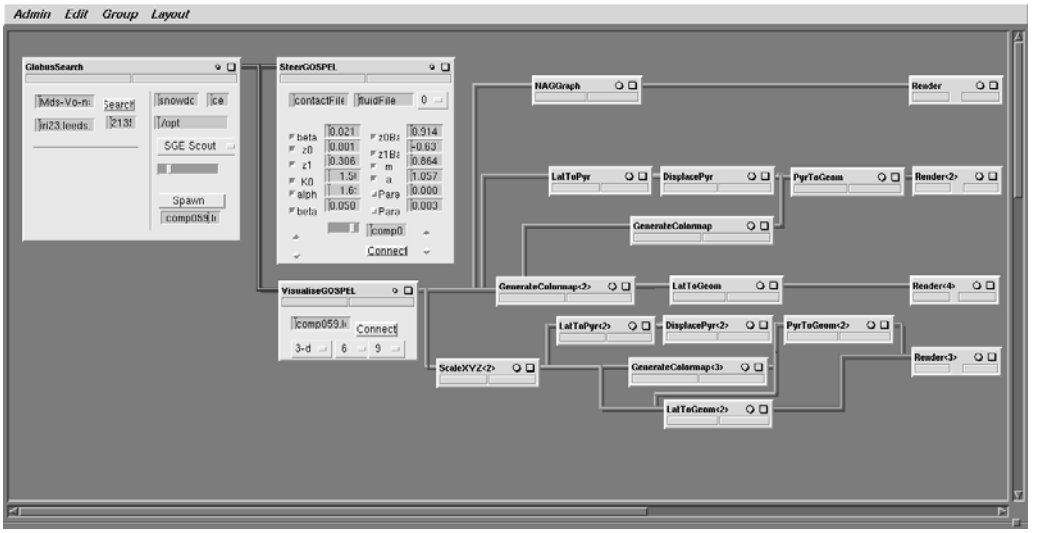


FIGURE 12.6. IRIS Explorer map of the optimisation PSE. Data–flow represented by wires between modules.

ical solver can be run on a single resource with all the messages handled by the fast-internal mechanisms; whereas the less frequent, synchronization communications at the end of each optimization iteration can be performed using the slower TCP/IP messages. In this manner it is possible to fully utilize many much smaller resources than would be typically available in traditional HPC applications.

An example of a typical map for the PSE is shown in Fig. 12.6, where the data–flow pipeline, generally from left to right is clearly visible. The majority of the modules are used in the visualization process and hence only the three modules on the left are described here.

The first module in the map interrogates a Grid information (GIIS) server to analyze the available resources and their current statuses [32]. The user can then select a resource and choose a suitable launch method, including launching the job onto the Grid using Globus. For this work we have extended the gViz library to include parallel launch mechanisms, including writing a parallel job submission script or Globus resource specification language (RSL) script which then gets submitted to Sun Grid Engine for scheduling onto a suitable node. When the job is spawned a socket connection back to the PSE is made telling the launching application which node of the Grid resource the simulation will be communicating from. Information about this node and port is then passed to the next two modules in the map which handle the steering inputs, and the receipt of the data for visualization. Knowledge of where the simulation is running also allows any other user access to the simulation through the gViz libraries. This means that one person, with Grid certification, say, can start the simulation and other collaborators around the world can then all see the results of that simulation and help to steer the computation [26, 32]. In fact, the person who originally launched the Grid job need not actually be involved from that point on.

The steering module has several uses. Firstly it shows the current best set of values found by the simplex, along with $\mathcal{R}_F$. This allows a user access to individual numbers from the simulation rather than much larger data sets for visualization purposes. These numbers can also be used for steering. For example it is possible to resubmit this current best set to the optimizer once

a minimum has been found. The NAG library will then build a new simplex around this previous minimum potentially allowing it to escape from local minima. Similarly, a different point in the search space can be specified away from where the optimizer has previously searched. Finally, as mentioned, the accuracy can be changed. A method we have implemented here is the ability to turn on (or off) the thermal components of the solution. The thermal solve is much more expensive but adds greater accuracy to the friction results obtained, especially for cases where more heat is generated [36].

Communication from the PSE to the simulation is done through the gViz libraries. At suitable points the simulation will check if any new input data has been received. If a steering request is for additional accuracy, say, then these changes can be introduced without changing the points of the current simplex and would therefore only apply to future calculations. If, on the other hand, a new simplex was requested then the NAG libraries do not allow movement of the current simplex points and hence use of the communication flag inside the routine will cause the optimization routine to drop out of the NAG routines and then the new simplex is submitted.

The visualization module communicates with the simulation to receive all the data sets for visualization. These are then packaged up into standard IRIS Explorer data types and sent down the rest of the map for visualization. When the full data sets are being shown then more information needs to be returned from the parallel nodes than is necessary for just the optimization process. The root process which is communicating with any attached users also needs to retain full copies of all output data previously generated so that any listeners joining the simulation later get the full set of results rather than just those generated from that stage.

The full optimization run generates hierarchies of multivariate data. Full descriptions of the data sets returned are described in [34], along with how the different techniques give added information to the user. Here we will content ourselves to simply show how the effect of steering can improve the quality of solutions obtained. In Fig. 12.7, we see the behavior of the variables changed by the optimizer over the course of the entire process. The first graph shown has the optimizer progressing without any steering, the second has a new simplex formed after the 30th improvement to the best point in the simplex. It can be clearly seen how this has encouraged the optimizer to a very different point in the search space which turns out to be a better overall result.

12.6 Conclusions and Future Directions

The use of problem-solving environments provide a visually striking and powerful tool for both developers and users of application code. The visualizations provided allow real-time evaluation of the results generated, and computational steering enables interactivity with running simulations. The use of PSEs will grow as even computationally light applications benefit from such techniques.

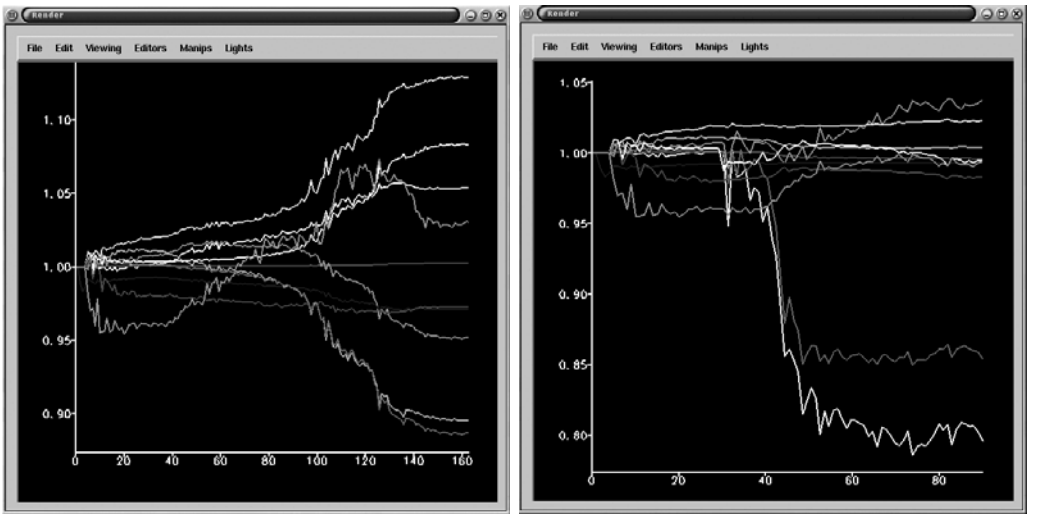
The use of Grid technologies increases the usefulness of the PSE as it potentially allows access to a much richer computational space which may provide the opportunity to learn about the application more quickly. More work is obviously required in the middleware between the PSE and the simulation. Projects such as gViz are providing these interfaces in a way which should be as transparent to the user as possible.

Future challenges relating to PSEs and the Grid mirror those issues affecting general Grid use: resource discovery, resource access, and security. These future directions are discussed briefly below.

Discovery of resources requires more than just knowing what machines are available. Knowledge of the architecture itself governs which executables will and will not run, but further knowledge is required for use of library functions, etc. Brokering of the resource choice will always be an issue, but if varying pecuniary charging models are applied to different resources then these considerations will need to be built in too.

The challenges regarding access extend beyond the authorization to use a resource. When a PSE job is submitted, it may be sensible for immediate access to be provided. This may require special queueing arrangements to be introduced on local machines, rather than the job scheduling software starting the job in the middle of the night, potentially waiting for input. Access is also an issue regarding the connectivity between the PSE and the resource: if the nodes where the job is running are not visible from the desktop, an intermediary staging server should be provided at the interface between the systems. Access to information regarding running simulations will need a standard location, such as a Web service, on each resource to enable users to fully know what is available.

Finally, security of access to information has not been fully developed thus far in the construction of the PSEs. Whilst secure transfer of data between PSE and Grid is possible, authenticated via Globus certificates, this encryption is computationally expensive. Also, since the infrastructure has been built in gViz to allow multiple collaborators, then the issues concerning who will have access to any running simulations needs to be fully addressed. This authorization should probably be tied to the information in the Web service listing of the jobs. Thus, while the use of Grid-based PSEs with computational steering and parallelism is an attractive way to solve computationally intensive problems, there remain many challenges to be addressed before such a paradigm is routinely and widely used.



(a) No steering (b) Steering after 30 iterations

FIGURE 12.7. Progression of optimizer showing relative change of best solution found to initial guess. Each line represents a different variable.

Acknowledgments. This work was funded by EPSRC and through the UK e-Science Core Programme. Thanks are also due to Jason Wood for providing the gViz library used in this work, and Laurence Scales from Shell Global Solutions for work on the lubrication project.

References

[1] I. Foster, and C. Kesselman, The Grid 2: The Blueprint for a New Computing Infrastructure (Elsevier, 2004)

[2] F. Berman, G.C. Fox, and A.J.G. Hey, Grid Computing: Making the Global Infrastructure a Reality (Wiley, 2003)

[3] R.B. Haber, D.A. McNabb, Eliminating Distance in Scientific Computing: An Experiment in Televisualization, International Journal of Supercomputer Applications 4, 71–89 (1990)

[4] K.W. Brodlie, D.A. Duce, J.R. Gallop, J.P.R.B. Walton, and J.D. Wood, Distributed and collaborative Visualization—State of the Art Report, *Computer Graphics Forum* 23, 223–251 (2004).

[5] J. Wood, H. Wright, and K.W. Brodlie, *Collaborative Visualization*, in Proceedings of IEEE *Visualization 97* (1997), pp. 253–259

[6] C.E. Goodyer, M. Berzins, *Eclipse and Ellipse: PSEs for EHL solutions using IRIS Explorer and SCIRun*, in Computational Science, ICCS 2002 Part I, Lecture Notes in Computer Science, Vol. 2329, ed. by Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G. (Springer, 2002), pp. 521–530

[7] K.W. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, A. Gay, *GRASPARC—A Problem-solving Environment Integrating Computation and Visualization*, in IEEE Visualization (IEEE, 1993), pp. 102–109

[8] C.R. Johnson, M. Berzins, L. Zhukov, R. Coffey, *SCIRun: Application to Atmospheric Dispersion Problems Using Unstructured Meshes*, in Numerical Methods for Fluid Mechanics VI. ICFD '98, Oxford ed. by Banies, M.J. (1998), pp. 111–122

[9] G. Allen, E. Seidel, J. Shalf, *Scientific Computing on the Grid*, *Byte* 24–32 Spring (2002)

[10] J. Kierzenka, F., S.L.: A BVP Solver Based on Residual Control and the Matlab PSE, *ACM Transactions on Mathematical Software* 27, 299–316 (2001)

[11] L.A, Treinish, *Interactive, Web-based Three-dimensional Visualizations of Operational Mesoscale Weather Models*, in Proceedings of the Eighteenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, American Meteorological Society (2002), pp. J159–161

[12] N. Furmento, W. Lee, A. Mayer, S. Newhouse, J. Darlington, *ICENI: An Open Grid Service Architecture Implemented with Jini*, in Proceedings of SuperComputing 2002 (2002)

[13] J. Chin, J. Harting, S. Jha, P. Coveney, A. Porter, S.Pickles, *Steering in Computational Science: Mesoscale Modelling and Simulation*, Contemporary Physics 44, 417–434 (2003)

[14] P. Watson, Databases and the Grid, in *Grid Computing : Making the global infrastructure a reality0*, ed. by. Berman, F., Fox, G.C., Hey, A.J.G., (Wiley, 2003), pp. 363–384

[15] S.G. Parker, C.R. Johnson, *SCIRun: A Scientific Programming Environment for Computational Steering*, in Proceedings of Supercomputer '95, New York, ed. by Meuer, H.W. (Springer-Verlag, 1995)

[16] J.P.R.B. Walton, *Now You See It—Interactive Visualisation of Large Datasets*, in Applications of Supercomputers in Engineering III, ed. by Brebbia, C.A., Power, H. (Computatational Mechanics Publications/Elsevier Applied Science, 1993)

[17] Scientific Computing and Imaging Institute (SCI), SCIRun: *A Scientific Computing Problem solving Environment (2002)*, `http://software.sci.utah.edu/scirun.html`

[18] J.D. Mulder, J.J. van Wijk, R. van Liere, A Survey of Computational Steering Environments, *Future Generation Computer Systems* 15, 119–129 (1999)

[19] J.M. Brooke, P.V. Coveney, J. Harting, S. Jha, S.M. Pickles, R.L. Pinning, A.R. Porter, Computational steering in RealityGrid, in Proceedings of the All Hands Meeting 2003, EPSRC, ed. by Cox, S. (2003), pp. 885–888

[20] J.W. Wood, K.W. Brodlie, J.P.R. Walton, gViz: Visualization and Computational Steering for e-Science, in, ed. by Cox, S. Proceedings of the All Hands Meeting 2003, EPSRC (2003), 164–171
[21] D. Dowson, P. Ehret, *Past, Present, and Future Studies in Elastohydrodynamics*, in Proceedings of the Institution of Mechanical Engineers Part J, *Journal of Engineering Tribology* 213, 317–333 (1999)
[22] C.H. Venner, A.A. Lubrecht, *Multilevel Methods in Lubrication* (Elsevier, 2000)
[23] C.E. Goodyer, Adaptive Numerical Methods for Elastohydrodynamic Lubrication. PhD thesis, University of Leeds, Leeds, England (2001)
[24] H. Wright, K.W. Brodlie, T. David, *Navigating High-dimensional Spaces to Support Design Steering*, in VIS 2000 (IEEE, 2000), pp. 291–296
[25] H. Wright, J.P.R.B. Walton, HyperScribe: *A Data Management Facility for the Data–flow Visualization Pipeline*, Technical Report IETR/4, NAG (1996)
[26] M.A. Walkley, J. Wood, K.W. Brodlie, *A distributed collaborative problem-solving Environment*, in Computational Science, ICCS 2002 Part I, Lecture Notes in Computer Science, Vol. 2329, ed. by Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G., (Springer, 2002), pp. 853–861
[27] Message Passing Interface Forum, MPI: A Message-passing Interface Standard, *International Journal of Supercomputer Applications* 8, (1994)
[28] C.R. Johnson, S.G. Parker, *Applications in Computational Medicine Using SCIRun: A Computational Steering Programming Environment*, in Proceedings of Supercomputer '95, New York, ed. by Meuer, H.W. (Springer-Verlag, 1995), pp. 2–19
[29] D. de St. Germain, J. McCorquodale, S. Parker, C.R. Johnson, *Uintah: A Massively Parallel Problem Solving Environment*, in Ninth IEEE International Symposium on High Performance and Distributed Computing (2000)
[30] I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications* 11, 115–128 (1997)
[31] C.E. Goodyer, J. Wood, M. Berzins, *A parallel Grid-based PSE for EHL problems*, in Applied Parallel Computing, Proceedings of PARA '02, Lecture Notes in Computer Science, Vol. 2367, ed. by Fagerholm, J., Haataja, J., Järvinen, J., Lyly, M., Råback, P., Savolainen, V., (Springer, 2002), pp. 523–532
[32] K.W. Brodlie, S. Mason, M. Thompson, M.A. Walkley, J.W. Wood, *Reacting to a Crisis: Benefits of Collaborative Visualization and Computational Steering in a Grid Environment*, in Proceedings of the All Hands Meeting 2002 (2002)
[33] C.E. Goodyer, R. Fairlie, D.E. Hart, M. Berzins, L.E. Scales, *Adaptive Techniques for Elastohydrodynamic Lubrication Solvers*, in Transient Processes in Tribology: Proceedings of the 30th Leeds-Lyon Symposium on Tribology, ed. by Dalmaz et al. (Elsevier, 2004)
[34] C.E. Goodyer, M. Berzins, P.K. Jimack, L.E. Scales, *Grid-based Numerical Optimization in a Problem-solving Environment*, in Proceedings of the All Hands Meeting 2003, EPSRC, ed. by Cox, S. (2003), pp. 854–861
[35] N. Karonis, B. Toonen, I. Foster, MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing* 63, 551–563 (2003)
[36] R. Fairlie, C.E. Goodyer, M. Berzins, L.E. Scales, *Numerical Modelling of Thermal Effects in Elastohydrodynamic Lubrication Solvers*, in Trobological Research and Design for Engineering Systems, Proceedings of the 29th Leeds-Lyon Symposium on Tribology, ed. by D. Dowson et al. (Elsevier, 2003), pp. 675–683