Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 1927–1936

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

A practical and comprehensive graduate course preparing students for research involving scientific computing

Gabrielle Allen[a,b,1,*], Werner Benger[b], Andrei Hutanu[b], Shantenu Jha[b,a], Frank Löffler[b], Erik Schnetter[b,c]

[a] *Department of Computer Science, Louisiana State University*
[b] *Center for Computation & Technology, Louisiana State University*
[c] *Department of Physics & Astronomy, Louisiana State University*

Abstract

We describe a new graduate course in scientific computing that was taught during Fall 2010 at Louisiana State University. The course was designed to provide students with a broad and practical introduction to scientific computing which would provide them with the basic skills and experience to very quickly get involved in research projects involving modern cyberinfrastructure and complex real world scientific problems. The course, which was taken by thirteen graduate students, covered basic skills, networks and data, simulations and application frameworks, scientific visualization, and distributed scientific computing. Notable features of the course include a modularized team-teaching approach, and the integration of national cyberinfrastucture with teaching.

Keywords: education, scientific computing, computation

1. Introduction

Computation is rapidly taking its equal place alongside experimentation and theory as the third leg of modern scientific inquiry. Numerous studies and reports are reflecting on the fundamental changes needed in academic institutions, infrastructure, and education for nations to support and encourage the computational research seen as crucial for advancing agendas in national security, strategic applications such as climate change, CO_2 sequestration or fusion, and preparing the workforce for a digital society and economy.

Core among these issues is the challenge of educating undergraduate and graduate students in computation so that they can contribute to research projects that involve diverse software, significant collaboration between researchers from multiple discipines, and require a broad understanding of computational science. In Louisiana, these challenges are faced at the Center for Computation & Technology (CCT) at Louisiana State University. The CCT is a research unit on the campus that includes over 30 faculty from some 13 different departments working in numerous large scale collaborative projects typically involving computation. These projects usually involve real world application problems, interdisiciplinary approaches, collaboration between faculty, the use of high end cyberinfrastructure, and software development. Students entering this research environment are typically highly motivated and excited to

*gallen@cct.lsu.edu
[1] Corresponding author

 Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.210

be working in such a diverse and challenging environment but can find it hard to develop the experience needed to contribute to projects.

This paper describes a new graduate level course on scientific computing that was offered from the Department of Computer Science at Louisiana State University in Fall 2010 [1]. The course focuses on educating students in practical, real-world issues in the areas of application frameworks, networks and data, visualization, and distributed scientific computing. Lectures cover basic principles and examples in each area, describe state-of-the-art technologies and approaches, and include a large practical component. The course also includes lectures on best practices for software development in diverse, interdisciplinary teams and basic computation skills.

The teaching staff for the course (the authors of this paper) were drawn from faculty and PhD level researchers at CCT. The motivation for joining the teaching staff was primarily to be able to train and recruit students to join research projects, and at a higher center level to encourage collaboration and overlaps between different projects. A second important motivator for some staff was the opportunity to gather teaching experience which is important for promotion and tenure.

The class complements other scientific computing related classes offered at a graduate level by the Computer Science Department at LSU, including *High Performance Computing: Models, Methods and Means* [2] which covers the core concepts of high performance computing and parallelization. LSU has also recently introduced a Concentration in Computational Science to its Masters degree in System Science.

The Scientific Computing class was comprised entirely of graduate students, with six masters students (System Science) and seven doctoral candidates. Of the seven PhD students, four were computer scientists and the remaining three were civil engineers. The course consisted of 25 lectures, 10 classworks, 3 projects and a final exam.

We present in Section 2 the desired course outcomes, Section 3 describes the software components which were highlighted in the course, Section 4 describes in detail the different course modules and their content. Section 5 describes the significant practical component of the course using national production systems. Finally, Section 6 summarizes the main conclusions and findings of preparing and delivering this course.

2. Course Outcomes

The computer science department at Louisiana State University, in collaboration with the Center for Computation & Technology, is developing new curricula in computational science, including a new concentration in computational science as part of the undergraduate degree and a new option to specialize in computational science as part of the Masters in System Science. The aim of the course was to contribute to the education of graduate students in the areas of computational science and scientific computing. While the home of the course was the computer science department, the course was aimed to also be appropriate for interested students from any other discipline, in that it would prepare computer scientists for working in areas related to scientific computing, and expose students in different science fields to the issues and technologies involved in high-end scientific computing.

Specific desired outcomes were that the students would be exposed to the state of the art techniques in practical scientific computing, which would been enable them to successfully pursue research projects at the CCT. Students would learn how to collaborate with colleagues and staff both locally and remote, including other students, postdocs, faculty as well as system administrators and help desk personnel.

An important outcome was that students would learn how to quickly overcome the various logistical challenges associated with using national cyberinfrastructure and using and extending research software. Our aim here was to install the confidence and expectations in students that would encourage them to seek out and use such resources through their research careers.

3. Integration with Research Software

One challenge faced in computational related research is educating students and researchers on how to deal with and contribute to software. Academic research software is often at the cutting edge of both the science and underlying enabling technologies such as large scale computers, high speed networks, advanced visualization or distributed computing. Software, which is often open source, is typically developed in a research environment with minimal funding, is of varying quality, and can be lacking in documentation, robustness, support and attention to details not of prime import for the motivating research agenda such as usability, user interfaces or integrated verification and validation.

Module	Software Component
A: Basic Skills	SSH/GSISSH, OpenGL, Subversion
B: Networks and Data	iPerf, GSISSH, GridFTP, CORBA
C: Simulations and Application Frameworks	Cactus Framework, Einstein Toolkit, gnuplot
D: Scientific Visualization	Vish
E: Distributed Scientific Computing	SAGA

Figure 1: Software components used in the Scientific Computing course by the students through coursework and projects.

This issue is well recognized and different initiatives are underway to improve the usability and sustainability of academic software. It is also clear though that students need to be better prepared to use and contribute to such software. This course involved research software in each of the components, in the main with software developed, supported, and distributed by the instructors themselves which is used in a range of national and international research projects:

In addition to the basic software components introduced in the Basic Skills module, the key software components used in this course included (see Fig. 1):

- The *Cactus software framework* [3, 4] is an open-source component framework designed for the collaborative development of large scale applications and primarily developed at LSU. Cactus is primarily used in research in physics and engineering, greatly simplifying implementing numerical algorithms on parallel and distributed cutting-edge high performance computing (HPC) architectures. Cactus is used by scientists in multiple scientific fields, but is largest set of users come from numerical relativity, where the *Einstein Toolkit* [5] provides an advanced research environment used by many students, postdocs and researchers around the world.

- Developed primarily in the Visualization Group at CCT, *Vish* is a visualization shell [6] which is based on pure C++ and provides strong encapsulation among components, such as the graphical user interface, I/O layer and actual rendering. *Vish* is used as the infrastructure for several graduate projects at LSU where students are developing and implementing new algorithms for visualizing and understanding complex fluids and astrophysical systems.

- *SAGA* is an API providing the basic functionality required to build distributed applications, tools and frameworks so as to be independent of the details of the underlying infrastructure, and can be used to provide simple access layers for distributed systems and abstractions for applications. This course introduced students to the open source C++ implementation of SAGA that is developed at LSU.

- In the *Networks and Data* module, students used *iperf*, TeraGrid GSISSH, GridFTP and for an, optional assignment they used an implementation of CORBA. In the *iperf* exercise the students installed iperf in user space on the TeraGrid; the GridFTP installation was already supported by TeraGrid. GSISSH is also supported by TeraGrid but can be considered experimental software. CORBA is a mature specification with many production quality implementations and documentation available.

4. Course Organization and Curricula

Aside from a first general lecture introducing the class and setting the scene for scientific computing, the course was divided into five different modules covering different topics and taught by a different instructor. Although the modules were designed to be relatively independent, the modules were also designed to be coherent with each other and carry consistent threads. Each modules consisted of four to seven lectures depending on the needs of the topic, with each lecture lasting eighty minutes. The module instructor was responsible for the module curricula, coursework and a component of the final exam.

The organization, content and coherence of the course was discussed between all instructors during weekly curricula meetings held during the summer of 2010 and during the course itself. The instructors regularly attended lectures in other modules, and one of the two computer science graduate faculty members responsible for the course attended

Course Component	Percentage of Final Grade
Coursework for A: Basic Skills	0
Coursework for B: Networks and Data	10
Final exam for B: Networks and Data	10
Coursework for C: Simulations and Application Frameworks	10
Project for C: Simulations and Application Frameworks	10
Coursework for D: Scientific Visualization	10
Project for D: Scientific Visualization	10
Coursework for E: Distributed Scientific Computing	10
Project for E: Distributed Scientific Computing	10
Final Exam	20

Figure 2: Grading schedule

all lectures. A mailing list was set up to encourage the students to communicate with all the instructors. A wiki [1] and SVN repository were used for all the instructors to be able to easily edit and post lectures and material.

The selected modules for the course, which are described in more detail in this section, were *A: Basic Skills*; *B: Networks & Data*; *C: Simulations & Application Frameworks*; *D: Scientific Visualization*; and *E: Distributed Scientific Computing*.

Our grading approach (see Fig 2) was to assign equal weights (20%) to all modules except Basic Skills and each instructor had the freedom of choosing how to provide the grade for their module. The remaining 20% was assigned to a final exam covering all modules (including Basic Skills).

4.1. Basic Skills

One of the issues we have identified with incoming graduate students into our research groups is that there is a wide variability in their basic computer and computational skills and experience. For this reason, we provided a first module *Basic Skills* which consisted of a large number of lectures, covering fundamental topics important for all the other modules such as Unix/Linux, versioning systems, compilation, collaborative working practices etc. Fig. 3 provides a list of the included lectures. Two noteworthy aspects of this module were that the lectures were not in a block but interwoven between other modules, and also that the content was not fixed and was dynamically modified to respond to the class' particular needs.

Practical components for this module was optional and did not count towards the final grade, but prepared the students for upcoming lectures of other modules. This included e.g. ensuring that students were able to log into all supercomputer systems, to write a simple program, to compile it on a supercomputer and to visualize the resulting output. Other tasks involved showing understanding of simple numerical methods by implementing, running and analyzing its results on a supercomputer, as well as understanding of the concepts of OpenGL by using it to show a simple animation of a molecule.

Not all lectures in this module directly involved course work to avoid conflicts with the larger projects of other modules. However, most topics of this module had to be applied within the other modules. This included topics like the advanced usage of Secure Shell, best coding practices and practices for software development, usage of revision control systems, as well as the usage of compilers and debuggers.

4.2. Networks and Data

Networks and data are becoming increasingly important resources for scientific discovery, and our aim was to provide students an understanding of the meaning and utility of networking in computing, give them an introduction to basic practical methods of using high-speed networks for scientific computing and introduce to methods of dealing with large scientific data. This module covers network basics, middleware and distributed data management, and grid-based visualization.

The module was structured progressively with each lecture building on information from the previous lectures, as were the concepts and tools introduced using tools and features presented in previous lectures. The module was started with an introduction to basic networking concepts in the first lecture, such as simple network tools and core transport protocols. The second lecture introduced the simple but widely used network applications of bulk data transfer and

	Lecture	Learning Objectives
A1	Preliminaries	Overview of essential knowledge: Unix/Linux, shells, secure shell, text editors, compiling and linking, Make-system and simple visualization.
A2	Introduction to Numerical Methods	Root finding, interpolation, extrapolation, integration, differentiation, ordinary differential equations, partial differential equations, random numbers and Monte-Carlo.
A3	Vector Algebra; Basic Visualization Programming	Vectors and vector addition, unit and base vectors, vector components, rectangular coordinates in 2D and 3D, vectors connecting two points, vector products; Basic visualization programming: OpenGL/GLUT, example implementation.
A4	Advanced Secure Shell	History, authentication definition and mechanisms, public-key cryptography, uses of secure shell, implementation examples, secure shell usage examples
A5	Best Coding Practices	Software project planning, plan strategy examples, testing, coding styles, good programming practices
A6	Software Development, Revision Control	Project management: communication channels, version control system types and usage examples, issue trackers, documentation and documentation formats
A7	Compiling, Debugging, Profiling	Compiler definition and sub-steps, object files, libraries and executables, filesystem placements and program loaders; bug prevention techniques, debugging definition and general debugging techniques, GDB overview and example session; profiling definition, profiling types, gprof example

Figure 3: Module curricula for Basic Skills

videoconferencing. The third lecture introduced middleware, as the means to build network/distributed applications and the final two lectures presented high-level applications of networking such as distributed data management and distributed visualization. More details on the module structure are given in Fig. 4.

The practical element of this module was a hands-on introduction to using high-speed networks on the TeraGrid. Students were introduced to *iperf* and GridFTP [7] and they were given an assignment to measure the network performance between various sites using these tools and to document it into a report. For this, the students have been provided with TeraGrid accounts (see Sec 5.1). An optional additional assignment for the students that wanted to improve their grade for the report was given in the form of a programming exercise, where students were requested to write a simple client/server example using CORBA [8].

	Lecture	Learning Objectives
B1	Networks Introduction	Network basics: network use/applications, orders of magnitude, network layers, transport protocols (TCP/UDP), iperf for transport performance.
B2	Network Applications, Special Subjects	Network "application layer". Example basic applications: bulk data transfer and videoconferencing. Application requirements. GridFTP and bulk or partial data transfer. Circuit-based network services and other special subjects.
B3	Middleware	Features offered by middleware to distributed application developers. Components of a middleware package (IDL, naming services, protocol). CORBA, Java RMI and SOAP. Dealing with network issues in a distributed system (e.g. latency). End-to-end arguments in system design.
B4	Distributed Data Management	Distributed storage/file systems (GFS, iRODS, DHT). Remote I/O systems: Middleware, GridFTP, Parrot, eavivdata. Staging and replication. Scheduling of distributed applications: objectives, and information used by the scheduling process.
B5	Grid-based visualization	Scenarios for distributed or remote visualization. Distributing the visualization pipeline. Examples: VNC, ParaView, VisIt, Visapult, Viracocha, eaviv.

Figure 4: Module curricula for Networks and Data

4.3. Simulations and Application Frameworks

The aim of the *Simulations and Application Frameworks* module was to explain the essential elements of modern simulation codes that are run on parallel supercomputers. At LSU such codes are being used to study complex real world problems such as modeling black holes, predicting the effects of hurricanes, or optimizing oil and gas production from underground reservoirs. Many research projects in computer science and electrical engineering at CCT are motivated by such real world problems, however the students often do not understand how these codes are structured, nor the goals and needs of the scientists developing and running them, nor the hardware and technology limitations that these codes encounter.

The module curricula (see Fig. 5) covered what a simulation code is, using as example a generic *initial value problem* that can be used to describe many systems in physics and engineering, and which are implemented in simulation codes via domain decomposition over parallel processors with time iterations and external boundary conditions.

The practical element of the simulation module used the Cactus framework [3, 4], a component framework which is developed at LSU, and used in research projects there in astrophysics, civil engineering, mechanical engineering and computer science. Students first learned how to assemble, configure, and build a Cactus configuration, and then execute it across many cores of a national supercomputer resource (Sec. 5.1), and finally visualize and verify output data created by the simulation.

In a second practical component students developed their own Cactus component implementing an additional physics analysis module, incorporating it into their copy of the Einstein Toolkit code base, and executing and validating it on the same supercomputer resource.

	Lecture	Learning Objectives
C1	Simulation Science Basics	Abstracting the real world via a physical system. Modelling a physical system with equations. Simulation domain, initial configuration, boundary conditions, time stepping, analysis. Cluster architecture: front-ends, file systems, remote access, allocations, queuing systems. Example session.
C2	Simulating Complex Systems	Complex physical systems, component model for large applications. Discretizing equations. Physical vs. computational components. Tight and loose component coupling. Example application: the Einstein Toolkit.
C3	Framework Architecture	Many-component systems; Cactus overview, component interfaces; Data flow, execution schedule; Concepts for component and interface design; Concepts for application design, selecting and combining components
C4	Getting Science Out of Computing	Computational efficiency, data representation, data movement between components; Component life cycle, reliability, repeatability of scientific results; Working in large, international collaborations

Figure 5: Module curricula for Simulations & Application Frameworks

4.4. Scientific Visualization

As data sizes generated by simulations and instruments increase, scientific visualization is growing in importance in scientific computing in providing insight beyond purely numerical results. One of the aims of this module was to explain to students the importance of this field, and to show them how multidimensional data should be analyzed using multidimensional visualization techniques; perhaps in contrast to common beliefs of scientists that a single number of 1D graph is a reasonable result from a simulation.

The *Scientific Visualization* module (Fig. 7) aimed to provide a general understanding of basic concepts, in particular contrasting related domains such as computer graphics and data visualization. The module consisted of four lectures to introduce the scope and concepts of scientific visualization; present mathematical concepts; review geometric algebra; and provide programming experience in visualization research software.

The first three lectures communicated generic concepts that were not tied to a specific software package and were intended to provide a knowledge basis that would remain valid throughout future years, covering cross-application concepts such as the visualization pipeline or the construction of visualization networks.

The module emphasized that scientific visualization is more than simply artistic rendering but rather involves mathematical concepts, and included major mathematical concepts relevant for visualization such as differential geometry and topology. In many practical cases, visualization algorithms are implemented on an "ad-hoc" basis, solving a particular problem without considering a more generic approach. The course content aimed to show the "big picture" and illustrate a systematic methodology, including discussing the use of a general data model.

The course also covered topics beyond the mainstream, such as Geometric Algebra. Since Geometric Algebra allows a direct interpretation of numerical operations it was directly insightful to most students, even though its concepts were rather new and unusual. Fundamental concepts such as Geometric Algebra are suitable to be communicated to young generations who are not yet determined already to use alternatives such as matrix algebra.

The final lecture presented a current research visualization framework, the "Vish" visualization shell [6], which allowed the students to get used to practical issues such as software installations on different platforms. A module extending Vish to do some basic rendering requires only 50 lines of source code, thus even with limited knowledge in C++ it is possible to get started to explore rendering capabilities and experiment with interactive feedback of visualization parameters.

The module culminated in a final project where students used actual datasets from ongoing research work or implemented their visualization modules in the Vish framework. The executed projects included visualizing the "Wind Field of Hurricane Katrina" (Fig. 6) or datasets from a Cactus simulation run involving 32GB of data. Coding projects provided shader-based rendering methods of stream lines, implementing a visualization module depicting the evolution of time-dependent surfaces or modules covering aspects of Geometric Algebra for visualizing particles.

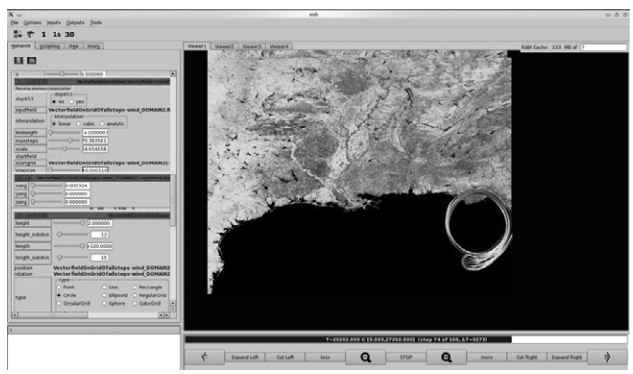

Figure 6: Wind Field of Hurricane Katrina, from Module D student project (Credits: Liu Ke, Ling Zhu and Qian Zhang).

	Lecture	Learning Objectives
D1	Concepts and Definition	Definition and Scope of Scientific Visualization; Concept of the Visualization Pipeline; Data Flow / Control Flow; Visual Programming; Overview of Visualization Software; "Vish" Visualization Shell; Data Types; File formats (HDF5).
D2	Mathematical Concepts	"Fiber Bundle" - Formal definition, the base and fiber space. Describing the Base Space via Topology: adjacency, neighborhood, connectivity, meshes. Describing the Fiber Space via Differential Geometry: differentiable manifolds, coordinate-free formulations, vector calculus. Geometric Algebra: n-dimensional coordinate-free formulations, rotations, bi-vectors, multivectors, quaternions.
D3	Mathematical Concepts II	Simple Fiber Bundles. Examples of Fiber Bundle visualizations. Visualizing tensor fields. Geometric Algebra. n-dimensional coordinate-free formulations. Bi-vectors and rotations. The "F5" Fiber Bundle Data Model. Implementation of the Mathematical Principles. The F5 File Format. Identifying properties and classification of data types.
D4	Vish Coding	The Concepts of Vish; Directory Organization; Vish Objects - inputs and outputs; Render Objects; Fiber Bundle Vish Objects; Field Render Objects; Discussion of implementation issues.

Figure 7: Module curricula for Scientific Visualization

4.5. Distributed Scientific Computing

This module (Fig. 8) covered the theory and practice of Distributed Scientific Computing, with an emphasis on production-grade distributed cyberinfrastructure. There exist a broad range of computational infrastructure with varying support for application characteristics, usage modes and even access policies, all of which influence the usefulness of a given infrastructure for scientific applications. This module provided sufficient understanding of production grade distributed cyberinfrastructure to enable students to discern and determine which infrastructure would be appropriate and effective for their applications.

The module began by motivating the role and need for distributed computing by understanding six rather different distributed applications – and analyzing them for the following points: why distributed, how distributed and the challenges, success and/or issues in distributing them. Elements of data-intensive computing and the role of distributed data were also covered. After establishing the *essential* role of distributed computing in these applications, as well as understanding the critical role of *execution environments* for distributed computing, the students were exposed to a range of production distributed infrastructures and a brief overview of the design principles, objectives and application characteristics supported. Specific examples of production Grids – high-throughput as well as high-performance were analyzed.

To appreciate the reasons behind the rise of Cloud Computing, the module provided a basic overview of the important underlying trend in computing technology and data-intensive computing. This led to the broader domain of scientific computing as enabled on commercial infrastructure (Amazon EC2, Azure) as well as other non-commercial Cloud offerings

Practically, the students were given hands on experience with both virtualized resource and "bare-metal" resources thanks to the FutureGrid – the US NSF's Track-2d Experimental Grid System as the test-bed, as well as local machines at LSU/LONI and CCT. Students mostly experimented with Eucalyptus on India and Sierra resources of the FutureGrid, as well as worked with a SAGA-enabled version of MapReduce.

	Lecture	Learning Objectives
E1	Introduction to the Practise of Distributed Computing - I	WLCG as a motivating example (order of magnitude estimates of number of jobs submitted, data transferred, CPU cycles consumed), Distributed Application Exemplars, Analyzing why and how distributed, and challenges & success in distribution. Introduction to SAGA and FutureGrid (FG).
E2	Introduction to the Practise of Distributed Computing - II	Examples of Production Grid Infrastructure - HPC vs HTC, Research vs Production, Commercial. Introduction to SAGA; Writing *your* first Dist. Application.
E3	Cloud Computing & Master-Worker Pattern	Cloud Computing. Convergence of multiple trends, Understanding Amazon Examples of M-W Pattern: SAGA-based MapReduce, Word Count Application and Mandelbrot Set.
E4	To Distributed or not to Distribute?	Case Studies, Observations on Distributed Applications, Development Objectives; Projects on FG.

Figure 8: Module curricula for Distributed Scientific Computing

The Sapir-Whorf hypotheis implies that "language influences the (habitual) thought". The implication and analog in scientific computing is that the infrastructure used shapes the practise and formulation of research. Conversely, "for a given scientific application/research question, which cyberinfrastructure should I use?". Lecture E2 provided a brief overview and understanding of available infrastructures, and E4 covered the reasons and answers behind which infrastructure would be suitable for a broad range of applications.

5. Practical Experience

The course was designed to include hands-on experience with high end, production scientific computing infrastructure. We decided to use two national infrastructures provided by the National Science Foundation: the TeraGrid[2] (including the system hosted at LSU) and the recently deployed FutureGrid[3]. Ideally a single infrastructure would

[2] http://www.teragrid.org
[3] http://www.futuregrid.org

have been used to reduce the administrative overhead and potential complexity for the students, however no single infrastructure was identified which could easily provide all the capabilities required. Both these infrastructures already have mechanisms in place to support educational activities.

5.1. NSF TeraGrid

An educational allocation was applied for and quickly awarded on the TeraGrid following a straightforward and simple procedure on the TeraGrid web page[4]. Accounts are then requested through this allocation for the students, and although these are quickly dealt with by TeraGrid staff it is difficult to do this in advance since the composition of classes is very dynamic during the initial weeks and special assistance to deal with this was needed from the TeraGrid staff. Further, the accounts require information about the students which may not be readily available to lecturers through standard class systems.

At the beginning of the course, the students were handed the information for their TeraGrid accounts and as a part of the first assignment in the *Basic Skills* module they were asked to log-in to all sites to confirm that their accounts were active.

For the *Networks and Data* module, access was provided to four different TeraGrid sites (LONI:QueenBee, TACC:Longhorn and Ranger, NCSA:Abe and tape system and Purdue:Steele) for students to measure network performance between various combinations of sites. One issue encountered was that TeraGrid nodes that have a high speed connection are not generally configured for interactive use, and in one case the network configuration was ambiguous (one host name for two different machines). Also, the baseline TeraGrid network performance is not documented, so we needed to test the performance of the network before the course started to select interesting resources (those with a high network capacity).

For the Simulations and Application Frameworks module, the students were given access to a compute allocation on QueenBee to install, compile, and run a Cactus application across multiple processors and then visualize the output data. Through this the students early on in the class became familiar with some of the real world issues that computational scientists deal with in their work.

5.2. NSF FutureGrid

About three weeks before the start of the Distributed Scientific Computing module we devoted approximately 0.5 of a lecture to provide an overview of FutureGrid (FG) and the process to be employed to get accounts and started. As part of a homework assignment each student had to confirm access and successful login to FG-Sierra and FG-India to be acquainted with FG when module E lectures began.

FutureGrid (FG) was used by students to (i) compile, deploy and execute basic SAGA commands (E1 coursework) (ii) learn the basics of remote job submission and elementary Master-Worker based distributed applications (such as MapReduce and computing the Mandelbrot Set) using FG-India and FG-Sierra nodes (E2 and E3 coursework), (iii) to get hands on training with IaaS Clouds, namely stand-up virtual machines using Eucalyptus and deploy software and/or applications from (i) and (ii) (E3 and E4 coursework).

Students also used Eucalyptus on FG-India and FG-Sierra to do their module E projects, which included the task of (a) using Clouds as accelerators for Cactus-based applications, (b) calculating the value of PI using distributed tasks, (c) extending the calculation of the Mandelbrot Set to "new" backends on FutureGrid (in addition to the "default" remote/ssh backends), and (d) executing of workers on bare-metal as well as Clouds concurrently (i.e., hybrid Grid-Cloud infrastructure) for M-W based applications.

In general, FG proved to be a valuable resource and without the active support of the FG team in the early stage of their deployment of this facility, module E would not have been possible.

[4] https://www.teragrid.org/web/user-support/startup

6. Conclusions

The course proved to be an interesting and worthwhile experience for both the instructors and students. The majority of the students were awarded the top grade for the class, reflecting their performance, interest and effort. No students failed the class. The students' reaction to the class was very positive, with the main criticism being that there was too much classwork and a lesser criticism being that some of the general material in the Basic Skills section was too basic. In particular, the format of the class with multiple, experienced instructors was viewed very positively.

One finding from teaching the class was that the students were not well prepared in all the basic research skills for scientific computing; for example in writing clear research or technical reports, in verification and validation of results obtained from computers, and in collaborative working procedures such as how to appropriately cite contributions from others or use software.

In terms of software development, it was clear that the students had varying abilities and experience with basic programming. Also it was found that students did not know how to find their way around large, open-source software packages. In addition, many students did not know Unix/Linux, or how to install an open-source software package in a generic manner.

In future offering for this course, this may be addressed through adding prerequisites for the course (for the Fall 2010 course there were no official prerequisites). However, this issue points to a more general problem of designing new comprehensive curricula for scientific computing and computational science.

The practical component of the class was very successful. It was clear that with the appropriate encouragement and guidance the students could make good use of national production resources as part of their educational curricula. The students were able to work with, and extend, diverse and experimental software.

We expect to offer the scientific computing course at LSU again, building on the experiences described in this paper. Future course offering will include surveying the students before, during and after the course to investigate more thoroughly the skills of students entering and completing the course.

Acknowledgements

We acknowledge NSF awards EPS-0701491, OCI 0947825 and PHY 0904015. Funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and HPCOPS NSF-OCI 0710874. We thank the Center for Computation & Technology for making the classroom used for the course available. We acknowledge TeraGrid allocation TG-SEE100004, LONI allocation loni_cactus05 and the support of Vicki Halberstadt. We thank the FutureGrid team, in particular Greg Pike and Geoffrey Fox, for outstanding support. FutureGrid is supported by NSF 0910812. We thank Dr Coretta Douglas for curricula and logistical advice. We thank Marcel Ritter, Andre Merzky and Sharath Maddineni for their assistance.

References

[1] Scientific Computing Course at Louisiana State University, Fall 2010. [link].
URL `http://wiki.cct.lsu.edu/sci-comp`

[2] High Performance Computing: Models, Methods and Means (CSC 7600). [link].
URL `https://www.cct.lsu.edu/csc7600/Home.html`

[3] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, J. Shalf, The Cactus framework and toolkit: Design and applications, in: Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science, Springer, Berlin, 2003.
URL `http://edoc.mpg.de/3341`

[4] Cactus Computational Toolkit. [link].
URL `http://www.cactuscode.org/`

[5] Einstein Toolkit: Open software for relativistic astrophysics. [link].
URL `http://einsteintoolkit.org/`

[6] W. Benger, G. Ritter, R. Heinzl, The Concepts of VISH, in: 4^{th} High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007, Berlin, Lehmanns Media-LOB.de, 2007, pp. 26–39.

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, The Globus Striped GridFTP Framework and Server, in: SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2005, p. 54. doi:http://dx.doi.org/10.1109/SC.2005.72.

[8] OMG, The Common Object Request Broker: Architecture and Specification, Tech. rep., Object Management Group and X/Open. OMG Document Number 91.12.1 (1991).