

# The TeraGyroid Experiment

S. M. Pickles<sup>1</sup>, R. J. Blake<sup>2</sup>, B. M. Boghosian<sup>3</sup>, J. M. Brooke<sup>1</sup>, J. Chin<sup>4</sup>, P. E. L. Clarke<sup>5</sup>, P. V. Coveney<sup>4</sup>,  
R. Haines<sup>1</sup>, J. Harting<sup>4</sup>, M. Harvey<sup>4</sup>, S. Jha<sup>4</sup>, M. A. S. Jones<sup>1</sup>, M. Mc Keown<sup>1</sup>, R. L. Pinning<sup>1</sup>,  
A. R. Porter<sup>1</sup>, K. Roy<sup>1</sup>, and M. Riding<sup>1</sup>.

*1. Manchester Computing, University of Manchester, Oxford Road, Manchester, M13 9PL*

*2. CLRC Daresbury Laboratory, Daresbury, Warrington WA4 4AD*

*3. Tufts University, Department of Mathematics, 211 Bromfield-Pearson, Medford, Massachusetts 02155*

*4. Centre for Computational Science, Christopher Ingold Laboratories, University College London,  
20 Gordon Street, London, WC1H 0AJ*

*5. Department of Physics & Astronomy, University College London, Gower Street, London, WC1E 6BT*

The TeraGyroid experiment at SC'03 addressed a large-scale problem of genuine scientific interest at the same time as showing how intercontinental grids enable new paradigms for collaborative computational science that can dramatically reduce the time to insight. TeraGyroid used computational steering to accelerate the exploration of parameter space in condensed matter simulations. The scientific objective was to study the self-assembly, defect pathways and dynamics of liquid crystalline cubic gyroid mesophases using the largest set of lattice-Boltzmann (LB) simulations ever performed, involving in some cases lattices of over one billion sites and for highly extended simulation times. We describe the application in sufficient detail to reveal how it uses the grid to support interactions between its distributed parts, where the interfaces exist between the application and the middleware infrastructure, what grid services and capabilities are used, and why important design decisions were made. We also describe how the resources of high-end computing services were federated with the UK e-Science Grid and the US TeraGrid to form the TeraGyroid testbed, and summarise the “lessons learned” during the experiment.

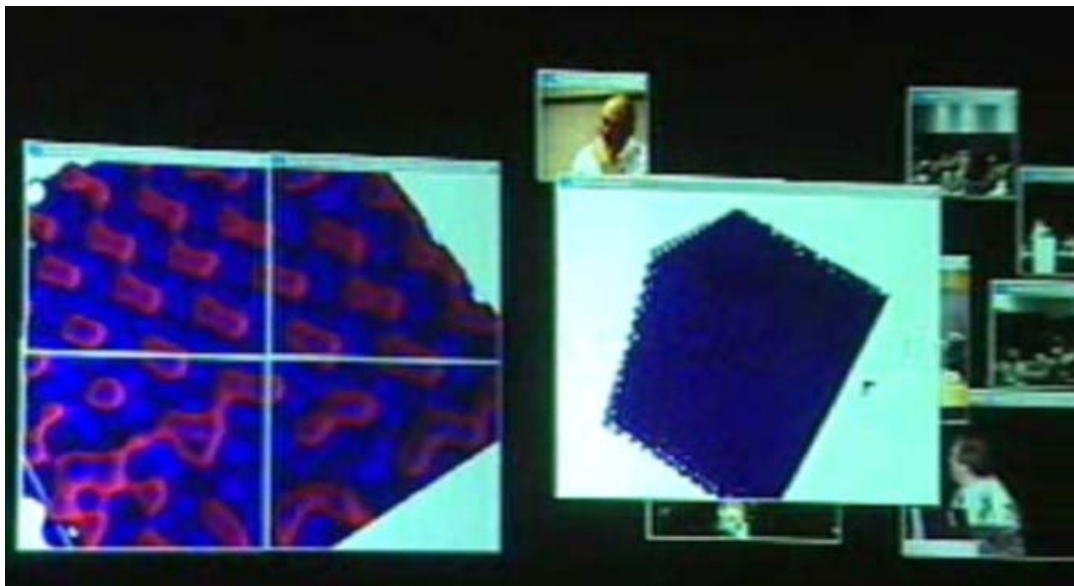
## 1 Introduction

The TeraGyroid project was an ambitious experiment to investigate new opportunities for computational science created by the Grid — for example, demonstrating new scientific capabilities and international collaborative working — at the same time as establishing technical collaborations to support the development of national and international Grids. High-end computing resources were federated with the UK e-Science Grid and the US TeraGrid, and harnessed in an accelerated programme of computational science that peaked during the Supercomputing 2003 conference (SC'03). The application was provided by the RealityGrid project (<http://www.realitygrid.org>).

The scientific objective of TeraGyroid was to study the self-assembly, defect pathways and dynamics of liquid crystalline cubic gyroid mesophases using lattice-Boltzmann techniques [1, 2, 3, 4]. This was achieved through the largest set of lattice-Boltzmann (LB) simulations ever performed, involving in some cases lattices of over one billion sites and highly extended simulation times. The gyroid phase was only very recently discovered in lattice-Boltzmann simulations using the LB3D code [2]; its stability is strongly dependent on minimisation of finite size effects (caused by the imposition of artificial three-dimensional periodic boundary), implying the need to perform “large” simulations (by current standards), on lattices of at least  $128^3$  sites. However, then one observes the presence of local, highly crystalline domains of the fluid at varying orientations, between which there are domain boundaries and a concomitant dynamics of defects. To study the dynamical properties of these defects, which inform the macroscopic properties of the fluid, one needs to perform extremely large simulations in an essentially finite-size-effect free environment, and for long enough to study the way the domains interact. Thus, not only the size of these lattice-Boltzmann simulations but also their duration is quite unprecedented — the computational effort to perform these

studies must be at least on the scale of what we had available within the TeraGyroid project — and, by judicious use of all the RealityGrid capabilities and grid computing infrastructure described below, the time taken to obtain scientific results has been massively reduced to just a few months. Very little is known about the large scale fluid properties of these amphiphilic liquid crystalline systems, either experimentally or theoretically. Some of the simulations, and the detailed analysis of our results, are still ongoing at the time of writing this paper and will be reported elsewhere.

A central theme of RealityGrid is the facilitation of distributed and collaborative exploration of parameter space through computational steering and on-line, high-end visualization [5, 6]. The TeraGyroid experiment realised this theme in dramatic fashion at SC'03. A series of demonstrations under the guise of “Transcontinental RealityGrids for Interactive Collaborative Exploration of Parameter Space” (TRICEPS), won the HPC Challenge competition in the category of “Most Innovative Data Intensive Application” [7]. The same work was also demonstrated live before an Access Grid audience during the SC Global showcase “Application Steering in a Collaborative Environment” [8]. A family of simulations at different resolutions were spawned and steered into different regions of parameter space by geographically distributed scientists on both sides of the Atlantic. The collaborative environment of Access Grid was essential, not only for project planning, but also for managing the distributed experiment and discussing the evolution of the simulations. Figure 1 shows a snapshot of the Access Grid screen as seen in Phoenix during SC Global.



**Figure 1: Access Grid screen as seen in Phoenix during the SC Global session on application steering.**

The remainder of this paper is organised as follows. In section 2, we describe RealityGrid’s approach to computational steering and its application to the efficient exploration of parameter space. We describe the architecture in sufficient detail to reveal how the Grid is used to support interactions between the parts of the distributed application, where the interfaces exist between the parts of the application and the middleware infrastructure, what Grid services and capabilities are used, and why important design decisions were made. In section 3, we describe the TeraGyroid testbed, its resources and networks, and the “lessons learned” during the experiment: problems encountered, solutions adopted and issues outstanding.

## 2 Computational Steering and Parameter Space Exploration

Computational steering — the ability to interact with the simulation while it is running — was central to the methodology of the TeraGyroid project. By monitoring the progress of simulations, aided by on-line visualization, we avoid losing cycles to redundant computation or even doing the wrong calculation. By tuning the value of steerable parameters, we quickly learn how the simulation responds to perturbations and use this insight to design subsequent computational experiments. Mapping out the parameter space of the model under study on relatively coarse lattices ( $64^3$  and  $128^3$ ) informs the choices for parameter values for the most expensive simulations on the largest lattices ( $512^3$ ,  $1024^3$ ). We now describe our computational steering technologies and how these facilitate parameter space exploration.

### 2.1 The RealityGrid Steering Library and Toolkit

We instrument our applications for computational steering using the RealityGrid steering library [9]. A fully instrumented application supports the following operations:

- Pause, resume, detach and stop
- Set values of steerable parameters
- Report values of monitored (read-only) parameters
- *Emit* "samples" to remote systems for e.g. on-line visualization
- *Consume* "samples" from remote systems
- Checkpoint and windback.

*Emit* and *consume* semantics are used because the application should not be aware of the destination or source of the data. *Windback* here means revert to the state captured in a previous checkpoint without stopping the application. In RealityGrid, the act of taking a checkpoint is the responsibility of the application. LB3D supports *malleable checkpoints*, by which we mean that the application can be restarted on a different number of processors on a system of different architecture.

In designing and implementing the steering library, we have aimed to enable existing parallel programs (often written in Fortran90 or C and designed for multi-processor supercomputers) to be made steerable with a minimum of effort. The steering library is implemented in C, has both Fortran 90 and C bindings, and permits any parallelisation technique (such as MPI or OpenMP), with the proviso that the application developer assumes responsibility for communicating any changes resulting from steering activity to all processes. The steering library requires the application to register both monitored (read-only) and steerable (changed only through user interaction) parameters. Similarly, the user may instruct the application to take a checkpoint or restart from an existing one. We use a system of reverse communication with the application. This means that, for actions such as emit, consume, checkpoint and windback, the library simply notifies the application of what it needs to do. It is then the application's responsibility to carry out the task, possibly using utility routines from the steering library.

Using the client-side part of the library, a *generic* steering client has been implemented using C++ and the Qt GUI toolkit; web-based and PDA clients are under development. The client discovers the commands supported by the application and its monitored/steered parameters as part of the connection process, and constructs the necessary widgets on the fly.

Checkpoint/recovery is a key piece of functionality for computational steering. Sometimes the scientist realises that an interesting transition has occurred, and wants to study the transition in more detail; this can be accomplished by winding back the simulation to an earlier checkpoint, and increasing the frequency of sample emissions for on-line visualization.

An even more compelling scenario arises when computational steering is used for parameter space exploration, as in TRICEPS. A scientist may be studying a physical system which is suspected to contain a rich phase structure, but does not have sufficient resources available to embark on a brute-force exploration of its multi-dimensional parameter space. Instead, the scientist uses computational steering to begin mapping out this space. The simulation evolves under an initial choice of parameters until the first signs of emergent structure are seen, and a checkpoint is taken. The simulation evolves further, until the scientist recognises that the system is beginning to equilibrate, and takes another checkpoint. Suspecting that further equilibration will not yield any new insight, the scientist now rewinds to an earlier checkpoint, chooses a different set of parameters, and observes the system's evolution in a new direction. In this way, the scientist assembles a tree of checkpoints — our use of *checkpoint trees* was inspired by GRASPARC [10] — that sample different regions of the parameter space under study, while carefully husbanding his or her allocation of computer time. Different branches of the tree can be explored in parallel. The scientist can always revisit a particular branch of the tree at a later time should this prove necessary. This process is illustrated in Figure 2, in which a Lattice-Boltzmann simulation is used to study the phase structure of a mixture of fluids. Here one dimension of the parameter space is explored by varying the surfactant-surfactant coupling constant  $g_{ss}$ .

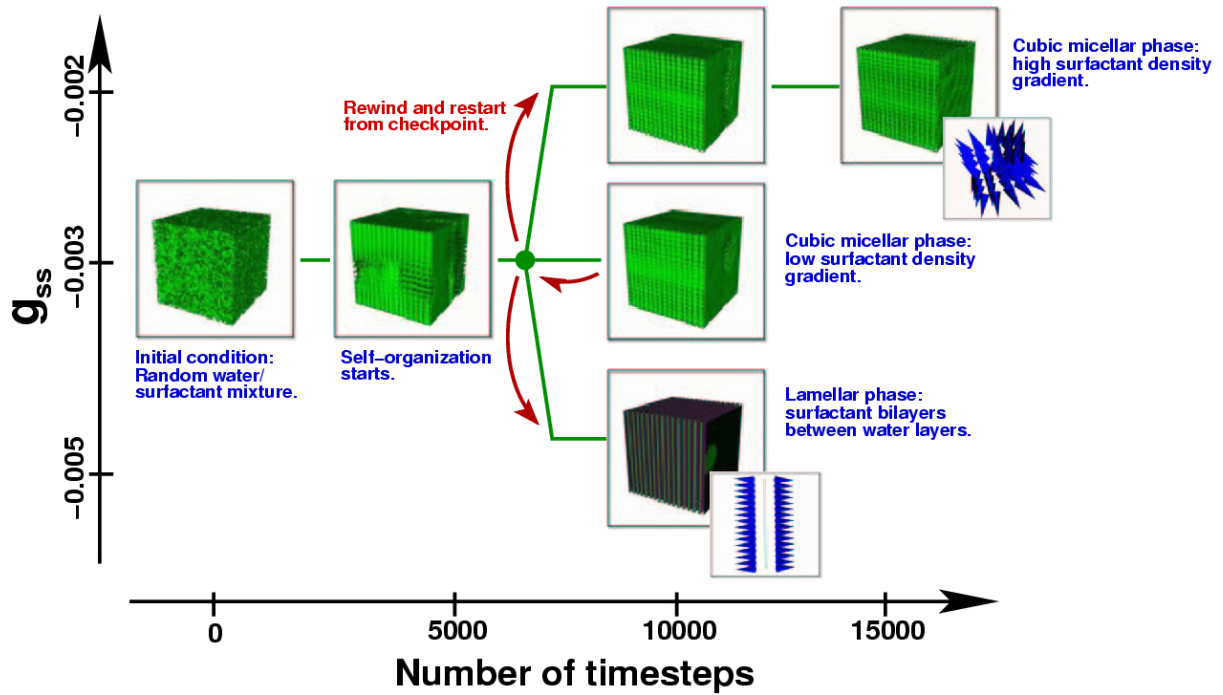
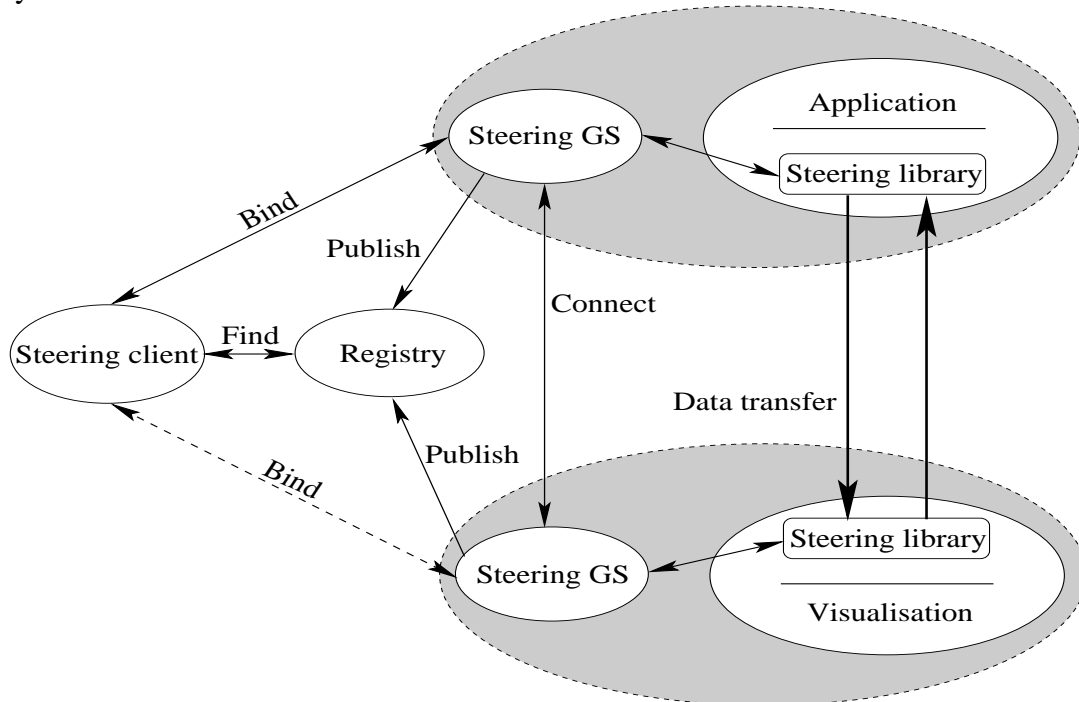


Figure 2. Parameter space exploration gives rise to a tree of checkpoints.

## 2.2 The Architecture of Steering

Figure 3 shows a schematic representation of our three-tier architecture for computational steering for the case in which a visualization component is connected to the simulation component. One or both of these components may be steered using the steering client. Both components are started independently and can be attached and detached dynamically. The service-oriented middle tier solves numerous issues that are problematic in a two-tier architecture, as we learned from our experiences with a prototype in which the client connected directly to the simulation.

By exposing the “knobs” (control) and “dials” (status) of computational steering as operations of a Web service, the protocol of computational steering is easily documented in WSDL, made accessible in a multitude of different languages, and permits independent development of steering clients. These clients may be stand-alone, web-based, or embedded in a Modular Visualization Environment (such as AVS, Amira or Iris Explorer) and may be customised for application-specific requirements. The mediating “Steering Grid Service” (SGS) acts as a white board. The client pushes control commands to and pulls status information from the SGS, while the application performs the inverse operations. This model facilitates asynchronous messaging. The SGS is stateful and transient, with state and lifetime reflecting the component being steered, and is a natural application for the service data constructs and factory and lifetime management patterns provided by the Open Grid Services Infrastructure (OGSI) [11]. Steering clients can either use the client-side part of the steering API to communicate with the SGS, or directly drive operations on the SGS using standard Web service methods. We implement the SGS in OGSI::Lite [12, 13], itself a lightweight Perl-based implementation of OGSI. The easily satisfied dependencies of OGSI::Lite mean that it can be deployed in user space on almost any system (including important platforms for which Java is not available). Thus we can, and do, use this technology on grids that are based on version 2 of the Globus Toolkit (GT2), and we sidestep the issue of needing to dynamically deploy Grid services in someone else’s container.



**Figure 3: Schematic architecture of an archetypal RealityGrid configuration.**

Although we have working solutions based on OGSI, we look forward to the Web Services Resource Framework (WSRF) [14], which solves a problem for us. In principle, the application and client should see different interfaces to the SGS (e.g. the operation to set the value of a monitored parameter should be visible only to the application, not to the client); WSRF allows us to define different Web service interfaces to the same stateful resource.

Our ability to deploy the SGS either on the same host as the simulation, or anywhere else on the Grid, has helped us to circumvent various firewall problems. Firewall issues are another reason why we use a whiteboard pattern instead of the OGSI notification mechanisms.

The registry in Figure 3 is implemented in OGSI::Lite using the OGSI service group constructs. The registry is a persistent service that acts as a central point of contact for client,

simulation and visualization to discover each other, and is essential for bootstrapping communications.

The services that support the checkpoint tree are not shown in Figure 3. Each node in the tree is implemented as a persistent, long-lived Grid service containing metadata about the simulation, such as input decks, location of checkpoint files and so on. The persistence of the tree is achieved by exploiting the ability of OGSI:Lite to mirror, transparently, the service data of a Grid service in a database.

On-line, real-time visualization is an important adjunct for many steering scenarios, but we avoid confusing the two, deliberately separating visualization from steering. Control and status information are the province of steering. Visualization is concerned with the consumption of samples emitted by the simulation. The reverse route can be used to emit samples from the visualization system to the simulation, where it might be interpreted for example as a new set of boundary conditions, as in NAMD [15] and VMD [16]. While we regard this separation as being fundamental at an architectural level, we do not insist that the same separation is reflected in user interfaces. Thus it is often desirable from an HCI perspective to allow the user to steer a simulation from control panels integrated into a Modular Visualization Environment, and conceptually it is attractive to think of the simulation as being a data source at the beginning of some kind of extended visualization pipeline. Indeed, we have designed our architecture to encourage such re-purposing.

All the communications with Web or Grid services in Figure 3 use SOAP over http or https as the transport mechanism. The application communicates with the SGS using gSOAP within the steering library. The volume of data transported in this way is fairly low, and the overhead of SOAP is not significant for us — a client can poll the SGS many times a second. However, the volume of data exchanged between simulation and visualization is much greater, and high-performance transport mechanisms are needed. We provide several mechanisms, such as: (a) writing to and reading from disk, relying on a shared file system or a daemon responsible for transferring files from source to destination; or (b) direct process-to-process communication using sockets. Our current implementation of (b) is based on Globus-IO, but this introduces a dependency that complicates the process of building the steering library and any applications that use it.

## **2.3 Run-time deployment and job management**

We have endeavoured to separate the concerns of computational steering from the concerns of how jobs and services are deployed. Thus a developer instrumenting a code for steering is unaware of the existence of any remote services or visualization components; the developer fulfils his or her contract with the library by providing through the API the information about the state of the simulation that the library needs to do its job, and by carrying out operations that affect the state of the simulation when instructed to do so by the library. Similarly, the client-side part of the steering library is used within the generic steering client to discover, connect to and interact with a steerable entity; but is completely unaware of whether the simulation and visualization are running locally or remotely.

Yet simulations and visualizations must be launched, and services must be deployed. We do this using a suite of scripts that use the command-line interfaces to the GRAM and GridFTP components of the Globus Toolkit. We also provide a graphical tool, or “wizard”, that allows the user to choose an initial condition (which can be done by editing the input deck, and/or browsing the checkpoint tree and selecting a pre-existing checkpoint), choosing a computational resource, launching the job (automatically starting the SGS), and starting the steering client. The wizard also provides capabilities to select a visualization technique and



visualization system, start the visualization component and connect it to the simulation. The wizard can migrate a running job to a different resource on the Grid, which involves taking a checkpoint, transferring the checkpoint files, and re-starting the application on the new host, in a complex process that involves fifteen distinct services and twelve distinct hosts. The wizard shells out to command-line scripts, which in general require some customisation for each application, to accomplish these tasks. With the exception of Globus-IO, we do not program to the Globus APIs directly.

Note that we do not use the MDS components of Globus (GIIS and GRIS). This is not because of any doubt about the importance of such services — we view them as essential for the long-term future of the Grid — but because our experiences with MDS as deployed on the grids available to us give us reason to doubt the robustness of the services and the utility of the information published through them. Instead, we maintain lists of computational and visualization resources in client-side configuration files read by the wizard.

### 3 Practical experiences: TeraGyroid, TRICEPS and SC Global

#### 3.1 The TeraGyroid testbed

The principle computational resources in the testbed are listed in Table 1. We also used a number of visualization systems including SGI Onyx systems at Manchester, UCL, NCSA and Phoenix (on loan from SGI and commissioned on site), and the TeraGrid visualization cluster at Argonne. The RealityGrid registry ran on a Sony PlayStation 2 in Manchester. The Globus Toolkit was installed on all systems, with various versions (2.2.3, 2.2.4, 2.4.3 and 3.1) in use. Our use of Globus (GRAM, GridFTP and Globus-IO) exposed no incompatibilities between these versions, but we note that the GT 3.1 installation included the GT 2 compatibility bundles.

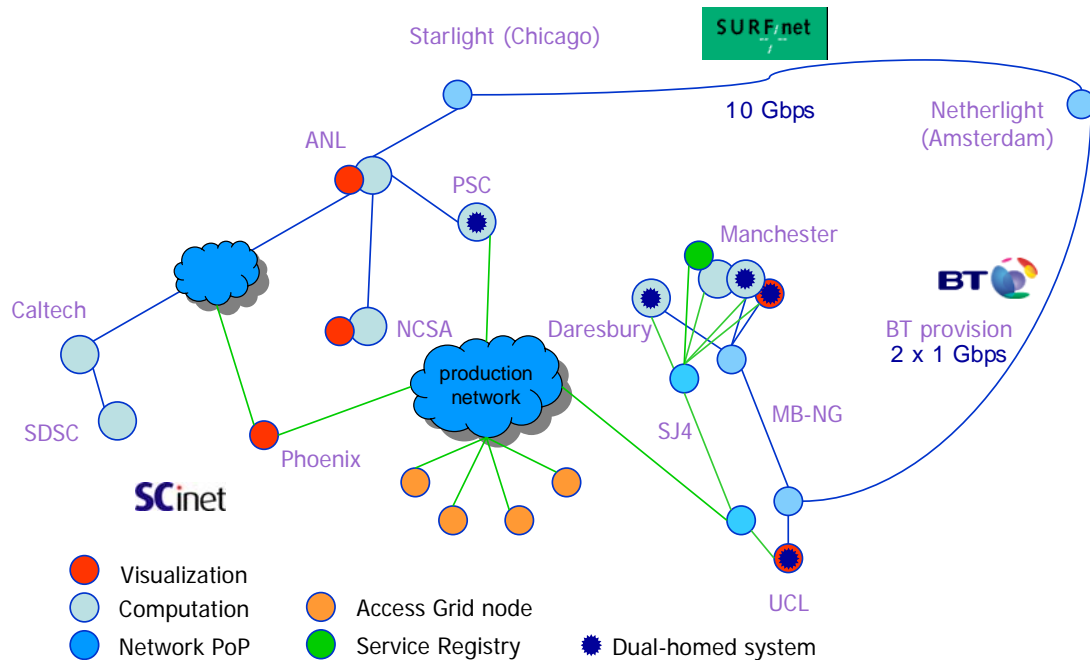
Site	Architecture	Processors	Peak (TF)	Memory (TB)
HPCx (Daresbury)	IBM Power 4 Regatta	1024	6.6	1.0
CSAR (Manchester)	SGI Origin 3800	512	0.8	0.5 (shared)
CSAR (Manchester)	SGI Altix	256	1.3	0.38
PSC (Pittsburgh)	HP-Compaq	3000	6	3.0
NCSA (Urbana-Champaign)	Itanium 2	512	2.0	4.0
SDSC (San Diego)	Itanium 2	256	1.3	1.0

**Table 1: Principle computational resources in the testbed**

The testbed and networks are depicted schematically (in much simplified form) in Figure 4. We knew from previous experience [17] that the bandwidth available to us from production networks would seriously hamper our ability to migrate jobs — which requires transfer of large (0.5 TB for our largest  $1024^3$  simulation) checkpoint files — across the Atlantic, and to generate data on one continent and visualize it on the other. Fortunately, BT donated two 1 Gbps links from London to Amsterdam, which, in conjunction with the high bandwidth SurfNet provision and the TeraGrid backbone, completed the circuit to the SC'03 exhibition floor at Phoenix. Unfortunately, we only had two weeks to debug the network. HPCx, the CSAR Origin 3800, and the visualization supercomputers in Manchester and UCL were all “dual-homed”, connected simultaneously to the SuperJanet 4 academic backbone and the BT provision via the experimental MB-NG (<http://www.mb-ng.net>) network. Complex routing tables were required on the UK end, while route advertisement sufficed on the US end.

Scientists and developers collaborated using Access Grid nodes located in Boston, London, Manchester, Martlesham, and Phoenix. The tiled display on the left of Figure 1 was rendered in real time at the TeraGrid visualization cluster located at Argonne, using Chromium. The

display on the right was rendered, also in real time, on an SGI Onyx system in Manchester. The visualizations are implemented using VTK with patches to enable automatic refresh whenever a new sample arrives from the simulation. The video streams were multicast to Access Grid using the FLXmitter library. SGI OpenGL VizServer™ was used to allow a remote collaborator to take control of the visualization. When necessary, the steering client was shared using VNC. An IRC back channel also proved invaluable. All Access Grid and VizServer traffic was routed over production networks.



**Figure 4: The testbed used in the TeraGyroid project**

### 3.2 Lessons Learned

Inevitably, the application and visualization must be ported to, and deployed on, each resource on the Grid where they are to be run. We do not find that Globus-based grids today do much to facilitate this. Instead, the increasing variety of resources that the Grid makes accessible means that users and developers are spending an ever-increasing fraction of their time porting applications to new systems and configuring them for the vagaries of each.

Considerable negotiation was necessary to persuade TeraGrid sites to recognise user and host certificates issued by the UK e-Science Certificate Authority (CA), and to persuade UK sites to recognise the five different CAs used within TeraGrid. In our case, the agreement was facilitated by a sense of common purpose. In general, however, establishing the necessary trust relationships to federate two grids is a process that will scale poorly if every systems administrator has to study the CP and CPS of every CA; there is a clear role here for Policy Management Authorities (see <http://www.gridpma.org>).

It is normally the task of the RealityGrid migration wizard on the user's laptop to initiate the transfer of checkpoint files using the third-party file-transfer capability of GridFTP. This proved problematic when one of the systems involved was dual-homed, and complicated by the fact that the systems were not accessible from everywhere via their MB-NG addresses. The problem could have been circumvented if GridFTP provided the ability to specify different addresses for control and data channels. We worked around the problem by setting



up a checkpoint transfer service on one of the dual-homed systems for which both networks were visible. This transfer service was aware of which hosts were dual-homed, which interface would provide the best bandwidth for each pair of endpoints, which host certificate to associate with an interface for Globus authentication, which options to globus-url-copy yielded the best performance, and which checkpoint replica was “closest” to the site where it was required. The transfer service was implemented as a Perl script and invoked remotely using globusrun, which introduced a significant start-up overhead over normal third-party transfers.

It is often difficult to identify whether a problem is caused by middleware or networks. We are used to the symptoms caused by the presence of firewalls, and were generally able to resolve these speedily once the relevant administrators were identified. However, more baffling problems arose. One problem, which manifested as an authentication failure, was ultimately traced to a router unable to process jumbo frames. Another was traced to inconsistent results of reverse DNS lookups on one of the TeraGrid systems. A third problem, manifesting as grossly asymmetric performance between ftp get and put operations, was never completely resolved, but MTU settings appear to have been implicated.

The maximum performance we achieved for trans-Atlantic file transfers during SC’03 was about 600 Mbps, a respectable fraction of the theoretical upper limit of 1 Gbps. But for some pairs of endpoints, the best we could achieve was a good deal less than this, for a variety of reasons. UDP tests between UCL and Phoenix yielded >95% of the theoretical peak, which suggests that there is still room for improvement in TCP/IP stacks and GridFTP.

In order for the simulation to exchange data with the visualization it is necessary for at least one process of the simulation to establish a connection with the visualization system. This is problematic on HPCx, where the back-end compute nodes are confined to a private network. On other cluster systems, such as Lemieux at PSC or Newton at CSAR, only some nodes have direct connections to the internet. In the latter case, it is possible, by arrangement with the system administrator, to pin one process of the application to the node with internet connectivity. In the former case, the only option is to forward internet connections via the front-end machine or head node. On HPCx, the forwarding of connections was carried out using bespoke software written by Stephen Booth at EPCC.

Even the simplest computational steering scenarios require co-allocation of computational and visualization resources [18]. For a scheduled collaborative steering session, it is also necessary to reserve these resources in advance at a time that suits the people involved. When Access Grid is used to provide the collaborative environment, one usually needs to book the physical rooms as well. Although some batch queuing systems support advance reservation, such support is far from universal, and standard protocols for reservation are still lacking. Making the necessary arrangements will continue to be problematic for some time to come.

## **4 Conclusions**

In the TeraGyroid experiment, we studied the cubic gyroid mesophase of liquid crystals using the largest set of lattice-Boltzmann (LB) simulations ever performed. By harnessing the federated resources of the TeraGrid and the UK e-Science Grid and, by judicious use of all the RealityGrid capabilities and grid computing infrastructure described here, the time taken to obtain scientific results was massively reduced to just a few months. We have shown how intercontinental grids enable new paradigms for collaborative computational science. However, we find that the capabilities of today’s grids are rudimentary, and heroic efforts are required to exploit them effectively.

## 5 Acknowledgments

The TeraGyroid project was supported by the Engineering and Physical Sciences Research Council (EPSRC) in the UK, and the National Science Foundation (NSF) in the USA. RealityGrid is an e-Science pilot project funded by the EPSRC as part of the UK e-Science programme. We are indebted to BT for donating the network connection from London to Amsterdam in order to connect with the transatlantic network connection from SurfNet, to whom we are also grateful. We would especially like to thank the hundreds of individuals in more than thirty institutions who helped to make TeraGyroid a success. We regret that space restrictions do not permit us to list them all here.

## 6 References

- [1] N. González-Segredo, M. Nekovee & P. V. Coveney, *Phys. Rev. E* **67**, 046304 (2003); M. Nekovee and P. V. Coveney, *J. Am. Chem. Soc.* **123**, 12380 (2001).
- [2] N. González-Segredo & P. V. Coveney, "Self-assembly of the gyroid cubic mesophase: lattice-Boltzmann simulations", *Europhys. Lett.* (in press, 2004) ; N. González-Segredo and P. V. Coveney, "Coarsening dynamics of ternary amphiphilic fluids and the self-assembly of the gyroid and sponge mesophases: lattice-Boltzmann simulations" *Phys. Rev. E*, (in press, 2004).
- [3] H. Chen, B. M. Boghosian, P. V. Coveney & M. Nekovee, *Proc. R. Soc. London A* **456**, 2043 (2000).
- [4] P. J. Love, M. Nekovee, P. V. Coveney, J. Chin, N. González-Segredo & J. M. R. Martin, *Comp. Phys. Commun.* **153**, Issue 3, 340-358 (2003).
- [5] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning and A. R. Porter, *Computational Steering in RealityGrid*, Proceedings of the UK e-Science All Hands Meeting, September 2-4, 2003 (<http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf>).
- [6] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter and S. M. Pickles, "Steering in computational science: mesoscale modelling and simulation", *Contemporary Physics* **44**, 417-434, 2003.
- [7] Stephen M. Pickles, Peter V Coveney and Bruce M Boghosian, *Transcontinental RealityGrids for Interactive Collaborative Exploration of Parameter Space (TRICEPS)*, Winner of SC'03 HPC Challenge competition in the category "Most Innovative Data-Intensive Application", ([http://www.sc-conference.org/sc2003/inter\\_cal/inter\\_cal\\_detail.php?eventid=10701#5](http://www.sc-conference.org/sc2003/inter_cal/inter_cal_detail.php?eventid=10701#5)).
- [8] Mathilde Romberg, John Brooke, Thomas Eickermann, Uwe Woessner, Bruce Boghosian, Maziar Nekovee, Peter Coveney, *Application Steering in a Collaborative Environment*, SC Global conference, November, 2003 ([http://www.sc-conference.org/sc2003/inter\\_cal/inter\\_cal\\_detail.php?eventid=10719](http://www.sc-conference.org/sc2003/inter_cal/inter_cal_detail.php?eventid=10719)). A Windows Media Stream archive of this session is available at <mms://winmedia.internet2.edu/VB-on-Demand/AppSteering.asf>.
- [9] Stephen Pickles, Robin Pinning, Andrew Porter, Graham Riley, Rupert Ford, Ken Mayes, David Snelling, Jim Stanton, Steven Kenny, Shantenu Jha, *The RealityGrid Computational Steering API*, Version 1.0, 9 July 2003, unpublished.
- [10] K. W. Brodli, L. A. Brankin, G. A. Banecki, A. Gay, A. Poon and H. Wright. *GRASPARC: A problem solving environment integrating computation and visualization*. In G. M. Nielson and D. Bergeron, editors, *Proceedings of IEEE Visualization 93 Conference*, p. 102. IEEE Computer Society Press, 1993.
- [11] S. Tuecke et al., *Open Grid Services Infrastructure (OGSI) (draft)*. OGSI Working Group of the Global Grid Forum, 2003 ([http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23\\_2003-02-17.pdf](http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf)).
- [12] M. Mc Keown, *OGSI::Lite – a Perl implementation of an OGSI-compliant Grid Services Container*. (<http://www.sve.man.ac.uk/Research/AtoZ/ILCT>).
- [13] J. Chin & P.V. Coveney, *Towards tractable toolkits for the Grid: a plea for lightweight, usable middleware*, 2003 (<http://www.realitygrid.org/lgpaper21.htm>).
- [14] Ian Foster et al., *Modeling Stateful Resources with Web Services*, January 2004, (<http://www.globus.org/wsrf/ModelingState.pdf>).
- [15] Laxmikant Kal, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, **151**:283-312, 1999.
- [16] Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", *J. Molec. Graphics*, 1996, vol. **14**, pp. 33-38.
- [17] S.M. Pickles, J.M. Brooke, F. Costen, E. Gabriel, M. Mueller, M. Resch, *Metacomputing across intercontinental networks*, *Future Generation Computer Systems* **17** (2001) 911-918.
- [18] Karl Czajkowski, Stephen Pickles, Jim Pruyne, Volker Sander, *Usage Scenarios for a Grid Resource Allocation Agreement Protocol*, GGF Memo, February 2003.