Petascale lattice-Boltzmann studies of amphiphilic cubic liquid crystalline materials in a globally distributed high-performance computing and visualization environment

BY RADHIKA S. SAKSENA*, MARCO D. MAZZEO, STEFAN J. ZASADA AND PETER V. COVENEY*

Centre for Computational Science, Department of Chemistry, University College London, 20 Gordon Street, London WC1H 0AJ, UK

We present very large-scale rheological studies of self-assembled cubic gyroid liquid crystalline phases in ternary mixtures of oil, water and amphiphilic species performed on petascale supercomputers using the lattice-Boltzmann method. These nanomaterials have found diverse applications in materials science and biotechnology, for example, in photovoltaic devices and protein crystallization. They are increasingly gaining importance as delivery vehicles for active agents in pharmaceuticals, personal care products and food technology. In many of these applications, the self-assembled structures are subject to flows of varying strengths and we endeavour to understand their rheological response with the objective of eventually predicting it under given flow conditions. Computationally, our lattice-Boltzmann simulations of ternary fluids are inherently memory- and data-intensive. Furthermore, our interest in dynamical processes necessitates remote visualization and analysis as well as the associated transfer and storage of terabytes of time-dependent data. These simulations are distributed on a high-performance grid infrastructure using the application hosting environment; we employ a novel parallel in situ visualization approach which is particularly suited for such computations on petascale resources. We present computational and I/O performance benchmarks of our application on three different petascale systems.

Keywords: petascale computing; materials simulation; lattice-Boltzmann methods; in situ visualization; ray tracing; grid computing

1. Introduction

We present lattice-Boltzmann simulation studies of the gyroid liquid crystalline cubic mesophase on 1024³ lattice sites, making these the largest yet reported production simulations of cubic-phase rheology. Our lattice-Boltzmann simulation code, LB3D, demonstrates excellent scalability on up to 130 560 cores on the Oak Ridge National Laboratory (ORNL) Jaguar supercomputer, on 65 536 cores on the IBM BG/P at the Argonne Leadership Computing Facility (ALCF)

*Authors for correspondence (r.saksena@ucl.ac.uk; p.v.coveney@ucl.ac.uk).

One contribution of 16 to a Theme Issue ‘e-Science: past, present and future I’.
Figure 1. Skeletal view of the oil and water channels of a $64 \times 64 \times 64$ lattice-site portion of the periodic gyroid mesophase self-assembled, from a random quench, in a symmetric ternary mixture of oil, water and amphiphile simulated with periodic boundaries in all three space dimensions. The self-assembled mesophase is used as the starting configuration for the rheological simulations reported here.

called Intrepid (http://www.mcs.anl.gov/uploads/cels/files/news/2008/ALCF%20Science%20Brochure.pdf) and on 65536 cores on the TeraGrid’s Sun Constellation Cluster, Ranger (http://www.tacc.utexas.edu/resources/hpcsystems/), which are currently the first, eighth and ninth fastest supercomputers, respectively. The impressive parallel scaling performance of LB3D (Love et al. 2003) arises due to (i) the local nature of the lattice-Boltzmann algorithm as all interactions are with nearest-neighbour lattice sites and (ii) the massive computational requirement of calculating complex interactions between the multiple species present in the amphiphilic fluid mixtures which results in a low communication-to-computation ratio in the domain decomposition parallelization scheme. In this paper, we compare, contrast and compute I/O benchmarks for our application on the three different petascale systems, Jaguar, Intrepid and Ranger (for which benchmarks have been reported previously; Saksena et al. 2009). Recently, we have acquired access to the ORNL Cray XT5, Jaguar (http://www.nccs.gov/jaguar/), which is the world’s fastest supercomputer according to the current Top500 list with a theoretical peak performance of 2.33 petaflops. We report computational, I/O and \textit{in situ} visualization benchmark performance of our lattice-Boltzmann simulations on Jaguar. We also present a parallel path-tracing algorithm which is very efficient, aids visual understanding through physically based global illumination techniques and, being employed \textit{in situ}, avoids storage and transfer of terabytes of data.

Our primary scientific objective is to gain insight into defect dynamics in liquid crystalline materials (see figure 1) under shear from simulations of very large systems. Experimentally, the distribution of defects within a liquid crystal sample
Petascale lattice-Boltzmann studies

Figure 2. Skeletal visualization of the water channels in a gyroid system with multiple domains. These are shear $xz$-plane views of a $256 \times 16 \times 256$ slice of the gyroid phase at (a) $t = 0$, (b) $t = 5k$ and (c) $t = 10k$ time steps from the onset of Couette flow at shear velocity $U = 0.05$ in lattice units. The evolution of the domain wall defect texture shown here is found to be strongly correlated to the rheological response of the gyroid system.

is known to play a significant role in its rheology (Rey & Denn 2002). Large-scale lattice-Boltzmann simulations allow for the treatment of physically realistic, defect-containing, liquid crystalline systems in contrast to smaller systems where, due to finite-size effects, self-assembly of perfect crystals is obtained (Saksena & Coveney 2008). Additionally, with larger systems, we are able to simulate the low-strain-rate regions of their flow behaviour, giving us access to the rheological response close to the yield point where the liquid crystal begins to deform and flow under applied strain. The response in this regime is important in functional applications where these materials are subjected to shear flows. It is only at low strain rates that the defect dynamics couple to the characteristic time scale of the applied strain (figure 2), giving rise to a strong correlation between the domain wall defects and the stress response of the gyroid mesophase.

2. Distributed petascale resources

(a) Description of computational infrastructure

Ranger was deployed at the start of 2008 for early-user access at the Texas Advanced Computing Centre (TACC) under the US National Science Foundation’s Track 2 initiative and effectively heralded the era of petascale computational resources (http://www.nsf.gov/news/news_summ.jsp?cntn_id=111179) for open scientific research. Ranger is a Sun Blade system comprising 15744 2.3GHz AMD Opteron Barcelona quad-core processors and 123TB of RAM. It was placed ninth in the Top500 list in November 2009 with a theoretical peak of 0.58 petaflops and a high-performance Linpack (HPL) Max benchmark performance of 0.43 petaflops. It has a local disk of 31.5TB and a total shared disk of 1.73PB. A Lustre file system, running across 72 I/O servers, provides a high-speed file system for the compute nodes. Another petascale resource which we will consider here is Intrepid, an IBM BlueGene/P system located at the ALCF. It is ranked eighth in the Top500 list with a theoretical peak of 0.56 petaflops and a HPL Max performance of 0.46 petaflops. It comprises 850 MHz quad core PowerPC 450 processors with a total of 163840 cores and 80TB of RAM. It has a total storage capacity of over 6 PB which is managed by two

Phil. Trans. R. Soc. A (2010)
parallel file systems, PVFS and a fast GPFS file system. Finally, we also profile our application’s performance on the Jaguar petascale system, a hybrid XT5/XT4 system, located at Oak Ridge Leadership Computing Facility. It comprises 7832 nodes XT4 component with Opteron Budapest processors and 18688 nodes XT5 component with Opteron Istanbul processors. The Jaguar XT5 component has recently been upgraded to six-core AMD Opteron processors with 224 256 cores (http://www.nccs.gov/computing-resources/jaguar/) and a total of over 255,000 compute cores when united to the quad-core XT4 component. The aggregate memory of the combined system is 361 TB (http://www.nccs.gov/jaguar/). Additionally, each Jaguar node runs Cray’s version of SuSE Linux which minimizes interruptions to the applications running on the compute nodes and thus avoids ‘OS jitter’. Jaguar uses a Lustre-based file system called Spider which serves as a large-scale storage cluster providing 10 PB of RAID6 capacity connected to the XT4 and XT5 systems via a high-speed scalable I/O network.

All three systems have different data storage and archival systems and visualization resources collocated with compute resources. TACC has a Sun StorageTek mass storage facility called Ranch, collocated with Ranger and which uses SAM-FS for storage and retrieval (http://www.tacc.utexas.edu/userservices/user-guides/ranch-mass-storage/). As we will describe in the following sections, there is also a visualization resource called Spur which shares disk with Ranger. Intrepid has an HPSS data archive and retrieval system. Intrepid also has a collocated pre-production visualization system called Eureka which provides 3.2 TB of RAM and 111 teraflops of compute power. The system consists of 100 dual quad core servers with 200 Quadro FX5600 GPUs (http://www.alcf.anl.gov/news/media_files/alcf-resource_1209.pdf). Jaguar users at ORNL have automatic access to Lens, a 32-node Linux cluster containing quad-core 2.3 GHz AMD Opterons, 64 GB of RAM and 2 NVIDIA 8800 GTX GPUs per node (http://www.nccs.gov/computing-resources/lens/). An HPSS-based mass storage facility is also available at ORNL to store data from simulations on Jaguar.

(b) Benchmarks for large-scale lattice-Boltzmann simulations

Here we present computational scaling benchmarks of the LB3D code on Jaguar and Intrepid. We have previously reported benchmarks of our lattice-Boltzmann simulations on TACC’s Ranger machine (Saksena et al. 2009), where we found close to linearly scaling on up to 32,768 cores for a large enough computational load per core. More recently, we have been able to run benchmarks on Jaguar with the largest ternary amphiphilic fluid simulation benchmarks to date, consisting of 4080³ lattice sites simulation on 130,560 cores (the minimum core count necessary for a 4080³ lattice due to memory requirements). The site updates per second (SUPS) statistics for the computational scaling benchmarks on Jaguar and Intrepid are plotted in figure 3. On Jaguar, the computational efficiency was found to be 96 per cent on going from 65,280 cores to 130,560 cores for a 4080 × 4080 × 2040 lattice. For a smaller lattice size of 4080 × 2040 × 2040, the computational efficiency on going from 65,280 to 130,560 cores was 84 per cent, still very impressive for such high core counts. Similarly, on Intrepid, the LB3D code exhibits linear scaling on up to 65,536
cores. In summary, we have found that the parallel scaling efficiency of LB3D to be more than 80 per cent for all the benchmarks run on Jaguar and Intrepid (see figure 3).

For approximately the same system size and core count, Jaguar (10203 lattice sites system on 4080 cores) provides more than 4.5 times greater SUPS compared to Intrepid (10243 lattice sites system on 4096 cores). Similarly, Jaguar provides 1.15 times greater SUPS compared to Ranger. Additionally, Jaguar and Ranger have more memory available per core, so that simulations of equal lattice sizes can run on fewer cores on Jaguar and Ranger compared to Intrepid. The production rheological simulations reported later in this paper are for a 10243 lattice site system, the smallest system considered in these benchmarks. Intrepid’s PowerPC processors have a smaller clock speed and smaller memory per core as compared to Ranger’s and Jaguar’s Opteron processors which lead to relatively smaller SUPS versus core count on Intrepid. However, due to increased user activity on Ranger in recent times, Intrepid provides relatively faster queue turnaround for large simulations, making it an attractive platform from a usability point of view.

These computational benchmarks indicate that, with the advent of systems such as Jaguar, production simulations of ternary amphiphilic systems on 40803 lattices are now computationally feasible. For a 40803 lattice system, we approach micrometre-sized domains that correspond to the dimensions of real cubosome systems. However, other key aspects of our workflow pipeline, i.e. (i) disk I/O, (ii) simulation data management, parameter steering, and (iii) high-performance and/or interactive visualization coupled with the need for advanced reservation of these (Herculean) resources, may well prove to be bottlenecks in production runs. We explore these issues in the following sections.

(c) I/O benchmarks for large-scale lattice-Boltzmann simulations

In figure 4, we report weak-scaling I/O write benchmarks of LB3D for a ternary amphiphilic gyroid cubic-phase simulation, such that as the core count increases, the lattice size per core is kept constant. Ideal scaling corresponds to a constant elapsed time with increasing core counts. Benchmarks in figure 4a are for a constant load of 643 lattice sites per core on the Jaguar (Lustre file system), Intrepid (GPFS) and Ranger (Lustre file systems) petascale systems, while in figure 4b we compare performance of a constant load of 85 × 102 × 160 with 643 lattice sites per core on Jaguar. Here, comparison of I/O performance is made for two output modes: (i) single-file-per-core output in the XDR binary format (XDR) and (ii) output using parallel HDF5 (HDF) (http://www.hdfgroup.org/HDF5/PHDF5/).

We have compared I/O performance for the three petascale systems on up to 8192 cores. Parallel HDF5 I/O parallel efficiency drops below 80 per cent for these systems beyond 8192 cores (this should be compared to >90 per cent parallel compute efficiency on 130560 cores). Single-file-per-core XDR I/O on Jaguar provides 85 per cent parallel efficiency on up to 32640 cores, better than parallel HDF5 I/O. In the single-file-per-core I/O approach, however, management of hundreds of thousands of files for post-processing and storage is a daunting task. Additionally, as has been noted elsewhere, the scalability of this single-file-per-core strategy is limited by metadata server performance.
Figure 3. Parallel performance benchmarks for LB3D on (a) Intrepid and (b) Jaguar and Intrepid benchmarks for comparison. Performance is given in terms of lattice site updates per second (SUPS). The scaling efficiency increases for large system sizes as we go to large core counts. (Shalf et al. 2007). Indeed, on Intrepid we observe that parallel HDF5 provides superior performance compared to each core writing out its own file. The surprisingly poor performance of the single-file-per-core XDR I/O on Intrepid, as compared to parallel HDF5, is possibly caused by this metadata server overhead of dealing with a large number of files. We also observed higher jitter
Figure 4. (a) Weak-scaling I/O performance benchmarks on Intrepid, Jaguar and Ranger are given here in terms of elapsed wall-clock time versus core count. A constant elapsed time with increasing core count corresponds to ideal scaling. Benchmarks are for both single-file-per-core (XDR) and parallel HDF I/O for a constant load of $64^3$ lattice sites per core. (b) For Jaguar, we compare the $64^3$ lattice sites per core weak-scaling I/O benchmarks with a load of $85 \times 10^2 \times 160$ lattice sites per core (which is close to the largest lattice that can fit into the memory of a single core and which corresponds to a $4080^3$ lattice on 130 560 cores).
in the single-file-per-core XDR I/O compared to parallel HDF5 on Ranger or Jaguar leading to poor reproducibility in the I/O performance for the same workload.

We have investigated aspects of the write I/O performance of our application on different petascale systems with distinct parallel file systems and I/O infrastructure. The I/O performance deterioration on going to higher core counts has non-trivial implications in our large-scale materials simulations of time-dependent processes, where very frequent checkpoints are required to follow defect dynamics at sufficiently high resolution. The I/O performance introduces a serious bottleneck in performing simulations at scales that approach physical dimensions although this has become computationally feasible due to the availability of large core counts. In our particular case of a memory- and compute-intensive application, optimization strategies that need to be applied to improve parallel I/O performance will range from simple approaches, such as increasing the file transaction size by aggregating data to be written, to more complex strategies such as asynchronous I/O where non-blocking I/O operations can be used to overlap I/O and computation, thus improving the overall scaling efficiency of the simulations at high core counts (Shalf et al. 2007).

(d) Storage and visualization resources

The size of datasets accumulated from simulations in order to produce a single rheological flow curve is of the order of tens of terabytes and so we use the NCSA Mass Storage System (MSS) for data archiving. MSS is NCSA’s hierarchical archival system running UberFS backended by DISKXTENDER software (http://www.ncsa.illinois.edu/UserInfo/Data/MSS/). MSS is capable of unlimited storage capacity (http://www.ncsa.illinois.edu/UserInfo/Data/MSS/) and we have found it to be very reliable and efficient.

Visualization of the large-scale three-dimensional datasets is done via two modes. In the post-processing mode, we have used the Spur visualization resource at TACC making use of the parallel visualization software PARAVIEW (http://www.paraview.org). Spur is an 8-node Sun Visualization Cluster which contains 128 compute cores, 32 GPUs and 1 TB of aggregate memory. A critical feature of Spur is that it shares the InfiniBand interconnect and Lustre Parallel file system of Ranger, thus avoiding massive data transfers from the compute resource over the production network (http://services.tacc.utexas.edu/index.php/spur-user-guide). Despite these advantages, using Spur for routine visualization of large-scale simulation datasets is still challenging. Firstly, this is due to the finite capacity (bandwidth of 10–100Mbps) of the trans-Atlantic production network link from the UK to TACC, which makes it difficult to use Spur for interactive visualization, for example, with PARAVIEW. The importance of optical lightpaths connecting computational resources and visualization and analysis centres is obvious here and we have been involved in the lightpath network-related activities between the US TeraGrid and University College London (Coveney et al. 2010). Secondly, it is difficult to incorporate visualization on Spur within the simulation workflow deployed and managed via the application hosting environment (AHE; described in §4). This is due to the access mechanism currently in place on Spur, where a user can only invoke
PARAVIEW on Spur via a tunnelled VNC session (http://services.tacc.utexas.edu/index.php/spur-user-guide); currently, this appears to be true even for non-interactive (image generation) batch jobs.

Recently, we have also adopted another in situ rendering mode for visualization in which a parallel ray-tracing algorithm is tightly coupled to the simulator and each processor is responsible for rendering the system subdomain that it is processing within the parallel lattice-Boltzmann scheme. This approach is described in more detail in the next section. The in situ approach provides a truly parallel visualization alternative when compute and visualization resources are not collocated and allows us to optimally exploit the available compute and memory capabilities for both simulation and visualization. For example, we can circumvent the need to transfer simulation datasets from the compute to the visualization resources and even the need to produce intermediate, periodic three-dimensional checkpoints during the ‘initialization’ or ‘equilibration’ steps when the system is en route to the production stage for which a snapshot image of the lattice would suffice. The current disadvantage in adopting this approach is that in situ rendering is not as widely adopted as conventional post-processing visualization approaches and there is thus a limited choice of third-party open source in situ visualization software libraries with functionality comparable to mature conventional offerings such as PARAVIEW. Additionally, one needs to reliably make advanced reservations on the current petascale computing resources to take advantage of the capability to interact with the simulation in real time.

3. Parallel in situ path tracing

Petascale simulations demand effective ways to explore their output. We have implemented a parallel in situ path tracer which can simulate inter-reflections between multiple isosurfaces reconstructed on-the-fly during ray traversal. Here, we briefly provide a review of large-volume visualization before presenting our work and our new algorithm.

Mainstream software technologies capitalize upon advanced out-of-core techniques to effectively visualize large volumes. Approaches based on min/max kd-tree ray tracing permit one to interactively isosurface static large-scale datasets by minimizing computation associated with invisible data and by leveraging sheared-memory multi-core architectures (Friedrich & Wald 2007). Distributed computing with dynamic load balancing, parallel I/O and data compression allows reasonable isosurface extraction times (Bajaj & Zhang 2009) and GPU-accelerated interactive volume rendering of large static (Marchesin et al. 2006) or time-varying datasets (Strengert et al. 2005). Crassin et al. (2009) showed that, by further tuning out-of-CPU and GPU schemes and by enhancing GPU utilization through view-dependent rendering, occlusion culling and multi-resolution level-of-detail mechanisms, it is possible to volume render very large datasets at highly interactive frame rates on a commodity computer. Efficient out-of-GPU-core data distribution and task management schemes represent a valuable strategy to approach complex rendering such as path tracing of large sparsely distributed primitives (Budge et al. 2009).
Data visualization can also be performed by one or more compute nodes involved in the simulation while it is running so as to exploit high-performance computational resources and, at the same time, to reduce or eliminate some of the stages involved in the visualization pipeline, such as I/O, data transfer and processing prior to rendering (Ma et al. 2007). This paradigm is called *in situ* visualization and we have deployed a version of it. Our rendering method permits us to inspect large datasets at several hundred time steps while avoiding many wall clock hours for I/O and data transfer purposes, insignificantly impacting simulation performance.

(a) Rendering algorithm

In the most extreme *in situ* visualization scenario, at a user-selected time step, each processor renders its own simulated data and the final image is recovered on a node by means of collective communications and subimage merging steps (image composition; Ma et al. 2007). We have implemented a variant of this solution: basically, we cast rays through the visible processor subdomains and repetitively perform neighbour communication to migrate rays when needed, until all the primary and scattered rays either terminate or leave the simulation domain. Parallel ray-tracing algorithms of this kind were originally developed two decades ago (Kobayashi et al. 1988), but our solution incorporates full path-tracing capabilities and is the first to be directly employed within a simulation environment. It is worth noting that, in contrast to previous *in situ* visualization strategies, we can integrate any illumination condition at the expense of some extra computation and communication associated with ray communication and processing termination criteria; here, however, we have supplemented the isosurfacing and ray communication protocols with memory-cheap algorithms tailored for the minimization of the impact of our path tracer on LB3D requirements.

(i) Path-tracing kernel and data structures

Each processor directly operates on the data available from LB3D for isosurfacing purposes. These data are composed of spatial subdomains plus the halo site layer surrounding it which form a single Cartesian grid of \((M + 2)^3\) points (for simplicity, we assume that the subdomain is cubic). Furthermore, since rays are allowed to migrate to the adjacent subdomains, we need to maintain \(6 \times 2\) arrays for the ray data to send and receive, and another (extra) stack to approach push and pop mechanisms: if a ray leaves the current subdomain to migrate to another, its data are pushed to the corresponding send-buffer while data of a ray coming from another processor (written to the receive-buffer after ray data communication) are stored in the extra stack; notably, the latter becomes a source of ray-tasks for the owner processor.

After partitioning parameters retrieved from LB3D and allocating ray-buffers, each processor casts only those rays passing through the clipped screen-based minimum rectangular projection of its subdomain bounding box and starts tracing the rays (if any) whose intersection with the total bounding box does not belong to other processors. Rays are iteratively traced and scattered, while a one-dimensional pixel-wise colour buffer stores the light contribution.
corresponding to the spanned paths; pre-allocated coloured pixels are avoided through two-dimensional screen-based lookup tables. Light integral computation accounts for situations in which ray paths traverse multiple subdomains or correspond to the same pixel (the algorithmic description of light integration mechanisms is lengthy and will be discussed elsewhere). We do not allocate more memory than that consumed by the buffers cited above, which amounts to only a few tens of megabytes; therefore, we do not significantly impact memory usage.

(ii) Ray scheduling protocol

Path tracing originates within those subdomains directly visible from the viewpoint. If there are several visible voxels, rays per pixel and multiple allowed ray/surface bounces, the primary rays and the ray data to communicate and to be processed by adjacent processors is likely to be substantial. A similar situation may occur if rays converge to a small volume region. To avoid large ray-buffer reallocations, we break path tracing in a number of steps, called syncro-steps, if multiple rays are cast per pixel and multiple bounces are allowed. In every syncro-step, each processor traces rays until a user-defined time threshold $t$ is reached or no ray is locally scheduled. We process MaxRays rays consecutively to mitigate scheduling overhead. Ray data are exchanged by means of blocking point-to-point communications. If the number of rays per pixel is low, we exhaustively enumerate the rays that need to be traversed; in this case, the stage occurring before neighbour communication is called exhaust-step. Path tracing terminates when no ray is queued. This involves an all-to-all (MPI_Allreduce) communication which we call every $T$ syncro-steps or exhaust-steps to alleviate its cost.

(iii) Ray traversal, on-the-fly isosurface extraction and triangle caching

Isosurface ray tracing usually avoids surface extraction by intersecting the ray with the equation provided by the eight vertices of the voxels and by shading any hit with the tri-linearly interpolated normal (Friedrich & Wald 2007). To intersect a ray with the voxel of coordinates $(M, M, M)$ we need the lattice site of coordinates $(M + 1, M + 1, M + 1)$, but a central difference calculation of the normal at this location requires the data of the site at coordinates $(M + 1, M + 1, M + 2)$ which is not stored by LB3D within the current subdomain. We do not communicate the necessary extra halo layer of sites prior to path tracing but we instead extract a triangulation within the reference voxel on the fly by using the marching tetrahedra technique which guarantees topological correctness. We intersect the ray with the generated surface (if any) and the computed normal is that pertaining to the triangle which contains the closest hit. A small voxel-wise triangle cache is maintained to avoid repetitive isosurface extractions such as those occurring when a ray bounces; for a similar reason, another triangle cache is kept to accelerate shadowing. Moreover, the two caches share data whenever possible. Ray-voxel traversal is accelerated by the algorithmic optimizations previously described by us (Mazzeo et al. 2010).
Figure 5. (a) Snapshot (three-dimensional projection) of our $960 \times 1088 \times 960$ gyroid mesophase rendered with our parallel \textit{in situ} path tracer. (b) Magnified slice of the $xz$-plane of a self-assembled $1024^3$ gyroid system.

(iv) \textbf{Image compositing}

When no more ray is left to traverse, every processor generally holds a subimage in the form of coloured points which sparsely correspond to pixels on the screen. We adopt the binary communication pattern presented previously by us (Mazzeo \textit{et al.} 2010) to assemble the subimages and produce the final one.

\textbf{(b) Benchmarks}

Here, we present some preliminary performance results for our parallel \textit{in situ} path tracer. The code is written in C, while LB3D is in Fortran90, the latter calling functions in the former through an appropriate interface. The path tracer was tested on Jaguar and Intrepid by using models comprising $960 \times 1088 \times 960$ and $1024^3$ sites, respectively, at a screen resolution of $2048^2$ pixels. At the current stage of development, the ray-tracing protocol can approach multiple opaque, semi-transparent or specular isosurfaces but we focused interest on the surface with isovalue equal to 0.4 only, that is, assumed to be Lambertian (completely diffusive).

We tested two different visualization modes. In the first one, called RT, we carried out simple rendering by tracing one ray per pixel and by assuming one light located at the viewpoint; in this visualization mode, we did not allow any shadow or secondary rays. In the second mode, called PT, we employed 64 eight-bounce paths per pixel and we modelled the illumination through white ambient light and inter-reflection capabilities. In both rendering modes, $T$ was set equal to 100, while $\tau$ and $\text{MaxRays}$ were 0.01s and 100, respectively. Figure 5 shows the system of $960 \times 1088 \times 960$ and $1024^3$ sites visualized by means of the second approach. The rendering performance is reported in table 1, which also shows the time needed by LB3D to execute one time step.

RT takes a small fraction of the time needed for a simulation time step, while PT is an order of magnitude more demanding. Assuming that the simulation requires $10^6$ time steps and one wants to visualize $10^5$ snapshots, the times spent by RT and PT are, respectively, about two and four orders of magnitude less than that of LB3D.
Table 1. Simulation step-time and visualization performance (in seconds) needed by LB3D and our parallel *in situ* path tracer, respectively (see text for details).

<table>
<thead>
<tr>
<th></th>
<th>Jaguar visualization</th>
<th>Jaguar LB3D</th>
<th>Intrepid visualization</th>
<th>Intrepid LB3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT</td>
<td>0.68</td>
<td>5.3</td>
<td>1.0</td>
<td>20</td>
</tr>
<tr>
<td>PT</td>
<td>38</td>
<td>5.3</td>
<td>140</td>
<td>20</td>
</tr>
</tbody>
</table>

(c) Future work

We have successfully met our goal to make the visualization of our large-scale systems practical by avoiding lengthy I/O and by reducing the data transfer to a series of images only. We are able to aid visual perception with full path-tracing capabilities and to do so via petascale machines thereby overcoming the limitations of the hardware and software technologies available on local workstations.

We have not explored the performance behaviour of the path tracer as a function of the parameters \( T \), \( \tau \) and \( \text{MaxRays} \) which might influence load and communication balancing. Our isosurfacing algorithm requires an insignificant memory footprint but triangle caches larger than the current ones can improve performance drastically. When multiple rays per pixel must be generated, it may be profitable to reorganize the subdomains prior to rendering so as to improve load balancing, for example, in a shuffled fashion as suggested by Mazzeo *et al.* (2010) and Kobayashi *et al.* (1988).

The main limitation of the current software version is its inability to render or display interior regions. We intend to provide these capabilities in the future. Ultimately interactive steering (Mazzeo *et al.* 2010) and feature extraction will be included which can dramatically aid scientific discovery.

4. Application hosting environment

The growing power and number of high-performance computing resources made available through computational grids present major opportunities as well as challenges to the application scientist. At issue is how these resources can be accessed and how their power can be effectively exploited. We have developed the AHE (Coveney *et al.* 2007; Zasada & Coveney 2009) to address these problems, by virtualizing access to grid-based applications and presenting users with a simple set of services with which they interact in order to launch and monitor grid applications. The latest release, AHE 2.0, provides access to a common platform of federated computational grid resources in standard and non-standard ways. This means that AHE gives the user *inter alia* the ability to interoperate seamlessly across different grids running a variety of different middleware stacks, to launch and steer simulations using the RealityGrid steering system (Coveney *et al.* 2007; Zasada & Coveney 2009) and to reserve time on resources in advance using the highly available robust co-scheduler (HARC; MacLaren *et al.* 2006). HARC allows...
Figure 6. (a) The layered architecture of AHE. The AHE client, running on a user’s desktop machine, interacts with the AHE services running in a Tomcat Web services container and also stages data via GridFTP or WebDAV. The AHE services interact with job submission components such as OGSA-BES or Globus GRAM in order to launch applications on distributed grid resources. (b) AHE features graphical and command line clients which can interoperate, meaning users can launch simulations via the command line, then monitor them and retrieve the output data using the desktop GUI client.

for time on multiple computational resources to be reserved in advance, meaning that real-time visualization of LB3D—which requires the user to know when the resource will become available—becomes possible.

The layered architecture of AHE is shown in figure 6. Full details of AHE architecture are given in Zasada et al. (2006). The AHE server is configured by an expert user within a community, with the server storing details of how to launch a particular scientific code. Users can then interact with the simple AHE services to launch and monitor applications, without having to worry about the details of how the application is run. AHE features simple command line and graphical desktop interfaces (shown in figure 6), to run applications on resources provided by national and international computational resources, in addition to local departmental and institutional clusters, while hiding from the user the details of the underlying middleware in use by the computational resource. AHE is able to manage applications interoperably on resources running both the UNICORE and Globus middlewares.

As noted above, our scientific simulations require access to some of the world’s biggest supercomputer class resources and involve marshalling large datasets to and from simulations. We use AHE to facilitate our simulation work in four ways: (i) to act as a single interface from which to launch simulations on a range of different machines, (ii) to steer parameters of interest in initial exploratory simulations, (iii) to marshal data between computational, storage and visualization resources, and (iv) to provide provenance information, by archiving simulation setup and results alongside input parameters so that such details can easily be retrieved.

AHE’s integration with workflow tools, and its ability to run applications comprising multiple computational components, has given us the ability to automate much of our current simulation workflow. The features of AHE address many of the needs of the computational scientist, especially when tasked with...
trying to successfully exploit large petascale resources. Specifically, AHE allows simulation input and output data to be staged to and from arbitrary locations. This means that large-scale simulations launched by users can retrieve data from, and store data to, large-scale data-storage facilities, such as those provided by TeraGrid. Data are transferred using the GridFTP protocol, meaning the data-storage facilities must be running a GridFTP server.

AHE monitors all simulations as they run, keeping a record of the location of the simulation output data, meaning that users can simply leave their simulations running, with AHE monitoring them, and then use AHE client tools to retrieve their output from the data-storage facility as needed.

5. Simulations

Here, we present simulations of the liquid crystalline gyroid phase which exhibits non-Newtonian properties, such as shear thinning, when subjected to shear forces. This interesting phenomenon is observed widely but not easily controlled or modelled. Non-Newtonian complex fluids behave like solids at certain strain rates, while at other strain rates flow like viscous liquids. Ketchup and paint
are common examples of complex fluid mixtures that exhibit non-Newtonian behaviour. We report the evolution of the defect texture and variation of the stress response in the shear plane for a $1024^3$ lattice-sites system sheared at a velocity $U = 0.1$. Couette flow is applied using the Lees–Edwards boundary condition with the shear velocity applied along the $z$-direction and shear gradient along the $x$-direction. In figure 7, we show a $256 \times 56 \times 256$ slice of the system (where the lattice dimensions vary as $0 \leq x < 256$, $0 \leq y < 56$ and $0 \leq z < 256$). The images show progressive shear deformation, initiated at the $x$ boundaries of the system where the magnitude of the shear velocity is highest. The advantage of simulating $1024^3$ lattice-sites systems is manifested here. For smaller $256^3$ systems, the correlation between the defect texture and the stress response of the system completely disappears by 5000 time steps at the lowest shear studied (Saksena & Coveney 2009), while for the $1024^3$ lattice system reported here we are able to access a much lower shear rate ($\dot{\gamma} = 2U/1024$) and the correlation between the defects and stress response persists beyond 5000 time steps as is evident in figure 7d; indeed some correlation is still evident at 10000 time steps after shear start-up (see figure 7c). This allows the study of defect dynamics under shear to unravel the correlation between defect and stress response over long times. We plan to publish our detailed scientific findings, based on the distributed computational infrastructure described here, in the near future (Saksena & Coveney in preparation).

6. Conclusions

The mesoscale lattice-Boltzmann method allows us to investigate multi-scale phenomena in liquid crystalline phases in which microscopic interfacial dynamics governed by complex fluid interactions couple to externally imposed hydrodynamic flow fields and mesoscopic defect features, producing a macroscopic non-Newtonian rheological response. We use a scalable lattice-Boltzmann code called LB3D, highly optimized in situ visualization software and well-designed middleware (AHE 2.0) in the work reported here. This software infrastructure, combined with access to petascale supercomputers as well as large-scale storage and visualization resources, has enabled us to realistically simulate and analyse multi-scale dynamical processes in these novel nanomaterials at a scale and resolution not feasible hitherto.

We are grateful for funding from EPSRC grants EP/C536452/1, EP/E045111/1 and a US NSF LRAC computational grant (charge nos. TG-MCA08X031T and TG-ASC090009) on the TeraGrid. This work was also supported by a DOE INCITE grant entitled ‘Large scale condensed matter and fluid dynamics simulations’ at ALCF funded by the US Department of Energy under contract DE-AC02-06CH11357. We are also grateful for Director’s discretionary time (account code CPH0003) on Jaguar, and to technical support staff at TACC, ALCF, ORNL and TeraGrid.

References


Saksena, R. S. & Coveney, P. V. In preparation. Defect texture dynamics under Couette flow in amphiphilic liquid crystals from large-scale lattice-Boltzmann simulations.


