The TeraGyroid project – collaborative steering and visualization in an HPC Grid for modelling complex fluids

Jonathan Chin and Peter V. Coveney

Centre for Computational Science, Department of Chemistry,
University College London, Christopher Ingold Laboratories,
20 Gordon Street, London WC1H 0AJ, United Kingdom

Jens Harting

Institute for Computer Applications 1, University of Stuttgart,
Pfaffenwaldring 27, D-70569 Stuttgart, Germany

The TeraGyroid experiment[1] addressed a large scale problem of genuine scientific interest and showed how intercontinental Grids enable new paradigms for collaborative computational science that can dramatically reduce the time to insight. TeraGyroid used computational steering over a Grid to study the self-assembly and dynamics of gyroid mesophases (found in novel materials and living cells) using the largest set of lattice Boltzmann simulations ever performed.

I. INTRODUCTION

A. Complex Fluids

The term "simple fluid" usually refers to a fluid which can be described to a good degree of approximation by macroscopic quantities only, such as the density field $\rho(\mathbf{x})$, velocity field $\mathbf{v}(\mathbf{x})$, and perhaps temperature $T(\mathbf{x})$. Such fluids are governed by the well-known Navier-Stokes equations[2], which, being nonlinear, are difficult to solve in the most general case, with the result that numerical solution of the equations has become a common tool for understanding the behaviour of "simple" fluids, such as water or air.

Conversely, a "complex fluid" is one whose macroscopic flow is affected by its microscopic properties. A good example of such a fluid is blood: as it flows through vessels (of order millimetres wide and centimetres long), it is subjected to shear forces, which cause red blood cells (of order micrometres wide) to align with the flow so that they can slide over one another more easily, causing the fluid to become less viscous; this change in viscosity in turn affects the flow profile. Hence, the macroscopic blood flow is affected by the microscopic alignment of its constituent cells. Other examples of complex fluids include biological fluids such as milk, cell organelles and cytoplasm, as well as polymers and liquid crystals. In all of these cases, the density and velocity fields are insufficient to describe the fluid behaviour, and in order to understand this behaviour, it is necessary to treat effects which occur over a very wide range of length and time scales.

This length and time scale gap makes complex fluids even more difficult to model than "simple" fluids. While numerical solutions of the macroscopic equations are possible for many simple fluids, such a level of description may not exist for complex fluids, yet simulation of every single molecule involved is computationally infeasible.

B. Mesoscale modelling

Over the last decade, significant effort has been invested in understanding complex fluids through computational mesoscale modelling techniques. These techniques do not attempt to keep track of the state of every single constituent element of a system, nor do they use an entirely macroscopic description; instead, an intermediate, *mesoscale* model of the fluid is developed, coarse-graining microscopic interactions enough that they are rendered amenable to simulation and analysis, but not so much that the important details are lost. Such approaches include Lattice Gas Automata[3–6], the Lattice Boltzmann equation[7–12], Dissipative Particle Dynamics[13–15], or the Malevanets-Kapral Real-coded Lattice Gas[16–19]. Recently-developed techniques[20, 21] which use hybrid algorithms have shown much promise.

C. Amphiphile mesophases

In a mixture containing many different fluid components, an amphiphile is a kind of molecule which is composed of two parts, each part being attracted towards a different fluid component. For example, soap molecules are amphiphiles, containing a head group which is attracted towards water, and a tail which is attracted towards oil and grease; analogous molecules can also be formed from polymers. If many amphiphile molecules are collected together in solution, they can exhibit highly varied and complicated behaviour, often assembling to form amphiphile mesophases, which are complex fluids of significant theoretical and industrial importance. Some of these phases have long-range order, yet remain able to flow, and are called liquid crystal mesophases. Of particular interest to us are those with cubic symmetry, whose properties have been studied experimentally [22–24] in lipid-water mixturess[22], diblock copolymers[25], and in many biological systems[26].

It was recently shown by González and Coveney[27] that the dynamical self-assembly of a particular amphiphile mesophase, the gyroid, can be modelled using the lattice Boltzmann method. This mesophase was observed to form from a homogeneous mixture, without any external constraints imposed to bring about the gyroid geometry, which is an emergent effect of the mesoscopic fluid parameters.

It is important to note that this method allows examination of the dynamics of mesophase formation, since most treatments to date have focussed on properties or mathematical description[28, 29] of the static equilibrium state. In addition to its biological importance, there have been recent attempts[30] to use self-assembling gyroids to constructing nanoporous materials.

During the gyroid self-assembly process, several small, separated gyroid-phase regions or domains may start to form, and then grow. Since the domains evolve independently, the independent gyroid regions will in general not be identical, and can differ in orientation, position, or unit cell size; grain-boundary defects arise between gyroid domains. Inside a domain, there may be dislocations, or line defects, corresponding to the termination of a plane of unit cells; there may also be localised non-gyroid regions, corresponding to defects due to contamination or inhomogeneities in the initial conditions. Understanding such defects is therefore important for our knowledge of the dynamics of surfactant systems, and crucial for an understanding of how best to produce mesophases experimentally and industrially.

In small-scale simulations of the gyroid, the mesophase will evolve to perfectly fill the simulated region, without defects. As the lattice size grows, it becomes more probable that multiple gyroid domains will emerge independently, so that grain boundary defects are more likely to appear, and the time required for localized defects to diffuse across the lattice increases, making it more likely that defects will persist. Therefore, examination of the defect behaviour of surfactant mesophases requires the simulation of very large systems.

II. THE TERAGYROID SIMULATION GRID

Sufficiently large-scale simulations of surfactant dynamics require large amounts of CPU time and disk space; the use of Grid technologies makes such simulations feasible. Other novel techniques such as steering, migration, and real-time visualization, were also deployed, as described below.

A. Logistics

The LB3D code used for the simulations was designed to run on parallel computers, using MPI for communication. In each simulation, the fluid is discretized onto a cuboidal lattice, each lattice point containing information about the fluid in the corresponding region of space. Each lattice site requires about a kilobyte of memory per lattice site so that, for example, a simulation on a 128^3 lattice would require around 2.2GB memory, and a single system checkpoint would require about 2.2GB of disk space.

The output from a simulation usually takes the form of a single 4-byte floating-point number for each lattice site, representing, for example, the density of a particular fluid component at that site. Therefore, a density field snapshot from a 128^3 system would produce output files of around 8MB.

As a conservative rule of thumb, the code runs at approximately 10^4 lattice site updates per second per CPU on a fairly recent machine, and has been observed to have roughly linear scaling up to order 10^3 compute nodes. A 128^3 simulation contains around 2.1×10^6 lattice sites; running it for 1000 timesteps requires 2.1×10^9 site updates, or 2.1×10^5 CPU seconds: this is about an hour of real time, split across 64 CPUs. The largest simulation performed used a 1024^3 lattice; the smallest, 64^3. Around $2TB$ of data was generated in total.

B. Migration

Table I describes the high-performance computing machines on which most of the simulation work was performed. Such machines are typically heavily used, so that intensive jobs are submitted in advance to a batch queue, and then run when enough resources are available to do so. The situation frequently arises, therefore, that while a simulation is running on one machine, CPU time becomes available on another machine which may be able to run the job faster or cheaper. The LB3D program which was used to perform these simulations, has the ability to "checkpoint" its entire state to a file. This file can then be moved to another machine, and the simulation restarted there, even if the new machine has a different number of CPUs or even a completely different architecture. It has been verified that the simulation results are independent of the machine on which the calculation runs, so that a single simulation may be migrated between different machines as necessary without affecting its output.

C. Computational Steering

The technique of computational steering[31–33] has been used successfully in smaller-scale simulations to optimize resource usage. Typically, the procedure for running a simulation of the self-assembly of a mesophase would be to set up the initial conditions, and then submit a batch job to run for a certain, fixed number of timesteps. If the timescale for structural assembly is unknown then the initial number of timesteps for which the simulation runs is, at best, an educated guess. It is

Machine name	Location	Number of CPUs	Architecture	Peak performance (Tflops)
HPCx	Daresbury, UK	1280	IBM Power4 Regatta	6.6
Lemieux	PSC, Pittsburgh, PA, USA	3000	HP/Compaq	6
TeraGrid Itanium2 Cluster	NCSA, USA	256	Itanium2	1.3
TeraGrid Itanium2 Cluster	SDSC, USA	256	Itanium2	1.3
Green	CSAR, Manchester, UK	512	SGI Origin3800	0.512 (shared)
Newton	CSAR, Manchester, UK	256	Itanium2	0.384 (shared)

TABLE I: The main TeraGyroid simulation machines

Machine name	Location	Architecture
Bezier	Manchester, UK	SGI Onyx 300, 6x IR3 graphics pipes, 32 CPUs
Dirac	London, UK	SGI Onyx 2, 2x IR3 graphics pipes, 16 CPUs
SGI loan machine	Phoenix, AZ, USA	SGI Onyx, 1xIR4 graphics pipe, 1xIR3 pipe
TeraGrid Visualization Cluster	ANL, USA	Intel Xeon
NCSA Visualization Cluster	NCSA	SGI Onyx

TABLE II: The main TeraGyroid visualization machines

not uncommon to examine the results of such a simulation once they return from the batch queue, only to find that a simulation has not been run for sufficient time (in which case it must be tediously resubmitted), or that it ran for too long, and the majority of the computer time was wasted on simulation of an uninteresting equilibrium system showing no dynamical behaviour.

Another unfortunate scenario often occurs when the phase diagram of a simulated system is not well known, in which case a simulation may evolve away from a situation of interest, wasting further CPU time.

Computational steering, the ability to watch and control a calculation as it runs, can be used to avoid these difficulties: a simulation which has equilibrated may be spotted and terminated, saving CPU time wastage. More powerfully, a simulation may be steered through parameter space until it is unambiguously seen to be producing interesting results: this technique is very powerful when searching for emergent phenomena, such as the formation of surfactant micelles, which are not clearly related to the underlying simulation parameters.

Steering was performed using the RealityGrid steering library. The library was built with the intention of making it possible to add steering capabilities to existing simulation codes with as few changes as possible, and in as general a manner as possible. Once the application has initialized the steering library and informed it of which parameters are to be steered, then after every timestep of the simulation, it is possible to perform tasks such as checkpointing the simulation, saving output data, stopping the simulation, or restarting from an existing checkpoint.

When a steered simulation is started, a Steering Grid Service (SGS) is also created, to represent the steerable simulation on the Grid. The SGS publishes its location to a Registry service, so that steering clients may find it. This design means that it is possible for clients to dynamically attach to and detach from running simulations.

The SGS code was implemented in Perl, and communication between clients, registries, and steered simulations, is performed using the SOAP protocol.

D. Visualization

Successful computational steering requires that the simulation operators have a good understanding of what the simulation is doing, in real time: this in turn requires good visualization capabilities. Each running simulation emits output files after certain periods of simulation time have elapsed. The period between output emission is initially determined by guessing a timescale over which the simulation will change in a substantial way; however, this period is a steerable parameter, so that the output rate can be adjusted for optimum visualization without producing an excessive amount of data.

The LB3D code itself will only emit volumetric datasets as described above; these must then be rendered into a human-comprehensible form through techniques including volume-rendering, isosurfacing, raytracing, slice planes, and Fourier transforms. The process of producing such comprehensible data from the raw datasets is itself computationally intensive, particularly if it is to be performed in real time, as required for computational steering.

For this reason, the project used separate visualization clusters, described in Table II, to render the data. Output volumes were sent using `globus_io` from the simulation machine to the remote visualization machine, so that the simulation could proceed independently of the visualization; these were then rendered using the open source VTK[34] visualization library into bitmap images, which were in turn multicast over the AccessGrid using

the Chromium[35] and FLXmitter[36] libraries, so that the state of the simulation could be viewed by scientists around the globe. In particular, this was demonstrated by performing and interacting with a simulation in front of a live worldwide audience, as part of the SCGlobal track of the SuperComputing 2004 conference.

There are many parameters for such a visualization, such as the region of the simulation being visualized, colour maps, isosurface levels, and orientation of the visualization geometry. These were controlled through SGI's VizServer software, allowing control of the geometry from remote sites.

The RealityGrid steering architecture was designed in a sufficiently general manner that visualization services can also be represented by Steering Grid Services: in order to establish a connection between the visualization process and the corresponding simulation, the simulation SGS can be found through the Registry, and then interrogated for the information required to open the link.

E.   Coordination

In order to be able to deploy the above described components as part of a usable simulation Grid, a substantial amount of coordination is necessary, so that the end user is able to launch an entire simulation pipeline, containing migratable simulation, visualization, and steering components, from a unified interface. This requires a system for keeping track of which services are available, which components are running, taking care of the checkpoints and data which are generated, and to harmonize communication between the different components.

This was achieved through the development of a Registry service, implemented using the `OGSI::Lite`[37] toolkit. The RealityGrid steering library[31], which exposes an API to which applications such as the simulation are linked, communicates with the rest of the Grid by exposing itself as a Grid Service, as shown in Figure 1. Through the Registry service, steering clients are able to find, dynamically attach to, communicate with, and detach from steering services to control a simulation or visualization process.

III.   LESSONS LEARNED

A significant amount of effort was required to make sure that all the required software was ported to and ran smoothly on the required platforms: not only the application and visualization codes, but also the libraries on which they relied. A significant problem in using such a heterogeneous Grid is that the location and invocation of compilers and libraries differ widely, even between machines of the same architecture. Environmental parameters, such as the location of temporary and permanent filespace, file retention policies, or executable paths, also varied widely. During the project, these issues were dealt

with through the use of *ad hoc* shell scripts, but this is not a satisfactory solution in general.

The TeraGyroid testbed network (Figure 2) was formed by federating separate Grids in the UK and US: this required each Grid to recognize users and certificates from other Grids. During the project, this was mostly dealt with by communication between the individuals involved, and by posting the IDs of the certificates which needed to be recognized on a Wiki; however, again, this is not a scalable long-term solution, and ideally the issue would be dealt with through use of a third-party certificate management system.

Much use was made of "dual-homed" systems, with multiple IP addresses on multiple networks. This caused problems due to the tendency of authentication systems such as SSL to confuse host identity with IP address, requiring ugly workarounds. More generally, most networking software at present assumes a homogeneous network, and delegates control of routing to much lower levels. This makes it difficult, for example, for a client process running on one host to move files between two other hosts using a specific network, in the case (as it was with the TeraGyroid project) where a high-bandwidth network has been constructed specifically for the transfer, and is to be preferred over other links.

Problems were encountered when the compute and visualization nodes were not directly connected to the Internet, but communicated through firewalls, which is a common situation on large clusters. Workarounds such as port-forwarding and process pinning were used during the project, but again do not represent good long-term solutions.

The simulation pipeline requires AccessGrid virtual venues and simulation, visualization, and storage facilities to be available simultaneously, at times when their human operators could reasonably expect to be around. This was often dealt with by manual reservation of resources by systems administrators, but the ideal solution would involve automated advance reservation and co-allocation procedures.

It was generally found that existing middleware toolkits such as Globus were rather heavyweight, requiring substantial effort and local tuning on the part of systems administrators to install and maintain, particularly due to their reliance on specific versions of custom-patched libraries. The high-level lightweight Grid Service Container `OGSI::Lite`[37] was found to be invaluable to the project, since it allowed very rapid development and hosting of high-level services such as the Registry in a way which would not have been possible on similar timescales using toolkits such as Globus. These issues are examined in closer detail in a separate document[38].

The TeraGyroid project involved collaboration between hundreds of individuals across two continents, and would not have been possible without good communication facilities. The AccessGrid teleconferencing system was used to good effect[39] to hold organizational meetings. Text-based Internet Relay Chat (IRC) proved in-
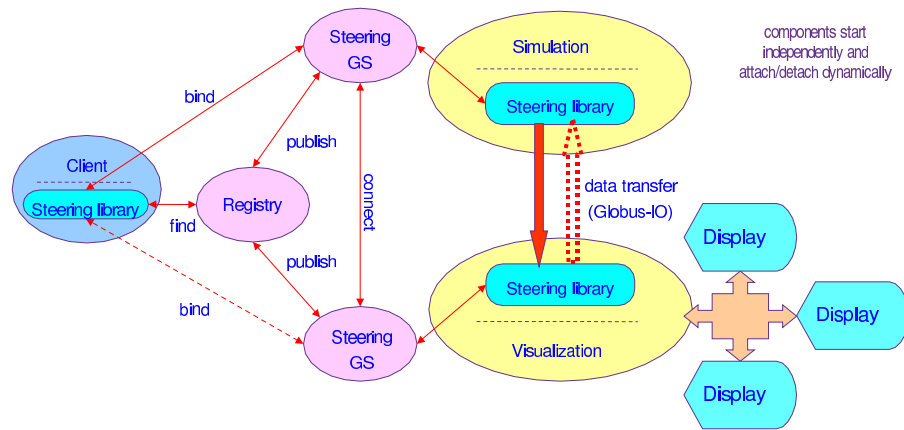
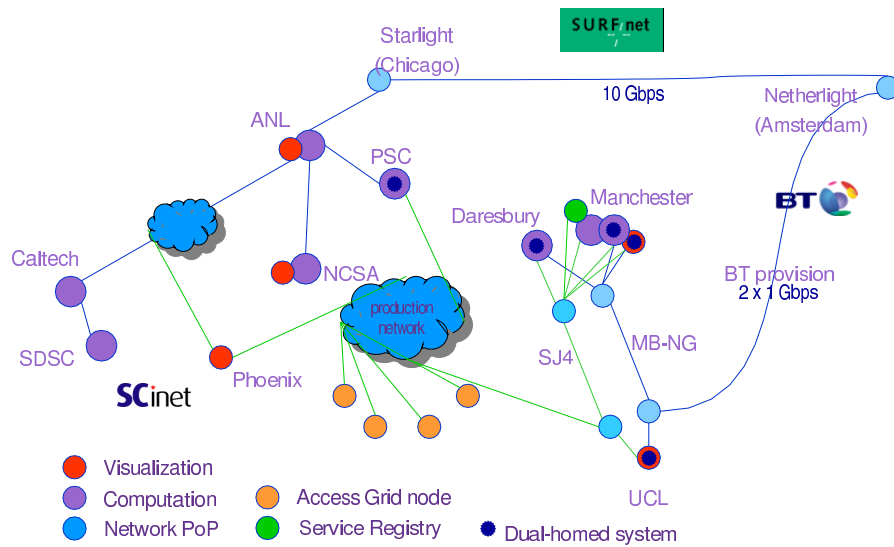FIG. 1: Grid Services in a TeraGyroid simulation

FIG. 2: The TeraGyroid testbed network

valuable as a back-channel for debugging the simulation Grid. A user-editable web site, or Wiki, was a valuable tool which acted as a combined bulletin board, configuration repository, scheduling tool, and distributed notebook.

IV. CONCLUSIONS

Simulation of defect dynamics in liquid crystal mesophases can require large amounts of computing power, not only for the calculation itself, but also for visualization and processing of the results. Computational steering can reduce wasted computer time and reduce time-to-insight.

A simulation Grid was formed to perform defect dynamics calculations, by federating US and UK-based HPC Grids through a custom high-performance network. Collaborative steering sessions with active participants on two continents and observers worldwide were made possible through this approach.

Several gaps in existing Grid tools were identified during the experiment: debugging network problems and

harmonizing authentication and authorization systems are still far from easy; portability and interoperability of the software underlying the Grid are still at an unsatisfactory level. These issues and others were dealt with at the time using quick-fix workarounds, but general solutions are much needed.

The project has produced a large amount of scientifically valuable data, which is still being analysed. Analysis of the data is itself a computationally and scientifically demanding problem: the fluid nature of the liquid crystal creates interesting problems when trying to identify defect regions[40]. We expect to publish details of morphological and dynamical analysis of the data in forthcoming papers.

V. ACKNOWLEDGEMENTS

[1] S. M. Pickles, R. J. Blake, B. M. Boghosian, J. M. Brooke, J. Chin, P. E. L. Clarke, P. V. Coveney, R. Haines, J. Harting, M. Harvey, et al., Proceedings of the Workshop on Case Studies on Grid Applications at GGF 10 (2004), URL http://www.zib.de/ggf/apps/meetings/ggf10.html.

[2] T. E. Faber, *Fluid Dynamics for Physicists* (Cambridge University Press, 1995), ISBN 0-521-42969-2.

[3] J.-P. Rivet and J. P. Boon, *Lattice Gas Hydrodynamics* (Cambridge University Press, 2001).

[4] U. Frisch, B. Hasslacher, and Y. Pomeau, Phys. Rev. Lett. **56**, 1505 (1986).

[5] D. H. Rothman and J. M. Keller, J. Stat. Phys. **52**, 1119 (1988).

[6] P. J. Love, Phil. Trans. R. Soc. Lond. A **360**, 345 (2002).

[7] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond* (Oxford University Press, 2001).

[8] G. R. McNamara and G. Zanetti, Phys. Rev. Lett. **61**, 2332 (1988).

[9] X. Shan and H. Chen, Phys. Rev. E **47**, 1815 (1993).

[10] A. Lamura, G. Gonnella, and J. M. Yeomans, Europhys. Lett. **45**, 314 (1999).

[11] H. Chen, B. M. Boghosian, P. V. Coveney, and M. Nekovee, Proc. R. Soc. Lond. A **456**, 2043 (2000).

[12] J. Chin and P. V. Coveney, Phys. Rev. E **66** (2002).

[13] P. J. Hoogerbrugge and J. M. V. A. Koelman, Europhys. Lett. **19**, 155 (1992).

[14] P. Español and P. Warren, Europhys. Lett. **30**, 191 (1995).

[15] S. Jury, P. Bladon, M. Cates, S. Krishna, M. Hagen, N. Ruddock, and P. Warren, Phys. Chem. Chem. Phys. **1**, 2051 (1999).

[16] A. Malevanets and R. Kapral, Europhys. Lett. **44**, 552 (1998).

[17] A. Malevanets and J. M. Yeomans, Europhys. Lett. **52**, 231 (2000).

[18] Y. Hashimoto, Y. Chen, and H. Ohashi, Comp. Phys. Comm. **129**, 56 (2000).

[19] T. Sakai, Y. Chen, and H. Ohashi, Comp. Phys. Comm. **129**, 75 (2000).

[20] A. L. Garcia, J. B. Bell, W. Y. Crutchfield, and B. J. Alter, J. Comp. Phys. **154**, 134 (1999).

[21] R. Delgado-Buscalioni and P. V. Coveney, Phys. Rev. E **67** (2003), URL http://link.aps.org/abstract/PRE/v67/e046704.

[22] J. M. Seddon and R. H. Templer, *Polymorphism of Lipid-Water Systems* (Elsevier Science B. V., 1995), chap. 3, pp. 97–160.

[23] J. M. Seddon and R. H. Templer, Phil. Trans.: Phys. Sci. Eng. **344**, 377 (1993).

[24] C. Czeslik and R. Winter, J. Mol. Liq. **98**, 283 (2002).

[25] T. A. Shefelbine, M. E. Vigild, M. W. Matsen, D. A. Hajduk, M. A. Hillmyer, E. L. Cussler, and F. S. Bates, J. Am. Chem. Soc. **121**, 8457 (1999).

[26] T. Landh, FEBS Lett. **369**, 13 (1995).

[27] N. González-Segredo and P. V. Coveney, Europhys. Lett. **65**, 795 (2004), URL http://arxiv.org/abs/cond-mat/0310390.

[28] P. J. F. Gandy and J. Klinowski, Chem. Phys. Lett. **321**, 363 (2000).

[29] K. Große-Brauckmann, Exp. Math. **6**, 33 (1997), URL http://www.expmath.org/restricted/6/6.1/gyr.ps.

[30] V. Z. H. Chan, J. Hoffman, V. Y. Lee, H. Iatrou, A. Avgeropoulos, N. Hadjichristidis, R. D. Miller, and E. L. Thomas, Science **286**, 1716 (1999).

[31] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter, and S. M. Pickles, Contemporary Physics **44**, 417 (2003).

[32] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M.

Pickles, R. L. Pinning, and A. R. Porter, in *Proceedings of the UK e-Science All Hands Meeting, September 2-4* (2003), URL `http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf`.

[33] P. J. Love, M. Nekovee, P. V. Coveney, J. Chin, N. González-Segredo, and J. M. R. Martin, Comp. Phys. Comm. **153**, 340 (2003), URL `http://www.arxiv.org/abs/cond-mat/0212148`.

[34] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics* (Kitware, Inc., 2003), 3rd ed., ISBN 1930934076, URL `http://www.kitware.com`.

[35] *The chromium project*, URL `http://www.sourceforge.net/projects/chromium`.

[36] *The FLXmitter SPU*, URL `http://www-unix.mcs.anl.gov/~jones/Chromium/flxspu.htm`.

[37] M. McKeown, *OGSI::Lite - an OGSI implementation in perl* (2003), URL `http://www.sve.man.ac.uk/Research/AtoZ/ILCT`.

[38] J. Chin and P. V. Coveney, *Towards tractable toolkits for the grid: a plea for lightweight, usable middleware* (2004), URL `http://www.realitygrid.org/lgpaper.html`.

[39] P. V. Coveney and M. J. Harvey, AG Focus **2**, 4 (2004), URL `http://www.chem.ucl.ac.uk/ccs/AG_Focus_Spring_04_Final.pdf`.

[40] J. Harting, M. J. Harvey, J. Chin, and P. .V.Coveney, submitted for publication.