

Steering UNICORE Applications with VISIT

BY THOMAS EICKERMANN, WOLFGANG FRINGS, PAUL GIBBON, LIDIA
KIRTCHAKOVA, DANIEL MALLMANN, ANKE VISSER

*Forschungszentrum Jülich GmbH, John von Neumann Institute for Computing,
D-52425 Jülich, Germany*

The UNICORE software provides a grid infrastructure together with a computing portal for engineers and scientists to access supercomputer centers from anywhere on the Internet. While UNICORE is primarily designed for the submission and control of batch jobs it is also feasible to establish an online connection between an application and the UNICORE user-client. This opens up the possibility of performing online-visualization and computational steering of applications under UNICORE control while maintaining the security provided by this system. This contribution describes the design of a steering extension to UNICORE based on the steering toolkit VISIT. VISIT is a light-weight library that supports bi-directional data exchange between visualizations and parallel applications. As an example application, a parallel simulation of a laser-plasma interaction that can be steered by an AVS/Express application, is presented.

Keywords: Computational Steering, visualisation, Grid Computing

1. Introduction

Grid computing (Foster & Kesselman 1998, Foster *et al.* 2001, Foster *et al.* 2002) is the new paradigm in wide-area distributed computing. Grids provide pervasive and seamless access to distributed computing resources across multiple administrative domains. They enable sharing of such resources and collaboration in virtual organisations. UNICORE, which was developed in several German and European projects (Erwin 2002, www.unicore.de, www.eurogrid.org) is one Grid system that aims to implement that vision. In contrast to e.g. Globus (www.globus.org) the most widely deployed Grid software today, UNICORE is not a toolbox that supports the development of Grid-enabled applications but follows a vertically integrated approach. It offers seamless and secure access to distributed computing resources, with a focus on workflow management of batch jobs. Besides its intuitive user interface, an advantage of UNICORE is that it does not require any modifications of the applications that run under its control. UNICORE is the preferred access to the HPC systems of the John von Neumann Institute for Computing. It has been selected as the Grid middleware for the new Japanese National Research Grid Initiative NAREGI (Furunishi 2003) as well as the Distributed European Infrastructure for Supercomputing Applications (www.deisa.org).

A serious limitation of the current UNICORE system is that it does not support concurrent use of distributed resources, as needed e.g. for online visualisation and

computational steering, features that are usually considered important aspects of Grids. This paper presents the design and a prototype implementation of an extension to UNICORE that supports computational steering. It is based on the steering toolkit VISIT (Eickermann & Frings 2000), a light-weight and minimally invasive library that has been developed at the Research Centre Jülich. With VISIT, simulations can be instrumented to dynamically attach to and detach from visualisations and to bi-directionally exchange data with just a few function calls. The VISIT extension to UNICORE has been designed to preserve the following strengths of UNICORE:

- single sign-on with strong authentication and encryption,
- firewall-friendliness; handling of all communication over a single fixed TCP server-port,
- immediate portability; any application that uses VISIT will be able to use the VISIT-UNICORE extension without modifications.

The software described in this contribution has been demonstrated successfully at the SC Global 2003, a globally distributed conference based on Access Grid technology, which provides a collection of resources to support group-to-group collaboration over the Grid (www.accessgrid.org). The organisation of the rest of this contribution is as follows. The next sections give a brief overview of UNICORE and VISIT, followed by a description of the VISIT-UNICORE extension. The presentation will finish with a short demonstration of the monitoring and steering capabilities of the system using a new plasma simulation code (PEPC) as an example application. This application and future developments are described in the last sections of the paper.

2. The UNICORE Grid system

UNICORE (UNiform Interface to COmputing REsources) provides a Grid software which combines resources of supercomputer centres and makes them available through the Internet. The UNICORE Grid system emphasizes strong authentication and data security relying on X.509 certificates and SSL. The security is enforced in a consistent and transparent manner, and the differences between platforms are hidden from the user thus creating a seamless HPC portal for accessing supercomputers. The UNICORE Grid system does not make any assumptions on the administrative and technical environment of a supercomputer centre and can therefore easily be deployed. The organisation decides which virtual organisation they will belong to and which resources they will provide. The UNICORE Grid system lets the user prepare jobs through the UNICORE client on a Unix workstation or a Windows PC. This client is a Java application with a graphical user interface.

Via a specified and documented API, application-specific plugins can be implemented that further simplify job creation for the user. Various plugins have been developed for applications mainly in the areas of computational chemistry, computational fluid dynamics and structural mechanics. After preparation, jobs can be submitted to any target system of the UNICORE Grid that satisfies the user resource requirements. The user can control the jobs through the job monitor of the client. A UNICORE job contains a number of tasks which may have

interdependencies. In dividing a UNICORE job into sub-jobs, a hierarchical job structure can be created. Different sub-jobs can be executed on different target systems within the UNICORE Grid. UNICORE jobs are stored in an abstract form and resources are specified in abstract units; thus a seamless environment is provided. The UNICORE servers convert the abstract jobs and resource requests into target system specific commands and options. Upon user request, the input and output files are transferred between the file space used by the job and the local file system of the UNICORE client. To transfer data between different sites, the UNICORE servers choose the most efficient transfer mechanism that is supported by both target systems. The UNICORE software and many extensions are available as Open Source under BSD license (unicore.sourceforge.net). The Central Institute for Applied Mathematics of the Research Centre Jülich supports academic users of UNICORE and coordinates the future development and deployment of the software.

The UNICORE GRID system consists of three distinct software tiers:

- UNICORE client interacting with the user and providing functions to construct, submit and control the execution of computational jobs,
- UNICORE servers that are divided into gateways acting as point-of-entry into the protected domains of the HPC centres, and Network Job Supervisors (NJSs) that adapt the abstract UNICORE job for the specific HPC system,
- UNICORE Target Systems that schedule and run the jobs on the HPC platforms.

The UNICORE client connects to a UNICORE gateway of a member belonging to the virtual organisation forming the UNICORE Grid (called a *Usite* in Fig. 1). The UNICORE gateway checks the user's membership in the virtual organisation, before contacting the UNICORE Network Job Supervisor (NJS) that controls the target system on which the job (or sub-job) will be run. The UNICORE NJS manages the submitted UNICORE jobs and converts the abstract jobs into batch jobs on the target system and runs them on the native batch subsystem. Status information and job output are retained by the NJS until the client requests the transfer and deletes the job. The NJS contains a platform-specific incarnation database that adapts to the multitude of current HPC systems. All other components are designed for portability. The UNICORE client and server are implemented in Java. The Target System Interface (TSI) on the HPC system is available as a Java application and alternatively a set of Perl scripts. The protocol between the components is defined using Java mechanisms. A low-level layer called the UNICORE Protocol Layer (UPL) handles authentication, SSL communication and transfer of data as inlined byte-streams. A high-level layer, the Abstract Job Object (AJO) library, contains the classes to define UNICORE jobs, tasks and resource requests. The UPL is designed as a protocol that transfers data regardless of its form. The AJO is the specification of the work that a UNICORE user wants to do on a UNICORE target system. It is generated by the UNICORE client. One aim of the AJO was to allow seamless specification of jobs. The specification of the work stays the same and is independent of different site and authorisation policies or different executables and options. Third-party components can be integrated into the system on top

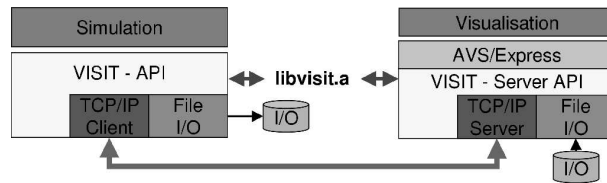


Figure 2. The basic architecture of VISIT. Simulation and visualisation using VISIT communicate via TCP sockets. Additionally, simulation results can be written to a file for later visualisation.

gabit Science Network, the G-WiN. Since then, VISIT has evolved into a stable software used in several application projects, mainly on the Supercomputers of the John von Neumann Institute for Computing (NIC) at Jülich. A main design goal of VISIT was to minimize the load on the steered simulation and to prevent failures or slow operation of the visualisation from disturbing the simulation progress. This means that all operations (like opening a connection, sending data to be visualized or receiving new parameters) have to be initiated by the simulation and are guaranteed to complete (or fail) after a user-specified timeout. This led to the design decision to implement VISIT as a simple client-server application where the visualisation acts as a server that dispatches the simulations requests - unlike many other steering toolkits that work the opposite way (Kohl & Papadopoulos 1998, Liere *et al.* 1997, Johnson *et al.* 2000, Rantzau *et al.* 1998, Brooke *et al.* 2003a). To keep VISIT portable to 'classic supercomputers' which often lack good implementations of common UNIX protocols or tools, the simulation side of VISIT in particular does not rely on any external software or special environment and has a lean and easy-to-use interface.

The current version of VISIT uses TCP/IP sockets for the connection between simulation and visualisation (Fig. 2). In order to connect to the server (visualisation) the client (simulation) needs to know address and port number of the server. This is accomplished by a simple directory service named SEAP (Service Announcement Protocol). The visualisation can register its address under a service name at the SEAP-server. The simulation can query this information (consisting of a interface-name and a port-number) and use it to connect to the visualisation. This service allows the user to dynamically attach a visualisation to the simulation without having to specify its location in advance. Of course, when integrating VISIT with a Grid system, this home-grown directory-service is replaced with the corresponding service offered by that system.

VISIT uses an MPI-like data transport mechanism based on messages that are distinguished via tags to transfer simple data types like strings, integers, floats, user defined structures, and arrays of these. The client either sends data along with a header describing its content or requests data from the server by sending a header that describes what is requested. Any data conversions (byte order, precision, integer-float) are performed transparently by the server, again so that the simulation is disturbed as little as possible. The client (simulation) part of VISIT has C, Fortran, and Perl language bindings. Fig. 3 gives a code example for the client API. On the server (visualisation) side VISIT supports C, Perl and AVS/Express, a commercial visualization software. Bindings for Java, IDL (Interactive Data Language) and vtk are currently being developed. VISIT is freely available for download

```

/* connect to a visualisation */
vcd = visit_connect_seap(service, passwd, 0, 1, msg_timeout, conn_timeout);

/* send a 3D-double array data[10][20][30] */
visit_send(vcd, itag, timestamp, data, VISIT_DOUBLE, 3, 30, 20, 10);

/* receive a string with at most 100 characters */
vtype = VISIT_STRING; size = 100;
visit_recv(vcd, itag, &timestamp, sdata, &vtype, &size);

/* close the connection to the visualisation */
visit_disconnect(vcd);

```

Figure 3. example C-code for a simulation exchanging information with a visualisation.

(www.fz-juelich.de/zam/visit) in source-form for common UNIX platforms and Microsoft Windows.

A major drawback of VISIT is that it does not provide any encryption or other means of security except for a connection password that is transferred in clear-text. Due to its dynamic TCP-port selection scheme it also does not work well with firewalls. These problems are resolved by the integration of VISIT with UNICORE which is described in the next section, since UNICORE offers exactly the features needed.

4. Accessing UNICORE jobs with VISIT

The main technical challenge when trying to steer VISIT applications running under control of UNICORE is to map the different communication models of these two systems correctly. In UNICORE, the user uses a client to submit jobs, query their status and eventually fetch the results. All these operations are separate transactions that do not require a stateful connection between the UNICORE client and the target system to be maintained. While this contributes to the robustness of the UNICORE system (a client can appear or vanish at any time) it does not match the connection-oriented architecture of VISIT well, where the steered application is the client and the visualisation the server. To overcome that we have designed and implemented a stateful tunnel for VISIT connections through the UNICORE system. The simulation-end of that tunnel is formed by VISIT and SEAP proxy-servers which are separate processes running on each target system. The other end of the tunnel is located at the UNICORE client, implemented as a client-plugin. By polling the target system for new data, that plugin is able to emulate the server capabilities that are required for the VISIT tunnel. Like the Shepherd, the SEAP proxy-server is a single permanent process with an instance running for each UNICORE target system. Before attaching a visualisation to a UNICORE job the user has to provide the SEAP-Service name of that visualisation into the UNICORE client. The client queries the SEAP-Server for the address of the visualisation and establishes a standard VISIT connection to it. After that, the client creates a special AJO that informs the SEAP proxy-server on the target system about this new connection. The SEAP proxy will in turn start a VISIT proxy-server that serves as a permanent end-point of the VISIT tunnel for the connection. Every VISIT send- or receive-request of the simulation is mapped to a single UNICORE task

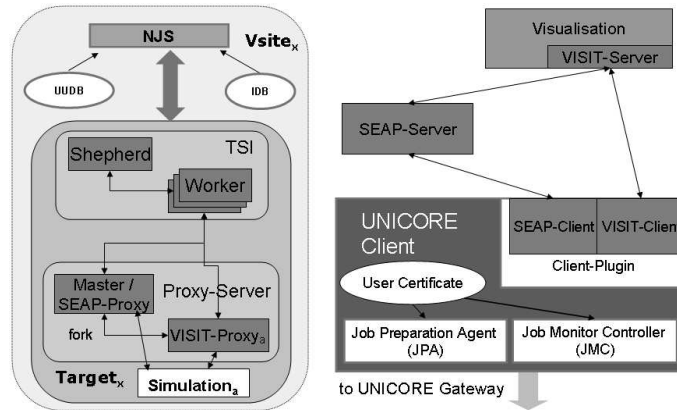


Figure 4. Architecture of the VISIT extension to UNICORE.

(AJO) that transfers data between the UNICORE client and the VISIT proxy-server. Since each task may be assigned to a different TSI worker, this implies that for each task a new connection between a worker and the VISIT proxy-server needs to be established and that the VISIT proxy-server and the UNICORE client are responsible for maintaining the state of their connection. Although the TSI is extended in such a way that all of the communication takes place via sockets and no intermediate files have to be created, this implementation of a stateful connection imposes a significant overhead. In typical computer and network environments a message delay of about 2-3 sec is added to the normal VISIT and TCP message delay of $< 100 \mu\text{sec}$. Throughput is limited to about 10 Mbit/s. The benefit gained by this overhead is that the communication between application and visualisation is completely handled by UNICORE, and is thus secured by encryption and strong authentication and does not require any additional ports to be opened on a firewall. The performance will be improved significantly in a new version that is currently under development within the EU-funded project UniGrids (www.unigrids.org). Here, the UNICORE services will mainly be used for signaling and control of the connection between simulation and visualisation. The actual data-transfer will be handled by a separate encrypted channel.

To allow multiple users to collaboratively view and steer a simulation, the simulation data has to be sent to all visualisation applications and these applications have to exchange information among each other to ensure that everyone has the same view of the data (e.g. position and orientation of view point or parameters like thresholds that influence the visualisation). Since the simulation acts as a client in VISIT, the former task can easily be implemented by a 'multiplexer' that sends all VISIT send-requests to all participating visualisations, ensuring that everyone views the same data. Receive-requests are only sent to a 'master' visualisation, so that only that master is able to actively steer the application. The master-role can be moved between the simulations allowing for a coordinated cooperative steering. This functionality has been implemented in an application (the vbroker) that is part of the standard VISIT distribution. For the VISIT-UNICORE extension this functionality has been moved into the VISIT proxy-server running on the UNICORE target system. This has the advantage that all users participating in the collaboration have to authenticate to the UNICORE system.

Like video and audio, the exchange of control information between the visualisations is sensitive to latency if a 'sense of presence' is to be created among the users. Therefore we do currently not use UNICORE communication mechanisms for that purpose. Instead, we have implemented an external server that collects and redistributes the control data. This server allows to assign different roles to the participants: one role allows to change visualisation parameters like the view angle and a second role is just for passive viewers.

5. An example application: plasma simulation code PEPC

As an example of an application that greatly benefits from using VISIT, we give a brief overview of PEPC (Parallel Electrostatic Plasma Coulomb-solver), a new class of plasma simulation code recently developed in Jülich (Gibbon 2003, 2004). PEPC has been instrumented to be coupled with VISIT independently of the VISIT-UNICORE extension and does not necessarily require UNICORE for its operation. The code kernel uses a Barnes-Hut hierarchical tree algorithm (Barnes & Hut 1986) to perform potential and force summation for charged particles in a time $O(N \log N)$, allowing the self-consistent dynamics of several million particles to be followed on length- and time-scales normally possible only with particle-in-cell (mesh-based) techniques (Birdsall & Langdon 1985).

The VISIT interface was integrated at a very early stage in the development of PEPC to assist in verifying the correctness of the tree structure and domain decomposition for geometrically complex systems: for example, a cylindrical particle beam striking a spherical plasma target. Data transport is managed by regularly shipping both particle data-space (coordinates, velocities, charge, processor number and tracking-label) plus information on the tree structure, at present consisting of a set of node coordinates representing each processor domain. Particles are displayed as points or spheres and vectors, including time-histories over several timesteps to enable visual, selective particle-tracking; tree domains as transparent or solid boxes, providing immediate insight into both the physical and algorithmic workings of the parallel tree code. The current version also provides selected diagnostic quantities mapped onto a user-defined mesh, such as charge densities and force fields.

This code, in tandem with the VISIT-UNICORE extension, formed the basis of a live session of the SC-Global 2003 Showcase (Brooke *et al.* 2003b) The specific example used for the demonstration was a 3-dimensional (3D) simulation of the 'Laser Wakefield Accelerator' (LWFA), an advanced scheme for producing high-energy electron beams with short short pulse, high-intensity lasers (Tajima & Dawson 1979). The laser pulse (on the right of the scene in Fig. 5), generates a large amplitude potential in its wake, which proceeds to trap and accelerate a small fraction of the plasma electrons to multi-MeV energies over distances of a few millimetres (as opposed to the 10s of metres required for conventional accelerator devices). Steering was also included in this example to give user control over the laser parameters (intensity, duration and width), providing novel 'tutorial' insight into the complex, 3D aspects of the wakefield structure and particle acceleration dynamics, and their dependence on the laser parameters. In the example in Fig. 5 only 200000 particles were used in order that the code could run fast enough to allow 'real-time' user interaction. Despite this, the wave structure – in the form of electron density surfaces – can be clearly made out. The combined system thus

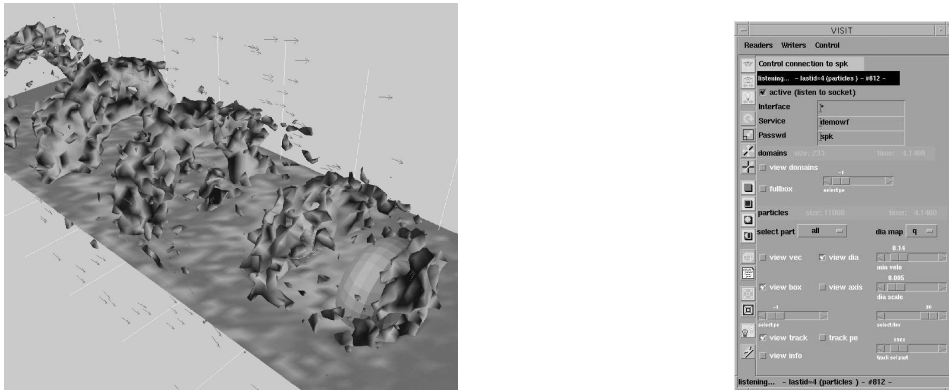


Figure 5. Snapshot of the VISIT-based online-visualisation of a Laser Wakefield simulation using PEPC; a 3D plasma wave excited by a high-intensity laser pulse (left), with corresponding application control panel on the right.

offers a unique, highly portable tool for planning and analyzing contemporary international LWF experiments (Mangles *et al.* 2004).

6. Related work and future developments

There are several activities related to online visualisation and computational steering of simulations in a UNICORE grid. Within the EUROGRID project, the Paralab has implemented an interactive extension to UNICORE. Their implementation allows to open an interactive shell on a UNICORE target system. Our VISIT-UNICORE interface has partly been inspired by their approach. Within the UNICORE Plus project, a dedicated UNICORE client-plugin has been developed that allows to view the progress of a CPMD simulation. Here, the application creates a movie-file that is continuously transferred to the client and displayed.

As part of the UK eScience initiative, RealityGrid has developed a well recognized steering library (Brooke *et al.* 2003a). It offers a similar functionality to VISIT, by providing dynamic attachment of a steering client to a simulation, exchanging parameters between the simulation and a steering client and emitting data to a visualisation component. The main difference is that the RealityGrid software contains a generic steering client and explicitly separates steering from visualisation, while most applications that use the VISIT communication primitives combine steering and visualisation in a single program. Both RealityGrid and VISIT are heading towards the utilisation of Web-Services but take a different path. While RealityGrid has been extended (Chin *et al.* 2003) to use OGSi (Tuecke *et al.* 2003) VISIT has been integrated into the current production version of UNICORE.

The UNICORE system development is moving towards Web Services. As part of that work, which is mainly performed within the UniGrids project, a version of VISIT will be developed that integrates more tightly in the new UNICORE system. We expect that the use of Web Services will significantly simplify our system and allow us to shortcut the rather complex communication chain between simulation and visualisation leading to further performance improvements and allowing for a closer integration of the collaborative features of VISIT into UNICORE.

References

- Barnes, J. and Hut, P. 1986 *A hierarchical $O(N \log N)$ force-calculation algorithm*. *Nature* **324**, 446–449.
- Birdsall, C. K., and Langdon, A. B. 1985 *Plasma Physics via Computer Simulation*. McGraw-Hill, New York.
- Brooke, J.M., Coveney, P.V., Harting, J., Jha, S., Pickles, S.M., Pinning, R.L., and Porter, A.R. 2003a Computational Steering in RealityGrid *Proceedings of the UK e-Science All Hands Meeting*.
- Brooke, J., Eickermann, T. & Woessner, U. 2003b Application Steering in a Collaborative Environment. In *Proceedings of the ACM/IEEE SC 2003 Conference, Phoenix 2003*.
- Chin, J., Harting, J., Jha, S., Coveney, P.V., Porter, A.R., and Pickles, S.M. 2003 Steering in computational science: mesoscale modelling and simulation *Contemporary Physics*, **44**, 417–434.
- Eickermann, T. (ed) 2000 Gigabit Testbed West – Abschlussbericht. *Technical Report IB-2000-17, Research Centre Jülich, Central Institute for Applied Mathematics*.
- Eickermann, T. & Frings, W. 2000 VISIT – a Visualization Interface Toolkit. *Technical Report IB-2000-16, Research Centre Jülich, Central Institute for Applied Mathematics*.
- Erwin, D. 2002 UNICORE – a Grid computing environment. *Concurrency Computation: Practice and Experience*. **14** 1395–1410.
- Foster, I. & Kesselman, C. 1998 *The Grid – blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers.
- Foster, I., Kesselman, C. & Tuecke, S. 2001 The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, **15(3)**.
- Foster, I., Kesselman, C., Nick, J.M. & Tuecke, S. 2002 The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Presentation at Open Grid Service Infrastructure WG, Global Grid Forum, June 22*.
- Furunishi, M. 2003 *Announcement at Global Grid Forum GGF7, Tokyo, March 2003*.
- Gibbon, P. 2003 Parallel Electrostatic Plasma Coulomb-solver. *Technical Report IB-2003, Research Centre Jülich, Central Institute for Applied Mathematics, 2003*.
- Gibbon, P., Beg, F.N., Clark, E.L., Evans, R.G. & Zepf, M. 2004 Tree-code simulations of proton acceleration from laser-irradiated wire targets. *Phys. Plasma* **11** 4032.
- Johnson, C.R., Parker, S.G. & Weinstein, D. 2000 Large-Scale Computational Science Applications using the SCIRun Problem Solving Environment. In *Proceedings of Supercomputer 2000*.
- Kohl, A.J. & Papadopoulos, P.M. 1998 Efficient and flexible fault tolerance and migration of scientific Simulations using CUMULVS. In *Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools, Welches, Oregon, August 1998*.
- Liere, R.v., Mulder, J.D. & Wijk, J.J.v. 1997 Computational Steering. *Future Generation Computer Systems* **12(5)** 41–450.
- Mangles, S. P. D., Murphy, C. D., Najmudin, Z., Thomas, A. G. R., Collier, J. L., Dangor, A. E., Divall, E. J., Forster, P. S., Gallacher, J. G., Hooker, C. J., Jaroszynski, D. A., Langley, A. J., Mori, W. B., Norreys, P. A., Tsung, F. S., Viskup, R., Walton, B. R., and Krushelnick, K. 2004 Monoenergetic beams of relativistic electrons from intense laser-plasma interactions. *Nature*, **431**, 535–538.
- Rantau, D., Frank, K., Lang, U., Rainer, D. & Wössner, U. 1998 COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data. In *Proceedings of the 2nd Workshop on Immersive Projection Technology, 1998*.
- Tajima, T., and Dawson, J. M. 1979 Laser electron accelerator. *Phys. Rev. Lett.*, **43**, 267–270.
- Tuecke, S. et al., 2003 Open Grid Services Infrastructure (OGSI). *OGSI Working Group of the Global Grid Forum*.