Curriculum Modules, Software Laboratories, and an Inexpensive Hardware Platform for Teaching Computational Methods to Undergraduate Computer Science Students

Submitted in partial fulfillment
of the Requirements for the

Degree of
Doctor of Philosophy in Interdisciplinary Studies
with a Concentration in Arts and Sciences
and a specialization in Computer Science
at the Union Institute & University

Cincinnati, Ohio

Charles Franklin Peck

April 5, 2009

Core Faculty: Chris Hables Gray, Ph.D.

Abstract

Computational methods are increasingly important to 21st century research and education; bioinformatics and climate change are just two examples of this trend. In this context computer scientists play an important role, facilitating the development and use of the methods and tools used to support computationally-based approaches. The undergraduate curriculum in computer science is one place where computational tools and methods can be introduced to facilitate the development of appropriately prepared computer scientists. To facilitate the evolution of the pedagogy, this dissertation identifies, develops, and organizes curriculum materials, software laboratories, and the reference design for an inexpensive portable cluster computer, all of which are specifically designed to support the teaching of computational methods to undergraduate computer science students.

Keywords: computational science, computational thinking, computer science, undergraduate curriculum

Contents

List of Tables

List of Figures

Chapter 1

Introduction

To varying degrees, computer scientists have long recognized the value of including computational science topics in the undergraduate computer science curriculum. Recently, the support for this notion has solidified and expanded to include the two most recent reports from the professional bodies that develop curriculum standards for computer science and a number of presidential commissions. This dissertation first establishes the necessity and value of teaching computational methods to undergraduate computer science students and then describes in detail a set of materials and tools to accomplish this task.

At the highest level, two areas form the larger context for this work: computationally-based methods and multidisciplinary science. [1]

Computationally-based research methods use computers and software to simulate laboratory experiments, and to perform experiments for which there yet is no laboratory analog. The methods employed are usually resource intensive, requiring high performance computing (HPC) systems, and often use large data sets (as input, output or both). The comparatively fast turnaround time and relative simplicity of computational methods initially attracted a wide variety of science disciplines to develop pilot research programs which used this methodology. These computational approaches, which HPC systems

[1] There is no consensus in the community about the use of interdisciplinary *vs.* multidisciplinary to describe work that involves people from more than one traditional academic discipline. I've chosen to use multidisciplinary because it implies the union of two or more disciplines rather than the space between them.

enable, are now an important part of basic and applied research in all of the natural sciences and a growing number of social sciences, the humanities, and the arts.

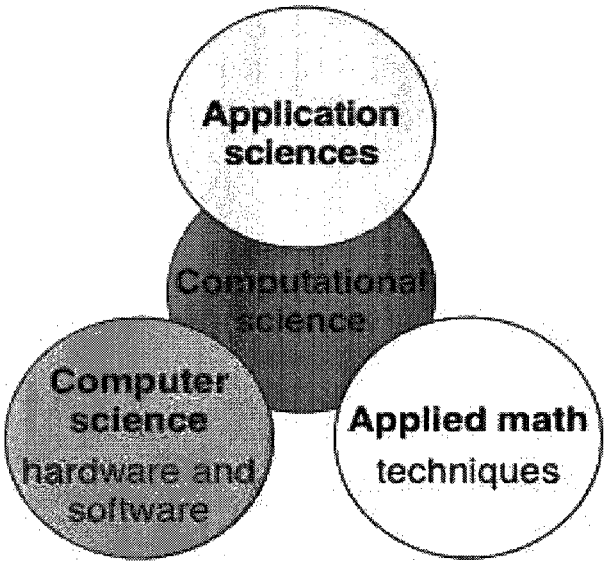
Why does the use of computationally-based research methods continue to grow in popularity across the disciplines? These methods enable us to consider phenomenon which are beyond the reach of our current experimental or observational methods, with characteristics which make them difficult or impossible for us to work with directly. For example:

- too small - atoms, molecules
- too large - galaxies, the universe
- too fast - photosynthesis, protein folding
- too slow - geological processes, climate change
- too complex - blood circulation, weather
- too dangerous - toxic materials, nuclear stockpile stability

Instead, we can build models and simulations which enable us to examine their behavior through the lens of computationally-based inquiry.

Multidisciplinary science is becoming more and more common as we seek solutions to some of the most vexing problems facing society today, *e.g.* climate change, pollution, and energy production and consumption, to name a few. These problems require specialists from a wide range of academic disciplines working together to understand the complex interactions these problems present, and to search for solutions to them. The prevalence of computationally-based approaches makes computer scientists an integral part of these teams. In order to work effectively in these teams, the computer scientists must be taught the language and tools of computational science.

One way to view computational science is as a bridge connecting the sciences to computer science and applied mathematics, as shown by this frequently employed diagram:



To facilitate the inclusion of computational methods into the undergraduate computer science curriculum, this dissertation describes a set of independent curriculum modules based on computational methods. Each contains lecture notes, reference materials and a software laboratory. Also described are detailed construction plans for an inexpensive portable teaching cluster called *LittleFe*, on which all the software laboratories can run. The bibliography is quite extensive and includes background material for each of the topics covered, in addition to items which provide support and context for this work.

To the extent possible, the curriculum modules and software laboratories are designed to be freestanding; it should be easy to extract individual units and use them in a local setting. Taken together, these components represent a complete repository of materials and facilities for teaching computational methods to undergraduate computer science students.

The curriculum modules and software laboratories developed for this dissertation have been used in a variety of contexts to date. A number of courses in the Computer Science department at Earlham College now include components of this work, as well as the SuperComputing (SC) Education Program's summer workshop series. About ten *LittleFe* units have been built and are now a part of the SC Education Program's computational

resources for supporting outreach and workshops. A second set of units is being built to support a component of Dine'h Grid, a project of the Navajo Nation in Crownpoint, New Mexico.

Chapter 2

Literature Review

There are five distinct categories of background material considered in this literature review. The principle area is computational science education, a variety of material in journal articles and books is available in this category. Second are the discipline-specific books that provide background material for the underlying mathematics and science. Third are the articles and books that discuss the pedagogy of teaching computational methods. Fourth is the documentation that accompanies each of the software laboratories. Last are the reports and findings of governmental panels, agencies, and national academies that support the need for this type of work.

The quantity of material required to cover those five categories makes the bibliography much longer than is typical; there are about 110 items in total. To make this quantity of references manageable, I developed a two level keyword structure which tags each entry with one or more resource types and one or more disciplines. This will facilitate easy identification of a subset of the resources by the content area the reader is interested in. I will publish the bibliography online with the curriculum modules in a form that will permit users to filter and display entries using this keyword structure. The *Bibliography* chapter of this document describes the keyword structure in detail.

In the literature review which follows, I consider the computational science education, pedagogical methods, and supporting document categories of materials. Together these three categories frame what I am doing and how it fits into the existing body of knowledge, shape the pedagogical characteristics of the work to be produced, and provide support from a broad community of experts as to the need for work in this specific area. The remaining two categories, background science and software manuals, are covered with each of the appropriate curriculum modules and/or software laboratories found later in this dissertation.

2.1 Motivation

There are many contemporary sources which document the need for broader and deeper computational science education at the undergraduate level, *e.g.* (President's Information Technology Advisory Committee, 2005), and give general recommendations for material to cover in this context, *e.g.* (Joint Task Force on Computing Curricula, 2001). Going back further, one finds references dating back to the early 1990's (Stevenson, 1994) calling for the inclusion of computational science concepts in the undergraduate computer science curriculum. However, there exist few resources for faculty which offer the combination of specific curriculum modules and software designed to teach undergraduate computer science students a modest amount of the theory, techniques, and tools of computational science.

A large body of work exists to support teaching computer science students how to build software, including software specifically for science. What is missing are course models and laboratory materials that assume the existence of the basic tools, and instead focus on how to use them to solve a wide variety of scientific problems; this is the heart of computational science. Dr. Robert Panoff of the Shodor Foundation was one of the first people to make this distinction. My work builds on the work that he and others have done over the past 15 years.

Computational science is a multidisciplinary field, typically involving professionals from computer science, mathematics and other disciplines, working together to understand and model the complex systems found in nature and society. Many of the interesting questions that science is attempting to answer now are multidisciplinary in nature and require such teams. The undergraduate environment is a good place to have students start working on multidisciplinary problems if we expect them to be able to do this effectively in graduate school and beyond. In order to participate effectively in such multidisciplinary endeavors, students of computer science must be equipped with the language concepts and tools of computationally-based methods of inquiry and analysis.

Computational methods have developed to the point where they now take their place alongside theory and experiment as the third method of scientific inquiry. In no way do computational approaches supplant theory and experiment. They depend on the knowledge gained from those methods to extend our ability to reliably develop new information. Computational methods provide a complimentary lens through which we can view the world, hopefully leading to a better understanding of the complex biological, geological, and social systems at work around us.

One driving force behind the widespread adoption of computational methods is the increasing availability of high performance computing resources. Driven by Moore's "Law" (Moore, 1965) the "power" of a typical computer roughly doubles every 18 months. [1] This allows more scientists and researchers from a broader range of disciplines to use computational methods, *e.g.* simulation, in their research and teaching. To work effectively in this climate, the next generation of academics, particularly computer scientists, needs to be facile with computationally-based research methods.

[1] Strictly speaking Moore's observation only applies to the unit space occupied by a transistor. As used here it is just a rough measure of the overall performance while holding the cost constant.

2.2 My Problem in the Context of Larger Issues

Four distinct areas form the larger context for my work: computationally-based methods, multidisciplinary science, undergraduate computer science education, and open source software.

2.2.1 Computationally-based Research Methods

Computationally-based research methods are increasingly important in Mathematics, Physics, Geosciences, Biology, Chemistry, Economics, Sociology, and Literature. For example, they are frequently employed in such diverse areas as:

- molecular dynamics (*e.g.* protein folding for disease origin and drug design)
- global climate change
- contaminant flow analysis in groundwater
- patterns of voter discrimination
- authorship

While many sources have put forth a similar view of computational science, the most powerful support I can find comes from the President's Information Technology Advisory Committee report from June of 2005. The report, titled *Computational Science: Ensuring America's Competitiveness*, contains a wealth of information related to both the current state of computational science and the future directions the field could take. Here is the committee's principal finding:

> Computational science is now indispensable to the solution of complex problems in every sector, from traditional science and engineering domains to such key areas as national security, public health, and economic innovation. Advances in

> computing and connectivity make it possible to develop computational models and capture and analyze unprecedented amounts of experimental and observational data to address problems previously deemed intractable or beyond imagination. Yet, despite the great opportunities and needs, universities and the Federal government have not effectively recognized the strategic significance of computational science in either their organizational structures or their research and educational planning. These inadequacies compromise U.S. scientific leadership, economic competitiveness, and national security.

While my own work focuses on only a small portion of the needs outlined by the report it does speak directly to the undergraduate education component of their findings.

2.2.2 Multidisciplinary Research

Multidisciplinary research is now becoming the norm in the sciences. I believe it is accurate to characterize most of 20th century natural science as both disciplinary and reductionist. That is, it was performed by groups of like-trained scientists focusing on taking objects apart as a way to understand those objects and the world around us. My belief is that in the 21st century natural science research will be more integrative; with modeling, simulation, complexity, dynamic systems and chaos included as parts of many research programs. In effect these research programs will be moving back and looking at much larger pictures. One place to find support for this view is Project Keliedescope's analysis of recent awardees from the National Science Foundation's Course, Curriculum, and Laboratory Improvement grant program.

> A review of projects funded over the past several years indicates increasing attention to transforming courses or sets of courses to make a variety of connections: between learning about content and about process; between

disciplines within and beyond the fields of science, technology, engineering, and mathematics; ...

The theories, techniques, and tools of computational science are the basis for many of these multidisciplinary approaches to contemporary problems in a wide variety of natural science disciplines, even in the humanities, arts and the social sciences.

Much of this multidisciplinary work involves computer scientists working with colleagues in other departments to develop the software and hardware systems to support computational research methods. The undergraduate setting is particularly conducive to the development of these multidisciplinary approaches. The frequent student-faculty interactions, combined with the unusually high level of inter-departmental relationships, provide a rich environment for this type of teaching and research.

2.2.3 Undergraduate Computer Science Education

Undergraduate liberal arts environments, including Earlham College where I teach, produce a disproportionally large percentage of future science Ph.D.s. Earlham has a long history of exceptional strength in science education. According to the 2000 Baccalaureate Origins Report, which ranks institutions according to the ratio of Ph.D.s granted to bachelors awarded, Earlham ranked 21st in the Science and Engineering category among all institutions of higher learning.

Many of these future science Ph.D. holders also go on to teach at the undergraduate level, which is exactly where more computational science education is needed. By exposing them to these techniques early in their careers, we increase the likelihood that they will use them in their research programs and include them in their own course syllabi.

The curriculum in computer science is largely guided by periodic reports issued by a joint task force of the two principle professional societies, the IEEE Computer Society (IEEE-CS) and the Association for Computing Machinery (ACM). The IEEE-CS/ACM

task force issues full curriculum guidelines roughly every 10 years. Their most recent, *Computing Curricula 2001 Final Report*, strongly supports the notion of including computational science material in the undergraduate computer science curriculum. This was also the first of these reports, which date back to 1968, to include a specific focus group for computational science as part of computer science's body of knowledge.

Liberal arts colleges also have another source for computer science curriculum guidance, the Liberal Arts Computer Science Consortium. Their most recent report, *A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science* also includes a suggestion to include computational science as an elective (Liberal Arts Computer Science Consortium, 2007).

In November of 2003, Charles Swanson, of the Krell Institute in Ames, Iowa, surveyed undergraduate and graduate programs in computational science (Swanson, 2003). There were only 16 undergraduate programs in computational science or a closely related discipline (*e.g.* computational mathematics), and 18 additional undergraduate schools offering at least one course in computational science. Only one of these was a liberal arts institution. Given their traditional role in producing future teachers, this paucity of undergraduate offerings should be addressed. While I will not be producing an entire program in computational science, I will be creating materials which can be adopted into a wide variety of undergraduate contexts. They will serve as a foundation on which more comprehensive offerings can be built over time.

2.2.4 Open Source Software and Open Protocols

Almost all of my work is based on open source software and open protocols, that which isn't is based on software which is freely available for academic use. This speaks directly to many of the needs outlined by the President's Information Technology Advisory Committee in their September 2000 document, *Developing Open Source Software to*

Advance High End Computing. In this document the panel stresses the importance of developing open source technologies for high performance computing. These quotes illustrate two of their key findings:

- Open source software development efforts are a promising means to enable high performance computing and should be considered an important infrastructure investment by the Federal government.
- Open source software is inherently driven by the community, and as such represents a grass-roots approach to answering user needs. A critical aspect of this approach is leadership based on merit without regard to external administrative hierarchy.

Open source software is a powerful concept which has been evolving for over 20 years. In the past 10 years it has coalesced and developed significant momentum moving into mainstream computing in both the academic and commercial sectors. Open source software is designed, built, and maintained by people from around the world, often without regard for direct financial gain.

The open source software development and distribution model represents a significant departure from the more common, restrictive, commercial licensing model currently typified by companies like Microsoft and Oracle. The definition of open source software, provided by the Open Source Initiative (`http://www.opensource.org`), includes the following attributes which distinguish open source from commercial software.

- Free Redistribution - The license may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.
- Source Code - The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not

distributed with source code, there must be a well publicized means of downloading the source code, without charge, via the Internet.

- Derived Works - The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

The definition of open source software makes it possible to *look inside the box.* Access to application source code is crucial to understanding and learning the algorithms and data structures employed by developers, addressing bugs encountered in the original code, and improving and extending the software itself.

The notion of open source software and open protocols has been endorsed well beyond the confines of science; one example comes from global finance. In 2005 the World Bank issued a report endorsing open technology standards. *"Open source does not define an open information and communications technology ecosystem, but it can be an important transformative element"* the report states. *"To date, open source has been the most disruptive element of the entire open agenda, provoking reexamination of information and communications technology ecosystems and policies."*

These four areas: computationally-based methods, multidisciplinary science, undergraduate education, and open source software, define the broad context of my dissertation. Computational approaches define the methodology of this work. The multidisciplinary aspects of the problems at hand shape how the work is approached. The undergraduate setting sets the level at which the materials are presented. Lastly, open source software and open protocols give us the tools to work with. Together these four areas form the context for the development and organization of curriculum materials, software tools, and a hardware platform for this dissertation.

2.3 My Work in the Context of Computational Science Education

My work builds on previous work done by a number of people who come from a variety of intellectual backgrounds in science, mathematics, and computer science. It is the developing area of undergraduate computational science education that forms the most important foundation for my work. I highlight the landscape of that area next.

One of the early papers to espouse a new view of computational science education was written by Robert Panoff in 1995, *The Four A's of Computational Science Education: Application, Algorithm, Architecture, and Active Learning* (Panoff, 1995). In it Panoff articulates a view of computational science education that uses active learning to explore the interplay between the application or model at hand, the underlying algorithms and mathematics which implement that model, and the architecture of the high performance computing resource on which the model is being run. It is exactly this integrated approach to computational science education that I intend to employ in the curriculum modules I am developing as part of this dissertation. I have observed firsthand how using this approach to teaching computational science improves students' engagement level and comprehension.

Understanding the relationships between these three aspects of computational science (application, algorithm and architecture), is critical to applying the technique correctly. Students should not have to understand all of the underlying details associated with each of the three aspects before exploring how they interact with and relate to each other. One key aspect of Panoff's paper, as it relates to my own work, is the notion of active learning as an integral part of computational science education. He illustrates the value of active learning with *RealSim Surface* software, which models the lowest energy configuration of a two-dimensional N-body with a simulated annealing algorithm. The paper also lists a number of other software packages suitable for similar exercises in other scientific disciplines. Active learning, having control over the variables and immediately observing

the results, gives students a powerful tool for understanding how application, algorithm, and architecture are related.

Panoff has authored and coauthored many papers, reports and presentations in the area of computational science education, dating back as far as 1990 and as recently as 2008. Many of the works considered below have been directly or indirectly influenced by his work. Today he serves as the director of the Shodor Foundation and the National Computational Science Institute; two entities dedicated to computational science education at all levels.

In *A New Perspective On Computational Science Education* (Yasar, Rajasethupathy, Tuzun, McCoy, & Harkin, 2000), Osman Yasar, *et al.* take a wide-ranging look at computational science education. They argue persuasively for an overhaul in that the disciplinary model of research institutions where computational science education is spread among disparate departments should be replaced with a model that separates computational science into a distinct educational discipline.

Of particular importance to my own work is their articulation of the need to include computational science education at the undergraduate level. To date, the overwhelming majority of computational science education was happening at the graduate level. This paper makes a case based on the author's own experiences implementing an undergraduate computational science curriculum at SUNY Brockport.

Another professional society which has considered the place of computational methods in the undergraduate curriculum is the Society for Industrial and Applied Mathematics' *Working Group on Computational Science and Engineering Undergraduate Education.* The most recent incarnation of the group included Peter Turner, Angela Shiflet, and Ignatios Vakalis. Both Shiflet and Vakalis have developed well regarded materials, courses, and programs in computational science at their respective home institutions. In the working group's September 2006 report (P. Turner, 2006), they outline both a set of competencies for computational science and a number of approaches to include computational material

in the undergraduate curriculum. While not directed at computer science *per se*, the content and the methods described are useful here in a general way.

Computational methods are almost always built on models of natural systems, models which encompass a wide range of approaches, methods, assumptions, scales, granularity, *etc.* One important characteristic shared by all models are their limits. The basic nature of a model is to represent a series of approximations, assumptions and estimations of a structure and/or process. The particular approximations, assumptions and estimations employed lead directly to the limitations associated with the use of that model.

The limitations of models can be explored both at the meta-level, where the considerations are of a general nature so as to be as broadly applicable as possible, and at the level of a particular model and implementation of that model being applied to a specific set of field-based conditions. Here the considerations can be given relatively precise dimensions which support an analysis grounded in the actual behaviors of the real and simulation systems.

Pilkey and Pilkey's book "Useless Arithmetic: Why Environmental Scientists Can't Predict the Future" (Pilkey & Pilkey-Jarvis, 2007) provides material which thoroughly explores both the meta-level material related to the limits of models and modeling but also provides a series of vignettes which thoroughly explore the specific issues related to particular model and implementation being employed at a specific time and place.

Having established the need for these materials, the appropriateness of teaching these materials to undergraduate computer science students, and the current sources of related materials, I next consider the form and content of the curriculum modules and software laboratories which I will develop and organize.

2.3.1 Curriculum Modules and Software Laboratories *vs.* A Single Course

While some institutions and faculty have chosen to organize computational science material for computer science students into a standalone course, courses, or even a minor program, these models are possible in relatively few settings. The pressures of finding enough faculty resources to teach the core material and faculty with the expertise in computational science or the release time to develop it combine to make even standalone courses problematic for many institutions. These challenges are very similar to those found when trying to include more parallel and distributed computing topics into the undergraduate computer science curriculum.

These pressures lead to the consideration of other techniques for incorporating this material into the undergraduate computer science curriculum. I propose to package the material in smaller chunks, making it possible to incorporate it into existing courses. This alleviates the need for increased faculty resources, decreases the possibility that key concepts in computer science will be missed, and gives faculty a shallower learning curve for adopting computational methods into the existing curriculum at their institutions. Smaller chunks also make it possible for faculty to develop new expertise at a more manageable pace.

A number of faculty in Physics have successfully applied the "small chunks" method as they sought to include computational methods into their undergraduate curriculum. Rubin Landau at Oregon State University and Wolfgang Christian at Davidson University both started out with this approach while working on the development of standalone courses and ultimately a complete program at Oregon State University (Landau, 2006).

Of the settings with enough resources to support a standalone model of computational science education, Angela Shiflet's program at Wofford College and the Ralph Regula School of Computational Science in Ohio are both good examples of contemporary programs. While their organization, breadth, and depth may not be easy to replicate, they

have both published much of their curriculum and tools so that they can be used by others. Both programs are much broader and deeper than the materials required to support work in computer science alone, but they have each provided valuable resources for my own work and are cited throughout the curriculum modules and software laboratories.

My work both compliments and builds on Shiflet's textbook (Shiflet & Shiflet, 2006) and curriculum materials. Where her thrust is support for a mostly stand-alone program in computational science, mine is aimed at distributing a sub-set of that material within the computer science curriculum. While components of the textbook can be used in this way it's much more material than is required for anything but a full introductory course in computational science. The on-line materials she maintains at Wofford are a rich source of exercises which I have consulted widely during my work. A very nice example of her work, which includes a nice structure for organizing material like this, is her "Spreading of Fire" module. This exercise was one of the ACM's Special Interest Group on Computer Science Education's "Nifty Assignments" in 2007.

Another advantage to the "small chunks" method is the flexibility it offers faculty in terms of where the material is incorporated into their school's particular computer science curriculum. For example, some schools are adopting the breadth-first model, where a much wider range of topics is introduced in CS1; essentially giving students perspective on the full range of concepts and applications which modern computer science encompasses, rather than the traditional programming-only approach to CS1. In this context it may be appropriate to introduce computational science and modeling within CS1.

Support for this integrative approach to teaching computational thinking to undergraduate computer science students can also be found in Jeanette Wing's article "Computational Thinking" (Wing, 2006). In it she describes a vast array of computational techniques and tools with roots in computer science, but broadly applicable to problems from a wide array of disciplines. The article supports this notion with a veritable laundry list of examples from every corner of the academy.

It should be clear that the path to curriculum revision may vary greatly by institution, depending on the context there are many correct ways to change curriculum. My own observations indicate that the inclusion of modules into a variety of computer science courses is often not enough to support the wide ranging adaptation of computational and algorithmic thinking across the science disciplines more broadly at a given institution.

2.3.2 Identifying Curriculum Modules and Software Laboratories

There is a wide range of computational science material which could be included in this work which is more than could be reasonably considered by any but the most resource rich computer science departments. By consulting the sources described above and the textbook entries listed in the bibliographies, I was able to discern the patterns and commonalities between them, in some sense an application of the "wisdom of crowds" [2] principle. I have identified the most important subset of the material to organize and develop as part of this work. The *Curriculum Modules and Software Laboratories* chapter outlines the topics covered and sources used.

The software laboratories were identified using a number of criteria. First and foremost is that they are built on open source software and open protocols. This single attribute enables a number of important outcomes. It assures that cost is not a barrier to adoption. When appropriate, it allows the lesson to include "opening the hood" so that students can study how the science is done; this is particularly important when computational science is taught in a computer science context. Open source software generally has a long life span, if not as a community project then locally at a particular institution. This is important in an academic context where faculty value a stable, consistent interface to the tools they use for teaching. Lastly, it makes it possible for students and faculty to perform a very thorough verification of the software if desired; that is, does the software accurately calculate what it is being asked to calculate?

[2] `http://en.wikipedia.org/wiki/The_Wisdom_of_Crowds`

Software which is cataloged by, or a part of, projects such as the National Computational Science Education Reference Desk (CSERD), GNU, SourceForge and NetLib is generally more robust and more likely to have been reviewed by the community. Over the past 10 years, authors have begun to cite the software they used as part of their research in published papers, making it possible to use tools like Google Scholar and CiteSeer (the scientific literature digital library) to develop usage metrics for software packages.

In order to take this *in situ* "small chunk" approach, the curriculum and software materials have to be shaped in such a way as to make incorporating them into the existing computer science curriculum as frictionless as possible. This can be accomplished by looking for the natural overlap between computer science and computational science, and exploiting that whenever possible. For example roundoff and accumulated errors are a matter of particular concern for software that supports long-running simulations. By studying those topics in the context of an open source scientific software package students can see how different approaches to *e.g.* roundoff can directly effect the performance and accuracy of something as tangible as a weather system or protein assembly.

Chapter 3

Curriculum Modules and Software Laboratories

As explained in the literature review, rather than organize the materials into a traditional course structure I have chosen to break them down into smaller units called curriculum modules. Each of these is a largely freestanding unit that covers a logical collection of material. This organization has a number of advantages, most important of which is the flexibility it gives adopters in terms of incorporating the materials into a variety of undergraduate computer science courses.

In order to fully engage students the materials will be:

- highly interactive
- relevant to contemporary problems
- graphical where possible
- integrative across science disciplines where possible
- readily available

This approach to designing curriculum builds on both my own experiences and those of others as documented in (Peck, Murphy, Gray, & Joiner, 2005).

The value of using guided-inquiry methods in HPC and computational science is well documented by (Meyers et al., 2007). The curriculum modules and software laboratories follow those principles whenever possible. One way I support this approach is through the use of "software scaffolding" (Krajcik, Soloway, Blumenfeld, & Marx, 1998). As employed here, this technique suggests a framework which enables a student to first run a model, then open the hood to see how it is built with an eye towards improving it. This is made possible by using open source software as the basis for all of the software laboratories. By examining the model's implementation in software, the student can verify the correctness and then rebuild the model and experiment with modified or additional parameters and conditions. It should be noted that the use of open source software is required to support this approach; it cannot be done with closed source (AKA commercial) software.

Many people have studied the way undergraduate students learn. The most notable findings come from the American Association of Higher Education's *Seven Principles for Good Practice in Undergraduate Education*, first published in 1987 and augmented since then with related material for faculty and institutional development. A summary of the seven principles follows:

- encourage student-faculty contact
- develop student reciprocity and cooperation
- encourage active learning
- give prompt feedback
- emphasize the importance of spending time on a task instead of skimming it
- communicate high expectations
- respect diverse talents and ways of learning

In their 1996 article, Arthur Chickering and Stephen Ehrmann discussed how the seven principles could be applied to science, technology, engineering, and mathematics (STEM) education (Chickering & Ehrmann, 1996). Many of their suggestions are directly enabled

by the techniques discussed above, guided-inquiry methods and "software scaffolding". While it would be impossible to develop materials which would, for example, encourage student-faculty contact, the materials have been developed where possible to support these principles following the guidelines Chickering and Ehrmann describe for using them in the context of STEM education.

Many potential sources of curriculum materials were reviewed while building this collection. The principle ones were CSERD, Capital University, Wofford College, and the Krell Institute. Whenever possible an effort was made to "stand on the shoulders of giants" and adopt previously published, reviewed materials. In many cases this was not as practical as one might like. Curriculum material can have a short shelf life if it is not maintained with respect to evolving methodologies and software tools. In some cases it was possible to separate the content from the technology and reuse portions of the material. In others the work was able to serve as a useful pattern.

It is important that materials which teach scientific principles are as accurate as possible. While this is true of any pedagogical setting, it is particularly important in this context where we are teaching both the science and how science is done. Science, the systematic acquisition of reliable information about the natural world, requires one to build knowledge representations from accurate components. In the context of modeling and simulation (the core of computational science) the processes employed to ensure this accuracy are verification and validation (Trucano & Post, 2004).

Verification is the task of confirming the mathematical accuracy of the calculations and the error-free operation of the software. This is primarily a mathematical and computer science process which involves both inspection of the source code and analysis of the output. Another way to say this would be to ask "Is the problem solved correctly?".

Validation is the task of confirming that the models and simulations implemented are correct representations of the physical phenomena of interest. This is primarily an

experimental task, requiring direct comparisons between the equations and the real world. Another way to say this would be to ask "Is the correct problem solved?".

V & V, as they are frequently referred to, manifest themselves in four distinct ways in this dissertation and associated work products. In the literature review I discuss how the limits of models and modeling are important parts of the context of this work, these are also important components of the validation process.

In the curriculum modules *V & V* are addressed both in a standalone module of the same name and the whenever a model is encountered in a module. This allows the discussion of limits and applicability to take place in a particular context with actual conditions, data and results to examine. This provides a much richer environment for exploring the limits of a model, a particular implementation of a model, and issues surrounding the model's use in the development of public policy.

The fourth place that *V & V* will appear is in the final production of the materials. During that organization and rendering process a verification and validation pass will be performed performed on the curriculum modules and software laboratories. This will be guided by the methods described in (D. A. Joiner, Gordon, Lathrop, McClelland, & Stevenson, 2005) for performing verification and validation on materials to be submitted to digital libraries. This will make it straightforward to make entries for this material into a number of digital repositories, *e.g.* the Computational Science Education Reference Desk and Ensemble, both Pathways projects of the National Science Digital Library.

3.1 Curriculum Modules

These modules cover both background and specialized material which are applicable to a wide range of computationally-based approaches in the natural sciences, social sciences, humanities, and arts. In some cases there are modest dependancies between them, *e.g.* the

Validation and Verification module depends on concepts covered in the *Numerical Considerations* and *Statistics* modules. These dependencies are noted in the complete module descriptions.

1. Computational Science and the Scientific Method
2. Foundational Material
3. Validation and Verification
4. Modeling the Real World
5. Cellular Automata and Agent Based Systems
6. Monte Carlo Simulations
7. System Dynamics
8. Overview of High Performance Computing
9. Overview of Parallel and Distributed Programming
10. Scaling Software for Large Problems and Large Systems
11. Visualizing Quantitative and Spatial Information

Each curriculum module contains the following:

- Topics
- Software Laboratory
- Resources - print and online

The curriculum modules are better rendered in an interactive online form than print form, allowing people to easily download and incorporate the materials into their own teaching. This also makes revising and extending the materials and registering them with national curriculum resources such as CSERD much more practical.

A second consideration which supports online rendering is that some topics should be visited in different contexts, *e.g.* validation and verification and the limits of models more

generally. Rendering the materials online makes it easy to support a rich structure without duplicating the work of keeping the materials current.

To give readers of this dissertation a sense of the content of a curriculum module, one entire module has been included below, *Overview of High Performance Computing.*

3.1.1 Overview of High Performance Computing

Topics

- What is Supercomputing?
- Why Supercomputing?
- Hardware Taxonomy
- Nodes
- Central Processing Unit (CPU)
- Primary Storage, aka Main Memory or Random Access Memory (RAM)
- Secondary Storage, aka Disk Drives
- The Tyranny of the Storage Hierarchy
- Strategies for Effective Storage Utilization
- Network Interconnects
- Power, Cooling, and Monitoring
- Software Infrastructure

Software Laboratory

There are two components of this material which lend themselves particularly well to illustration through software laboratories: the effect of the storage hierarchy and the network interconnect on performance. Both of these can be explored using the *Benchmarking and Tuning HPC Platforms and Scientific Software* software laboratory. By changing the speed of the network interconnect one can clearly see the effects of bandwidth and latency (the two most important characteristics of the network interconnect) on the benchmark's performance. The centrality of the memory

hierarchy to overall performance can be illustrated by adjusting how much cache is used during the benchmark.

Resources

The first chapter of many parallel and distributed programming textbooks, *e.g.* "Parallel Programming in C with MPI and OpenMP" (Quinn, 2004) and "Parallel Programming" (Wilkinson & Allen, 2005), cover the majority of this material.

Robert Brown's online Beowulf cluster manual (Brown, 2008) contains a wealth of good information about the details of cluster design and deployment.

Henry Neeman's "Supercomputing in Plain English" materials (Neeman, 2009) are an excellent source of information on these topics. While he goes into more detail than is possible in this context he uses very accessible analogies to describe the material. Recently this material has been augmented with a series of videocasts, giving students the opportunity to see and hear the same material, see `http://www.oscer.ou.edu/education.html`

The remaining curriculum modules are organized similarly. See the chapter *Results and Future Work* for information about the different contexts in which these modules have been used to-date.

3.2 Software Laboratories

The software laboratories cover the tools and techniques of HPC and computational science. They are designed and distributed to be used either in conjunction with one of the curriculum modules listed above, as part of other related material, or as a standalone exercise. This gives potential adopters a variety of ways to use them in their own teaching context.

3.2.1 Linear Algebra Systems

- Software - Goto BLAS, Octave
- Disciplines - Mathematics, Computer Science

3.2.2 Calculating $1/\sqrt{x}$ in Software and Hardware

- Software - Performance counter interface, GNU Scientific Library
- Disciplines - Mathematics, Computer Science

3.2.3 Multi-scale Modeling

- Software - GNU Scientific Library, GNU Multiple Precision Library
- Disciplines - Computer Science, Physics

3.2.4 Benchmarking and Tuning HPC Platforms and Scientific Software

- Software - LINPACK, Goto BLAS, gprof, performance counter interface
- Disciplines - Computer Science, Statistics

3.2.5 Bioinformatics

- Software - MrBayes, MPIBLAST, NetLogo
- Sub-disciplines - phylogenetic reconstruction, genomics, population ecology.

3.2.6 Molecular Dynamics

- Software - Gromacs, PyMol
- Disciplines - Biology, Chemistry

3.2.7 Computational Chemistry

- Software - Gaussian, GAMESS, Tinker, MOPAC, PyMol
- Disciplines - Chemistry, Physics

3.2.8 Groundwater Flow Modeling

- Software - NetLogo
- Disciplines - Geoscience, Environmental Science

3.2.9 Climate Modeling

- Software - Weather Research and Forecasting Model (WRF)
- Discipline - Environmental Science

3.2.10 Topic Modeling

- Software - d2k, dl-topic
- Disciplines - Literature, Sociology

3.2.11 Behavior of Crowds

- Software - NetLogo
- Disciplines - Sociology, Economics

3.2.12 Visualization

- Software - POVray, gnuPlot, Google Earth, NCSA DataBridge and EasyViz
- Disciplines - Physics, Biology, Geosciences, Mathematics, Chemisty, Environmental Science, Computer Science

The software laboratories are published online in one of two ways. Tools that can be packaged in a web interface using *Web2.0* technology are the easiest type of tool to adopt in a wide variety of contexts. WebMO, a popular web front-end to computational chemistry packages, is a good example of this type of interface. A sample screen-shot from WebMO can be seen below in Figure 3.1. What you can't see in that image is how easily the molecule can be rotated in 3 dimensions, simply by clicking and dragging the molecule. When necessary, tools that must be run locally are packaged and distributed with clear instructions on how to install and run them under Linux, OS X, and, when possible, Windows clients. This distinction makes it clear how much easier *Web2.0* based tools are to adopt in a wide variety of educational settings.

To give readers of this dissertation a sense of the content of a software laboratory, one entire lab has been included below, *Molecular Dynamics.* This particular laboratory exercise has been used in the context of two undergraduate courses, "Scientific Computing" and "Parallel and Distributed Computing", and as part of a workshop on curriculum development for faculty from primarily undergraduate institutions. A significant amount of feedback was incorporated into it as a result of those experiences, what you see here is the current version of the laboratory.

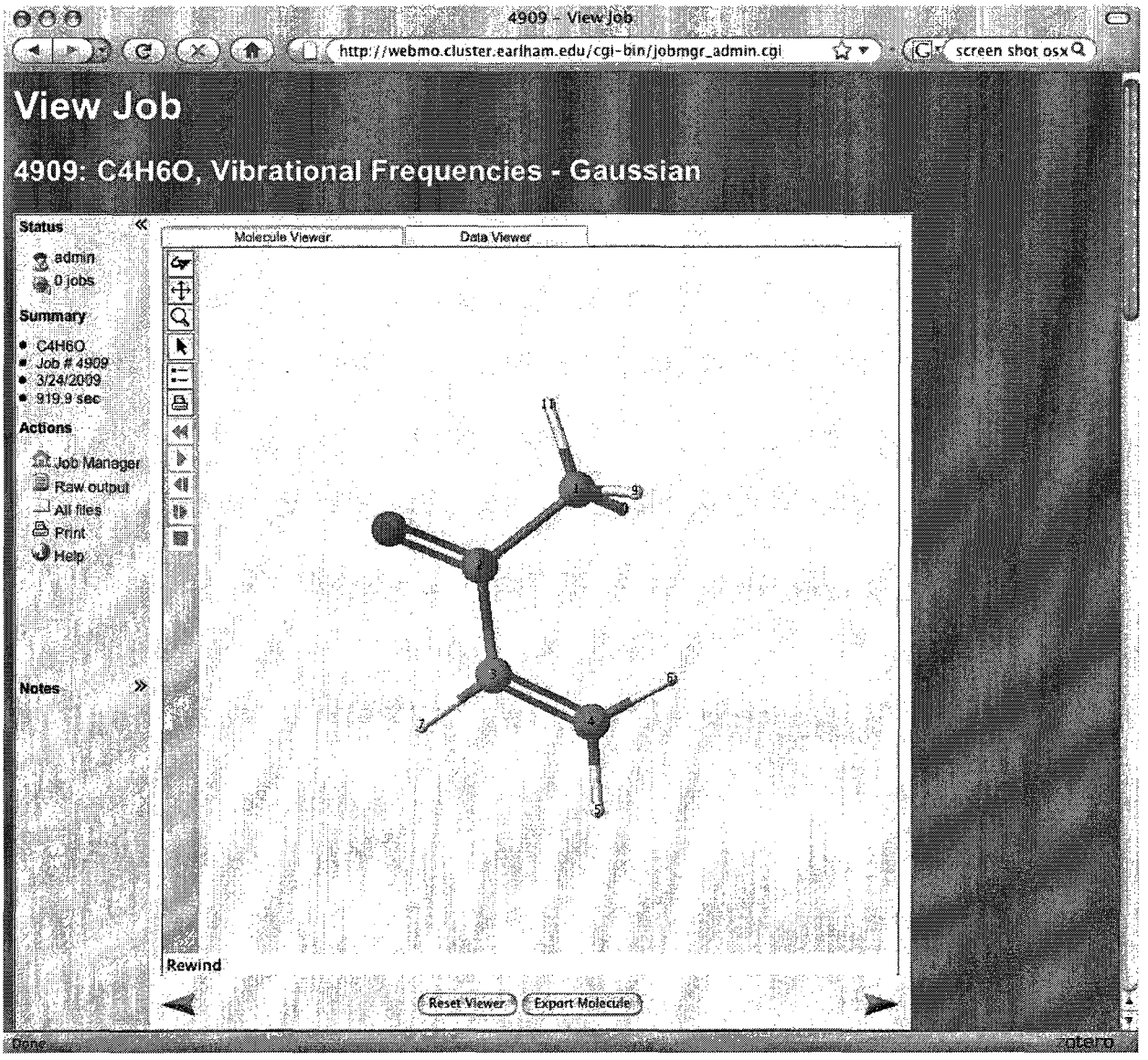


Figure 3.1: Viewing a molecule using WebMO within Firefox. The molecule can be rotated in 3 dimensions by clicking and dragging.

3.2.13 Molecular Dynamics

Overview

This document describes the steps associated with using the Gromacs software package to do perform a molecular dynamics simulation of a simple protein. It also contains a fairly basic description of the science involved. If you need to learn more about the underlying mathematics,

physics, chemistry, computer science, and/or biology which support molecular dynamics, you should consult one or more of the resources found in the appropriate section of the *Bibliography.*

This document presumes you are using one of Earlham College's Cluster Computing Group's (CCG) computational resources or the Bootable Cluster CD to do your work. If not, you will need to translate the paths, command names, *etc.* to your environment.

Molecular Modeling

Molecular modeling is the study of molecular structure and function through model building and computational methods. Molecular dynamics is a computational method that solves Newton's equation of motion for all of the atoms in a molecule. At each point in time, six quantities are known for each atom; position x_p, y_p, z_p and force x_f, y_f, z_f.

Each simulated time step involves computing the forces on every atom and integrating them to update their positions. The forces are from bonds and electrostatic forces between atoms within a certain cutoff distance. The desired properties are usually obtained as statistical mechanical averages of the atom trajectories over many runs. The averages tend to converge slowly with the length of the simulation or the size of the molecular system.

Molecular dynamics simulations are very computationally intensive; until recently it has not been practical to compute more than a couple of nanoseconds for systems with relatively few atoms. Recent developments in cluster and distributed computing algorithms and hardware now make it possible to (relatively) efficiently and easily harness hundreds or even thousands of processors as part of a single ensemble simulation.

One of the more common uses of molecular modeling/dynamics is to simulate the self assembly of amino acid necklaces (long chain molecules) into proteins, *i.e.* protein folding.

Protein Folding

Proteins are the basis of how the human body builds many of the parts we are made from, *e.g.* enzymes, structural components, and antibodies. Proteins are made-up of amino acids. There are

8 essential amino acids which can only be obtained through diet and 14 more which are manufactured by the body. DNA contains recipes for arranging a specific collection of amino acids into an ordered chain, which will then fold into a particular protein. There are many, many possible arrangements of the 22 amino acids taken in varying numbers and orderings.

Protein folding is a good example of consilience between mathematics, physics, chemistry, computer science, and biology. Sequencing the human genome gave us blueprints for all of the amino acid necklaces, which in turn fold into proteins whose structure determines their function within the body. There are connections between the various disciplines at multiple levels.

Why study protein folding? The process is integral to all of biology, yet it remains largely a mystery. For example, when proteins mis-fold they may be the cause of diseases, *e.g.* Alzheimer's, ALS (Lou Gehrig's), and Parkinson's. Protein structure is also an important component of drug design, since potential docking sites must be examined. Protein structure is also a key component of gene therapy research.

Why study protein folding computationally? Folding proteins *in vitro* is time consuming and expensive compared to folding them in a simulation running on a computational system, *i.e. in silico.* Due to the very short time scales involved, *in vitro* only allows you to see the final protein conformation, not any of the intermediate conformations. In the past it was difficult to marshal enough computational resources to do this, but with the advent of commodity clusters, commonly known as Beowulf clusters (Brown, 2008), there are now enough compute cycles within reach of most researchers and educators to fold proteins and other large biomolecules efficiently.

The Software

Groningen Machine for Chemical Simulations (Gromacs, `http://www.Gromacs.org`) is an open source software package originally developed by Groningen University's department of Biophysical Chemistry. Gromacs simulates the forces and movements of atoms in molecular systems over time in a biomolecule such as a protein, *i.e.* molecular dynamics. Gromacs also works well with other large biomolecules such as lipids and even polymers.

The CCG maintains a number of Gromacs installs, you will probably want to use the most recent

version built with the Fastest Fourier Transform in the West (FFTW, `http://www.fftw.org`). If you are using the BobSCEd cluster, the binaries for this version of Gromacs are in `/cluster/bobsced/bin`, which should be in your default path.

There are a number of tools available for visualizing the results of a molecular dynamics run, either as a movie of the entire run or as an image of the final conformation. The CCG uses PyMol (W. L. DeLano, 2009) for most of our work; it is capable of both movies and still images. A Google search using "visualize molecular dynamics" will yield many other visualization software choices.

The Process

The molecular system used in this example, 2LZM, is a lysozyme from bacteriophage T4. The commands described below use particular force fields, etc. appropriate for that system. Depending on the molecular system you are working with, other choices may be more appropriate.

All of the Gromacs programs have built-in descriptions of their command line arguments, to see these type `<program-name> -h` at the command line. This is the best way to start learning about the many choices Gromacs offers. Common command line options, *e.g.* `-f`, are the same across all Gromacs programs. In the instructions each option is documented the first time it appears. The instructions that follow are patterned after (Lindahl, 2004).

Download the Molecular System

The Research Collaboratory for Structural Bioinformatics (RCSB) provides the Protein Data Bank at `http://www.rcsb.org`. Typing in 2LZM as the PDB ID at that site returns a reference to a particular molecule, Lysozyme. Selecting "Display File" and then downloading an uncompressed PDB yields a file on the local machine which gives coordinates of every atom in the molecule. You will need to upload that file to the CCG file system using `scp` or `ftp` from your local machine. The host name to use is `cluster.earlham.edu` along with your CCG username and password.

Convert to Gromacs File Formats

`pdb2gmx` will convert the description of the system from the PDB format to a full topology file and a coordinate file. The OPLS-AA/L force field is used.

```
$ pdb2gmx -f 2lzm.pdb -o 2lzm.gro -p 2lzm.top -i 2lzm.itp \
  -ff oplsaa
```

This command creates three files: `2lzm.gro`, the coordinates of the atoms in the molecule, `2lzm.top`, the topology, and `2lzm.itp`, the position restraint data. At each time step the simulation will compute the forces between each atom in the system. Option `-f` is the input structure, `-o` is the output structure, `-p` is the topology file, `-i` is the include file for the topology, and `-ff` is the force field.

Place the Molecule in a Box full of Water

For this simulation to take a reasonable amount of time, we restrict the size of the system by placing the molecule in a box. Only forces within this box are calculated. To restrict a cubic box to a size of 0.5nm to the box's edge from any atom in the molecule we run the command:

```
$ editconf -f 2lzm.gro -d 0.5 -bt cubic -o 2lzm-box.gro
```

Option `-d` is the distance between the molecule and the box and `-bt` specifies the box type.

At this point the box is empty except for the molecule. In order to create a more realistic simulation we fill the empty space in the box with the SPC 216 water model using `genbox` and output the resulting system in PDB format:

```
$ genbox -cp 2lzm-box.gro -cs spc216.gro -p 2lzm.top \
  -o 2lzm-solvated.pdb
```

Options `-cp` and `-cs` specify the protein and water input structures respectively.

Minimize the Energy in the System

Next we use energy minimization to remove overlapping atoms. This requires preparing the system for a short run with `grompp` and then running it with `mdrun`. First copy the configuration file `/cluster/bobsced/share/gromacs/2lzm-em.mdp` to your working directory. This file contains the parameters required for the different steps in preparing and running the energy minimization simulation. The `-np 1` parameter to `grompp` and `mdrun` specifies that a single process will be used.

Some of the command lines given below do not fit on a single line, `$` denotes the beginning of a command line. If there are one or more arguments without a leading `$` they are a part of the line above them.

Both `grompp` and `mdrun` have extensive command line options, this example uses a small subset of them. Option `-c` specifies the configuration for the input structure, `-v` turns on verbose mode, `-s` specifies the input topology file, `-e` specifies the input energy file, `-g` is the output log file and `-x` is the output trajectory file in compressed form.

```
$ cp /cluster/bobsced/share/gromacs/2lzm-em.mdp .
$ grompp -np 1 -f 2lzm-em.mdp -p 2lzm.top -c 2lzm-solvated.pdb \
  -o 2lzm-em.tpr
$ mdrun -np 1 -v -s 2lzm-em.tpr -o 2lzm-em.trr -c 2lzm-em.gro \
  -e 2lzm-em.edr -g 2lzm-em.log -x 2lzm-em.xtc
```

The next step further prepares the system by holding the target molecule stable but allowing the water to settle around it. This process is called position restraining. To do this you will need to copy the MD parameter file `/cluster/bobsced/share/gromacs/2lzm-pr.mdp` to your working directory. Again we will use `mdrun` to run the simulation once the system is prepared.

```
$ cp /cluster/bobsced/share/gromacs/2lzm-pr.mdp .
$ grompp -np 1 -f 2lzm-pr.mdp -p 2lzm.top -c 2lzm-em.gro \
  -o 2lzm-em-pr.tpr
```

```
$ mdrun -np 1 -v -s 2lzm-em-pr.tpr -o 2lzm-em-pr.trr \
  -c 2lzm-em-pr.gro -e 2lzm-em-pr.edr -g 2lzm-em-pr.log \
  -x 2lzm-em-pr.xtc
```

Run the Molecular Dynamics Simulation

Finally we are ready to run the full molecular dynamics simulation. This involves using a different set of configuration options for `mdrun` which are found in `/cluster/bobsced/share/gromacs/2lzm-md.mdp`. Again we will prepare the system for the simulation with `grompp`.

```
$ cp /cluster/bobsced/share/gromacs/2lzm-md.mdp .
$ grompp -np 1 -f 2lzm-md.mdp -p 2lzm.top -c 2lzm-em-pr.gro \
  -o 2lzm-em-pr-md.tpr
$ mdrun -np 1 -v -s 2lzm-em-pr-md.tpr -o 2lzm-em-pr-md.trr \
  -c 2lzm-em-pr-md.gro -e 2lzm-em-pr-md.edr -g 2lzm-em-pr-md.log \
  -x 2lzm-em-pr-md.xtc
```

Visualize the Results of the Simulation

In order to visualize the system that has just been simulated we use PyMol, which can view both the final conformation and animate the entire simulation run. To use the output of Gromacs with PyMol, we must first remove the water from the system and extract the protein coordinates, etc. in a form PyMol can read, `trjconv` does this for us. Option `-ol` specifies the output trajectory and `-nf` sets the filter length and the output interval.

```
$ g_filter -s 2lzm-em-pr-md.tpr -f 2lzm-em-pr-md.xtc \
  -ol lowpass.xtc -nf 10 -all
$ trjconv -s 2lzm-em-pr-md.tpr -f lowpass.xtc -o 2lzm-final.pdb
```

Select Group 1 to convert just the Lysozyme molecule with `trjconv`.

The easiest way to use PyMol is to install the appropriate package on your client machine (running OS X, X/Un*x, or Windows) and then download and view the resulting PDB file, `2lzm-final.pdb` in this example, on the client. Sample command line for most U*nix systems are:

```
$ pymol 2lzm-final.pdb
or
$ pymol -S 2lzm-final.pdb
```

The second command uses PyMol's `-S` stereo mode option, if your hardware supports stereo viewing.

A sample screen-shot from PyMol can be seen below in Figure 3.2. Unfortunately the motion the molecular system shows during playback of the simulation does not translate well to the printed form.

Where to Go from Here

Gromacs has many other programs which, for example, measure the energy in the system (`g_energy`) and calculate the root mean square displacement of the system (`g_rms`). The Gromacs manuals (van der Spoel, Lindahl, Hess, & Groenhof, 2006) and Erik Lindahl's article in Cluster World (Lindahl, 2004) are good sources of information about these and other Gromacs utility programs.

The amount of setup contained in this software laboratory is comparable to the other laboratories, although the particulars of the interfaces (*e.g.* command lines, web portals) vary widely. In general there is a data collection/preparation component, a simulation component, and an analysis/visualization component.

The remaining software laboratories are rendered similarly. See the chapter *Results and Future Work* for information about the different contexts in which these laboratories have been used.

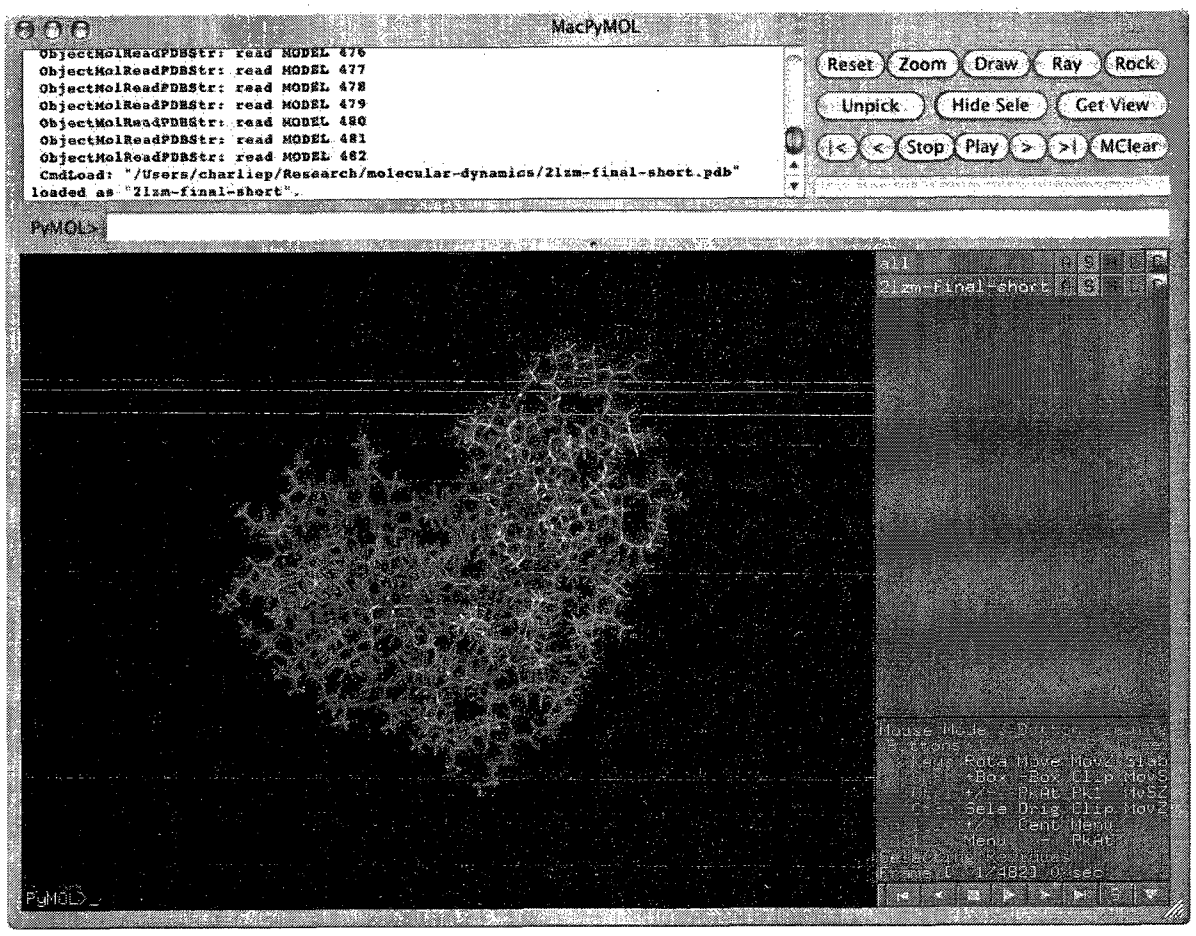


Figure 3.2: Viewing the simulation of a molecular system using PyMol under OS X. Controls are provided to *e.g.* play, rewind, step, skip, zoom, rotate and change the molecular annotations and form. Note that PyMol also provides a command line interface (in the lower left-hand corner) for use within the GUI.

Chapter 4

LittleFe

LittleFe is a complete multi-node Beowulf-style (Brown, 2008) portable computational cluster designed as an "educational appliance" for substantially reducing the friction associated with teaching high performance computing (HPC) and computational science in a variety of settings. The entire package weighs less than 50 pounds, easily travels via checked baggage, and sets-up in 5 minutes. Working with colleagues Paul Gray, Thomas Murphy, and David Joiner, I am jointly responsible for the design and primarily responsible for the engineering and production of *LittleFe*.

LittleFe's design grew out of our work building stationary clusters and our experience teaching workshops in a variety of places that lacked parallel computational facilities. Once we had some gear and some experience moving it around we worked through three different approaches before arriving at the system described here. The principle design constraints for *LittleFe* are:

- $3,000USD total cost
- Less than 50lb (including the Pelican travel case)
- Less than 5 minutes to setup
- Minimal power consumption; less than 100 Watts peak, 80 Watts average

The current production *LittleFe* design is composed of the following major components:

- 6 mainboards (Mini-ITX, 1–2GHz CPU, 512MB–1GB RAM, 100Mb/1Gb ethernet)
- 6 12VDC-ATX power supplies
- 1 320 Watt 110VAC-12VDC switching power supply
- 1 40GB 7200RPM ATA disk drive (2.5" form factor)
- 1 DVD/CD optical drive (slim-line form factor)
- 1 8 port 100Mb/1Gb ethernet switch
- 1 rack assembly
- 1 1610 Pelican travel case
- Fasteners, cabling, and mounting hardware

The $3,000USD cost per unit includes about 10 hours of student labor to assemble and test each unit. This includes liberating the Bootable Cluster CD image onto the disk drive and configuring the users. The mainboards, CPUs, and RAM comprise the bulk of the cost. With all 6 nodes idling, *LittleFe* draws about 80 Watts of power (about the same as an incandescent light bulb). When running a CPU-intensive molecular dynamics simulation using every node *LittleFe* draws about 88 Watts of power. See Appendix A *LittleFe - Parts Manifests* for a detailed parts manifests, cost estimates and sources.

4.1 Motivation

One of the principle challenges to computational science and HPC education is that many institutions do not have access to HPC platforms for demonstrations and laboratories. Paul Gray's Bootable Cluster CD (BCCD) project (P. Gray, 2004) has made great strides in this

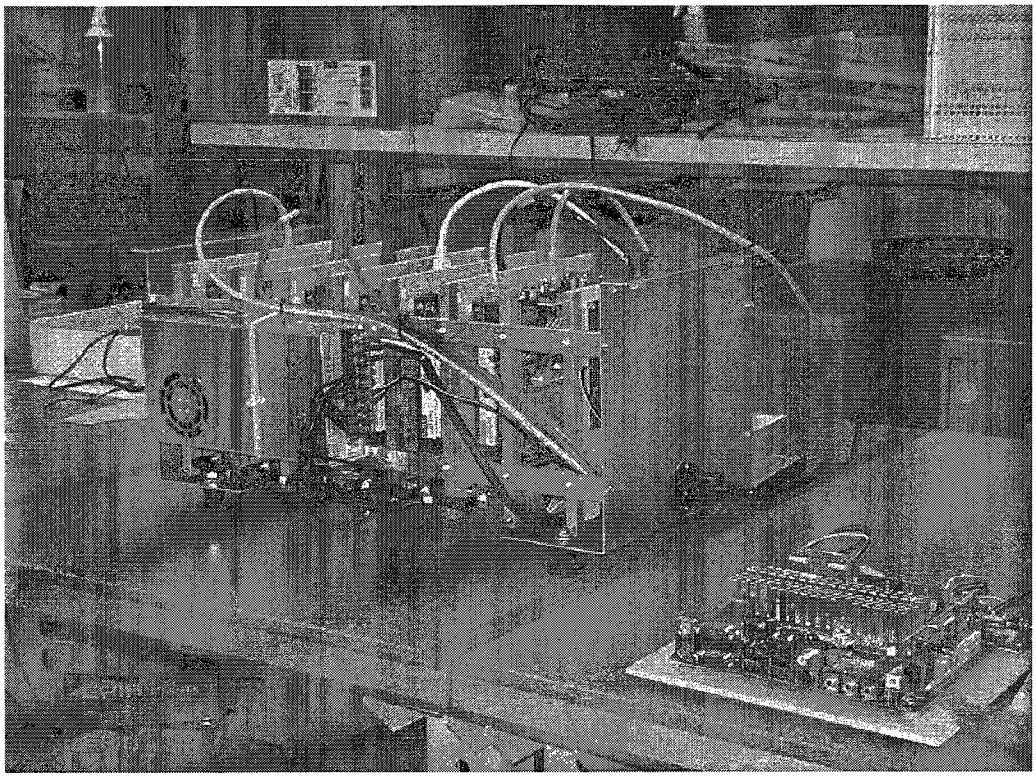


Figure 4.1: An early version 3 production unit almost ready for deployment.

area by making it possible to non-destructively, and with little effort, convert a computer lab of Windows or Macintosh computers into an *ad hoc* cluster for educational use.

LittleFe takes that concept one step further by merging the BCCD with an inexpensive design for an 6-8 node portable computational cluster. The result is a machine that weighs less than 50 pounds, easily and safely travels via checked baggage on the airlines, and sets-up in 5 minutes wherever there is a 110VAC outlet and a wall to project an image on. The BCCD's package management feature supports curriculum modules in a variety of natural science disciplines, making the combination of *LittleFe* and the BCCD a ready-to-run solution for computational science and HPC education.

LittleFe's principle edge is resource availability for computational science education. To teach a realistic curriculum in computational science, there must be guaranteed and predictable access to HPC resources. There are currently two common barriers to this

access. Local policies typically allocate HPC resources under a "research first, pedagogy second" prioritization scheme, which often precludes the use of "compute it now" science applications in the classroom. The second barrier is the capital and ongoing maintenance costs associated with owning an HPC resource. This affects most mid-size and smaller educational institutions and is particularly acute in liberal arts environments, community colleges, and K-12 settings.

While relatively low-cost Beowulf style clusters have improved this situation somewhat, HPC resource ownership is still out of reach for many educational institutions. *LittleFe's* total cost is less than $3,000USD, making it easily affordable by a wide variety of K-16 schools. This is particularly important for institutions which serve traditionally under-served groups; typically they have access to fewer technology resources than other schools.

LittleFe's second important feature is ease of use, both technically and educationally. Our adoption of the BCCD as the software distribution toolkit makes it possible to smoothly and rapidly advance from bare hardware to science. Further, we have minimized ongoing maintenance since both hardware and software are standardized. Paul Gray, from the University of Northern Iowa, and a number of our student research assistants have successfully maintained the BCCD for many years now via a highly responsive listserv and well organized web presence, `http://bccd.net`.

Portability is useful in a variety of settings, such as workshops, conferences, outreach events and the like. It is also useful for educators, whether illustrating principles in the K-12 arena or being easily passed from college classroom to college classroom.

4.2 Overall Design

The first *LittleFe* consisted of eight Travla Mini-ITX VIA computers placed in a nearly indestructible Pelican case. To use it one would take all the nodes, networking gear, power supplies, *etc.* out of the case and set it up on a table. Each node was a complete computer with its own hard drive. While this design met the portability, cost, and low-power design goals, it was overweight and deployment was both time-consuming and error-prone.

Successive versions of *LittleFe* have moved to a physical architecture where the compute nodes are bare Mini-ITX mainboards mounted in a custom designed frame, which in turn is housed in a Pelican traveling case. To accomplish this we stripped the Travla nodes down, using only their mainboards, and replaced their relatively large power supplies with daughter-board style units that mount directly to the mainboard's ATX power connector. These changes saved both space and weight. Current *LittleFe's* use disk-less compute nodes, only the head node has a disk drive. Removing seven disk drives from the system reduced power consumption considerably and further reduced the weight and packaging complexity.

4.3 Hardware

4.3.1 Mainboard

Basing our design around the Mini-ITX mainboard form factor standard has served us well. Currently there is significant demand in the industry for this size system, which yields rapid evolution, a significant number of choices from multiple vendors such as VIA, Intel, and Advanced Micro Devices (AMD), and consequently low price points.

Smaller boards such as the PC-104 form factor, while using less electrical power, lack the computational power to be useful. Larger boards, while offering much more computational

power, would be impractical in terms of both the physical packaging and electrical power consumption.

When specifying the mainboard, care should be taken to ensure that the overall height is less than the inter-board spacing in the frame (see the diagrams in Appendix B *LittleFe - Assembly Instructions.*

The head node should have a minimum of 1GB of RAM since it will be a file server for itself and all the compute nodes. Compute nodes should have at least 512MB of RAM.

4.3.2 Storage

All of the persistent storage devices are attached to the head node. Due to packaging and weight constraints the disk drive must be in a 2.5" form factor. The disk drive can be ATA or SATA: spindle speed, buffer size, and overall transfer rate are the most important criteria. Since there is only a single disk in each *LittleFe*, it should be a fast one. For most applications a 60GB disk drive is sufficient.

The speed of the CD/DVD is not particularly important as it is usually only used to load software. The CD/DVD drive must be a slimline form factor to fit with the disk drive on the frame.

4.3.3 Network Fabric

While 100Mb networking is sufficient for some applications, the availability and cost of Mini-ITX mainboards with 1Gb NICs makes them very attractive now for most new units.

Any small unmanaged 8–12 port network switch which uses 9–12VDC for line-in voltage can usually be mounted in the frame. Some *LittleFe* units sport a small WiFi access point, allowing a group of people to interact with a simulation from their laptops.

4.3.4 Power

120-240VAC line-input is brought to a frame mounted fused switch and then routed to a 320 Watt 12VDC regulated power supply. This minimizes the amount of high-voltage wiring in the system and provides the source for powering the 120 Watt daughter-board ATX power supplies located on each mainboard. With a peak draw of about 100 Watts, the primary power supply is generously sized for cool operation and increased reliability and longevity. For the head node, the daughter-board ATX power supply provides Molex connectors for the disk drive and CD/DVD drive.

4.3.5 Cooling

LittleFe's open frame, vertically mounted board design promotes a significant amount of natural cooling. This reduces the need for additional fans, further reducing the power consumption profile and the amount of system noise. *LittleFe* is quiet enough to use even when sitting in the arrivals lounge at an airport.

The Pelican traveling case, with the lid open and nothing packed around *LittleFe*, provides enough circulation that the unit can be run without removing it from the case. This is a particularly useful feature when using *LittleFe* in the field; for example when collecting, analyzing, and visualizing data from an attached water parameter probe.

4.4 Packaging

4.4.1 Frame

The frame is made of .080" smooth plate aluminum with punches for the rail mounts, case mounts, and line-input power switch. The rails and board guides are also made of standard

aluminum stock with pre-drilled mounting holes. See Appendix B *LittleFe - Assembly Instructions* for a diagram of the frame.

The mainboards are mounted on 1/4" AA luan plywood plates using 1/8" nylon standoffs. We explored many other materials for the mainboard plates, particularly aluminum and a variety of plastics and polymers. None could match the strength to weight ratio, cost, or ease of use associated with high quality plywood.

4.4.2 Traveling Case

The system is shock-mounted in a Pelican 1610 traveling case using a two-part rubber cup and plug system. The cups are mounted on the floor and lid of the Pelican case. The plugs are mounted on the top and bottom of each frame's end-plates. When you place the frame in the case the plugs nestle in the cups on the floor. When the lid is closed those cups encase the plugs on top. This gives the system support and shock resistance in each direction. We have tested this system extensively both in field trials and by examining the results of approximately 25 commercial flights where *LittleFe* travelled as checked baggage. While there is occasional distortion of an end-plate if an excessive amount of baggage is placed on top of the Pelican case, on the whole the system appears to functional adequately.

In terms of overall weight, *LittleFe*, traveling case, and any accessories, can be no more than 50lbs. This is the maximum allowed by airlines before heavy baggage surcharges apply. Practically, it is also about the maximum amount that most people can safely maneuver around an airport or school building. One advantage of the Pelican 1610 is that is has built-in wheels and a retractable tow handle.

4.5 Assembly and Testing

Assembling *LittleFe* consists of the following steps:

1. Assembling the frame and rails
2. Mounting the regulated 110/240 VAC power supply to the frame
3. Mounting the network switch and installing the network cabling
4. Mounting the mainboards to the cards
5. Mounting the power supplies and switches to the mainboards
6. Installing the up-link NIC on the head-node
7. Installing the mainboards in the cage
8. Cabling the power supplies
9. Mounting the disk drive and CD/DVD drive to the frame and installing the power and data cables
10. Plugging in the monitor, keyboard, and mouse
11. Performing the initial power-up tests
12. Configuring the BIOS on 5 of the mainboards to boot via the LAN and PXE

Basic hand tools: screwdrivers, pliers, wire cutters, adjustable wrench, drill, and a soldering iron are all the tools which are needed to fully assemble a unit. Most people budget a full day to do a complete assembly, test, and software installation. With practice, it has been shown that if nothing goes wrong, a single unit can be assembled in about 4 hours. See Appendix B *LittleFe - Assembly Instructions* for detailed step-by-step instructions and the URL of a video that illustrates the assembly of a unit.

4.6 Software

Early versions of *LittleFe* used the Debian Linux distribution as the basis for the system software. This was augmented by a wide variety of system, communication, and computational science packages, each of which had to be installed and configured on each

of the nodes. Even with cluster management software such as the C3 tools, this was still a time-consuming process.

One of the primary goals of this project has been to reduce the friction associated with using HPC resources for computational science education. This friction is made up of the time and knowledge required to configure and maintain HPC resources. To this end, *LittleFe's* system software was re-designed to use Paul Gray's Bootable Cluster CD distribution (P. Gray, 2004). The BCCD comes ready-to-run with many of the system and scientific software tools necessary to support a wide range of computational science education.

4.7 Status

Funding from TeraGrid, the SuperComputing Conference, and private sources has enabled my group to put about 15 *LittleFe* units into production as of this writing. *LittleFe* units are used in a variety of contexts: undergraduate computer science education at Earlham and other colleges and universities, K-12 science outreach and engagement programs, the SuperComputing Education Program's workshops and conference program, and the Dine'h Grid project of the Navajo Nation in Crownpoint, New Mexico.

LittleFe is very much a work in progress. Over the past three years, my colleagues and I have done extensive work in this area prototyping and testing fundamental design considerations, developing power and cooling solutions within a narrow design envelope, and porting and developing software laboratories for education, outreach, and training. As Moore's "Law" (Moore, 1965) continues to hold true, we reconsider design choices in an effort to make *LittleFe* smaller, cheaper, more powerful computationally, and lower in power consumption.

For more information about *LittleFe* see the chapter *Results and Future Work.*

Figure 4.2: A demonstration of *LittleFe* at the Oklahoma Supercomputing Symposium in 2006.

Chapter 5

Results and Future Work

5.1 Curriculum Modules and Software Laboratories

To date I have used material from this collection in the following courses in my teaching at Earlham College: Introduction to Computational Science, CS1, Scientific Computing, Parallel and Distributed Computing, Principles of Computer Organization, Operating Systems, and Software Engineering. Working to extend this material to other disciplines, I have begun to incorporate one module into a course on sustainable energy systems.

On the main, my experiences using the curriculum modules and software laboratories have been very good. In some cases the students who used them the first time suffered more glitches than they should have but in all cases the work was completed and the learning objectives largely realized. Like any curricular materials they improve with successive use. This is particularly true of the "scaffolded learning" aspect, almost every time the materials are used one or more students ask questions or pursue lines of thought which haven't been explored previously. Often these observations lead directly to a new or revised component in the material.

Maintaining a current relationship between the instructions and the particular version of

the software used in the laboratories can be time-consuming. The current state of affairs is such that whenever a minor or major release of a given piece of software changes someone has to step through the instructions point-by-point to make sure no changes are required. If the instructions can be provided at a somewhat more generic level this lock-step interdependency can be relaxed somewhat. The challenge is to find the appropriate balance between specificity and maintainability.

In 2006 a group of science colleagues at Earlham College were funded by the Keck Foundation to develop multidisciplinary units for a wide variety of science classes. These units are designed to be both environmental and computational in nature. Our work is organized around the notion of metals in the local environment *i.e.* in the water supply, structures, and soil. The materials developed for this dissertation proved useful as a source of potential tools and techniques. The notion of "small chunks" and embedding computational material *in situ* within existing classes has been extended to cover classes in chemistry and biology. During the last year of the grant our group will be organizing a workshop for faculty who would like to use this approach at their own institutions.

Many of the materials presented in this dissertation were developed in the context of the SuperComputing Conference's Education Program (`http://sc-education.org`). The SC Education Program's summer workshops for undergraduate faculty reach over 200 faculty each season with materials designed to cover a wide range of high performance computing (HPC) and computational science topics. As the curriculum and tools we use for the workshops and ongoing support operations develops, I will continue to contribute materials from this collection and adopt new ones as appropriate. Recently our efforts were recognized by a group of executives from Intel, IBM, and Sun. They are organizing a fresh look at the national computer science curriculum in conjunction with the professional organizations of the discipline, the ACM and the IEEE Computer Society.

5.2 LittleFe

In January of 2006 *LittleFe* was given honorable mention by the Krell Institute as part of their annual "Undergraduate Computational Engineering and Sciences" (UCES) award program. Krell describes the program as *"The UCES award program was created to promote and enhance undergraduate education in computational engineering and science. The goals of the program are to encourage further development of innovative educational resources and programs; to recognize the achievements of CES educators; and to disseminate educational material and ideas to the broad scientific and engineering undergraduate community."*

The SuperComputing Education Program and TeraGrid, the nation's open scientific discovery infrastructure, funded the production of ten *LittleFe* units during 2006-2007. These units have been placed in settings as diverse as an urban after-school program in Washington, D.C. to the Navajo Technical College on the Crownpoint Reservation in New Mexico. By using the Bootable Cluster CD (P. Gray, 2004) Linux distribution, these units are capable of running all of the software laboratories described in this work. In this context they are used to support outreach and workshops.

During the summer of 2008, four more units will be constructed to support one aspect of Dine'h Grid, a project to bring Internet service to all of the schools on the Crownpoint Navajo Reservation. This will enable the Navajo schools to include modern science in a variety of grade levels across their reservation, something not possible today.

A couple of projects have paid their respects to us by basing their own work on *LittleFe's* core principles. The most well known of these is Joel Adam's *Micro Wulf* project at Calvin College (Adams, Brom, & Layton, 2007). Joel's group made the decision to trade portability for compute power, *Microwulf* is much more computationally capable than *LittleFe*, but transportation is difficult and power consumption a concern for certain venues. Stationary operation also saves a significant amount of cost, *Microwulf's* price tag

is about $1,500USD. Microsoft recently published documents (Morgan, 2009) describing a low-power, work-load adaptive cluster which uses many of the same design techniques as *LittleFe*. During a conversation with one of their people at a conference I learned that they had in fact studied our materials while doing their design work.

LittleFe has appeared in many of the courses I teach and as part of the sustainable energy display in Dennis Hall at Earlham College. In classes I frequently use it to explain how computers and computing work generally, in CS1 for example it shows-up a couple of times. Being able to quickly power-down and remove boards to (carefully!) pass around the room makes it possible to really engage introductory students. For upper-level classes *LittleFe* units are used as the basis of projects in parallel and distributed computing and operating systems. In all cases the accessibility of the inner-workings in conjunction with the quick setup and use make it popular with students.

5.3 Future Work

Once this work has been approved, the curriculum modules and software laboratories will be published at a stable URL for public review and consumption. My previous experiences with maintaining software interfaces and curriculum material for high performance computing and computational science workshops for faculty have taught me a number of lessons that will prove valuable here.

After publication and additional use in the field, the curriculum modules and software laboratories will be given a second pass through the verification and validation processes (*V&V*). Verification confirms if the calculations and equations which make-up the model are implemented correctly. Validation seeks to insure that the models are correct representations of the physical phenomena they seek to represent. (See the *Curriculum Modules and Software Laboratories* chapter for a more complete description of this process.)

Once V&V is completed, the appropriate modules and laboratories will be submitted to the Shodor Foundation's Computational Science Education Reference Desk (CSERD, `http://shodor.org/cserd`), one of the Pathways projects of the National Science Foundation's National Science Digital Library (NSDL, `http://nsdl.org`). CSERD is the premier national dissemination channel for quality reviewed materials for computational science education. Another Pathways project which some of the materials would be appropriate for is Ensemble.

While reviewing materials for this dissertation, I was reminded of how temporal Internet based resources can be. Many of the items referred to in papers which were only two or three years old had become stale URLs and the material is no longer available. Every effort will be made to avoid that fate for the materials organized here. A significant portion of the materials are already in use within multiple contexts, insuring that they will be maintained and extended as underlying technology and pedagogical demands dictate. Publishing the materials in "source" form, permitting unrestricted adoption, and encouraging users to contribute their modifications back to the repositories will also help to keep the materials fresh and current.

Chapter 6

Bibliography

This bibliography is a bit longer than is typical for a similar work because I have included all of the supporting material for the curriculum modules and software laboratories. This is necessary to give a complete background for all the science, and those sources are also a potentially valuable resource for people adopting any part of my materials.

The bibliography is also published online with the curriculum modules and software laboratories. This allows people to immediately identify the subset of the materials germane to the unit they are considering. The bibliography is also published online *in toto* with an interface that permits filtering by keywords.

In the online version each bibliographic entry is tagged with one or more disciplines and one or more resource types. This makes it relatively easy to organize a large collection of materials into useful, focused subsets.

The list of discipline and technique tags includes:

- Biology
- Chemistry
- Computational Science
- Computer Science

- Economics
- Environmental Science
- Geoscience
- High Performance Computing
- Literature
- LittleFe
- Mathematics
- Modeling
- Physics
- Parallel and Distributed Computing
- Scientific Computing
- Sociology
- Visualization

Resource types are:

- Curriculum
- Pedagogical Methods
- Reference
- Software
- Support
- Textbook

References

Adams, J., Brom, T., & Layton, J. (2007). *Microwulf: Breaking the $100/GFLOP barrier.* Retrieved January, 2008, from `http://www.clustermonkey.net/content/view/211/1/`

Andersson, K., & Aronsson, D. (2001). *An evaluation of the system performance of a Beowulf cluster* (Tech. Rep. No. 2001:4). Uppsala, Sweden: Department of Scientific Computing, Uppsala University.

Apon, A., Mache, J., Buyya, R., & Jin, H. (2004). Cluster computing in the classroom and integration with Computing Curricula 2001. *IEEE Transactions on Education, 47*(2), 188–195.

Bell, G., & Gray, J. (2002). What's next in high-performance computing? *Communications of the ACM, 45*, 91–95.

Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., E. F. Meyer, J., Brice, M. D., Rodgers, J. R., et al. (1977). The Protein Data Bank: A computer-based archival file for macromolecular structures. *Journal of Molecular Biology, 112*, 535–542.

BEST: Building Engineering and Science Talent. (2004). *Higher education design principles to broaden participation in science, engineering and mathematics.* Retrieved December, 2008, from `http://bestworkforce.org/PDFdocs/BEST_BridgeforAll_HighEdFINAL.pdf`

Bonnett, J. (2007). *High performance computing: An agenda for the social sciences and the humanities in Canada* (Tech. Rep.). St. Catharines, Ontario, CA: Department of

History, Brock University.

Bradner, S., & McQuaid, J. (1999). *RFC-2544 - Benchmarking methodology for network interconnect devices.* Internet Engineering Task Force (IETF).

Bradner, S., et al. (1991). *RFC-1242 - Benchmarking terminology for network interconnection devices.* Internet Engineering Task Force (IETF).

Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn: Brain, mind, experience, and school.* Washington, DC: National Research Council and National Academic Press.

Briggs, W. (2005). *Ants, bikes, and clocks.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

Brown, R. G. (2008). *Engineering a Beowulf-style compute cluster.* Durham, NC: Physics Department, Duke University. Available from `http://www.phy.duke.edu/~rgb/Beowulf/beowulf_book.php`

Bruer, J. T. (1994). *Schools for thought: A science of learning in the classroom.* Cambridge, MA: MIT Press.

Burns, G., Daoud, R., & Vaigl, J. (1994). LAM: An open cluster environment for MPI. In *Supercomputing Symposium '94* (pp. 379–386). Toronto, Canada: IEEE Computer Society.

Burton, O. V. (Ed.). (2002). *Computing in the social sciences and humanities.* Urbana, IL: University of Illinois Press.

Bushey, D., & Stevenson, D. (2007). Problem-solving and critical thinking skills of experts and their importance in computer science education. In *TG07 - TeraGrid conference.* Indianapolis, IN: TeraGrid.

Chickering, A., & Ehrmann, S. (1996). Implementing the seven principles: Technology as lever. *American Association of Higher Education, 19*(10), 3–6.

Denning, P. J. (2007). Computing is a natural science. *Communications of the ACM, 50*(7), 13–18.

Distasio, J., & Way, T. (2007). Inclusive computer science education using a ready-made computer game framework. In *ITiCSE '07: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 116–120). New York, NY: ACM Press.

Dix, A. (1997). *Human-computer interaction.* Essex, England: Pearson Education.

Dongarra, J., London, K., Moore, S., Mucci, P., & Terptra, D. (2001). Using PAPI for hardware performance monitoring on Linux systems. In *Conference on Linux Clusters: The HPC Revolution.* Urbana, IL: Linux Clusters Institute.

Dowd, K., & Severance, C. R. (1998). *High performance computing* (2nd ed.). Sebastopol, CA: O'Reilly and Associates.

Einarsson, B. (Ed.). (2005). *Accuracy and reliability in scientific computing.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

Farian, H., Anne, K. M., & Haas, M. (2008). Teaching high-performance computing in the undergraduate college computer science curriculum. *Journal of Computing Sciences in Colleges*, *23*(3), 135–142.

Farrell, P., & Ong, H. (2003). Factors involved in the performance of computations on Beowulf clusters. *Electronic Transactions on Numerical Analysis*, *15*, 211-224.

Fosdick, L. D., Jessup, E. R., Schauble, C. J. C., & Domik, G. (1996). *An introduction to high performance scientific computing.* Cambridge, MA: MIT Press.

Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE special issue on Program Generation, Optimization, and Platform Adaptation*, *93*(2), 216–231.

Goedecker, S., & Hoisie, A. (2001). *Performance optimization of numerically intensive codes.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

Gray, J. (2003). *Distributed computing economics* (Tech. Rep. No. MSR-TR-2003-24). San Francisco, CA: Microsoft Research.

Gray, P. (2004). *The Bootable Cluster CD (BCCD).* Web site: `http://bccd.cs.uni.edu`.

Gray, P., & Murphy, T. (2006). Something wonderful this way comes. *IEEE Computing in Science & Engineering*, *8*(3), 82–87.

Gropp, W., Lusk, E., & Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, *22*(6), 789–828.

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI.* Cambridge, MA: MIT Press.

Gropp, W., & Snir, M. (1999). *MPI - The complete reference volume 2, the MPI extensions.* Cambridge, MA: MIT Press.

Harrison, P. (2001). *Computational methods in physics, chemistry, and biology.* West Sussex, England: John Wiley and Sons, Ltd.

Hess, B., Lindahl, E., & Spoel, D. van der. (2005). *GMX benchmarks: The Gromacs benchmarking suite.* Web site: `http://www.gromacs.org/benchmarks`.

Huelsenbeck, J. P., & Ronquist, F. (2001). MrBayes: Bayesian inference of phylogeny. *Bioinformatics*, *17*, 754–755.

Intel Corporation, Performance Computing Group. (2007). *Intel MPI benchmark.* Available from `http://www.intel.com/cd/software/products/asmo-na/eng/307696.htm`

Joiner, D., Gray, P., Murphy, T., & Peck, C. (2006). Teaching parallel computing to science faculty: Best practices and common pitfalls. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* New York, NY: ACM Press.

Joiner, D. A., Gordon, S., Lathrop, S., McClelland, M., & Stevenson, D. (2005). Applying verification, validation, and accreditation processes to digital libraries. In *ACM/IEEE-CS Society Joint Conference on Digital Libraries* (pp. 382–384). New York, NY: ACM Press.

Joiner, D. A., & Panoff, R. (2002). *Interactivate - On-line tools for teaching the principles of modeling and computational science.* Durham, NC: Shodor Education Foundation. Materials available at `http://www.shodor.org/interactivate`.

Joint Task Force on Computing Curricula. (2001). *Computing Curricula 2001: Computer science, final report.* New York, NY: ACM Press. Available online at

http://www.acm.org/education/curricula.html.

Joint Task Force on Computing Curricula. (2005). *Computing Curricula 2005: The overview report.* New York, NY: ACM Press. Available online at http://www.acm.org/education/curricula.html.

Kaplan, D. T. (2004). *Introduction to scientific computation and programming.* Belmont, CA: Brooks/Cole.

Karniadakis, G. E., & Kirby, R. M. (2003). *Parallel scientific computing in C++ and MPI.* Cambridge, England: Cambridge University Press.

Krajcik, J. S., Soloway, E., Blumenfeld, P., & Marx, R. W. (1998). Scaffolded technology tools to promote teaching and learning in science. *ASCD Yearbook: Learning and Technology.*

LAM/MPI Team. (2003). LAM/MPI user's guide (7.0 ed.) [Computer software manual]. Bloomington, IN: Open Systems Laboratory, Pervasive Technology Labs, Indiana University.

Landau, R. (2006). Computational physics: A better model for physics education? *IEEE Computing in Science & Engineering*, *8*(5), 22–30.

Landau, R. (2007). *A first course in scientific computing.* Princeton, NJ: Princeton University Press.

Larson, S. M., Snow, C. D., Shirts, M., & Pande, V. S. (2003). Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *Modern Methods in Computational Biology, Horizon Press.*

Liberal Arts Computer Science Consortium. (2007). *Model curriculum for a liberal arts degree in computer science.* New York, NY: ACM Press.

Lindahl, E. (2004). Parallel molecular dynamics: Gromacs. *Cluster World*, *2*(8), 8–18.

Lindahl, E., Hess, B., & van der Spoel, D. (2001). GROMACS 3.0: A package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling*, *7*, 306–317.

Linn, M. C. (1995). Designing learning environments for engineering and computer science. *Journal of Science Education and Technology*, *4*(2), 103–126.

Loan, C. F. van. (1996). *Introduction to computational science and mathematics.* Boston, MA: Jones and Bartlett.

Loeb, S., Lankford, H., & Wyckoff, J. (2002). Teacher sorting and the plight of urban schools: A descriptive analysis. *Educational Evaluation and Policy Analysis*, *24*(1), 37-62.

Maki, D., & Thompson, M. (2006). *Mathematical modeling and computer simulation.* Belmont, CA: Brooks/Cole.

Mayo, M. J. (2007). Games for science and engineering education. *Communications of the ACM*, *50*(7), 30–35.

Meyers, E., Gadde, H., Kurdzo, J., Daley, T., Vogt, J., Junod, R., et al. (2007). Integrating LEAD research in education. In *TG07 - TeraGrid conference.* Indianapolis, IN: TeraGrid.

Microsoft Research, Cambridge. (2005). *Towards 2020 science.*

Minnich, R. (2005). The ultimate Linux lunchbox. *Linux Journal.* (Article available at `http://www.linuxjournal.com/article/8177`)

Mooney, D. D., & Swift, R. J. (1999). *A course in mathematical modeling.* Washington, DC: The Mathematical Association of America.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, *38*(8).

Morgan, T. P. (2009). Microsoft plays with small, sleeply servers. *The Register*, *02*(25), 3.

National Academies. (2005). *Rising above the gathering storm: Energizing and employing America for a brighter economic future* (Tech. Rep.). Washington, DC: Committee on Prospering in the Global Economy of the 21st Century and the National Academy of Sciences and the National Academy of Engineering and the Institute of Medicine.

National Science Board. (2004). *Science and engineering indicators 2004.* Available online

at http://www.nsf.gov/sbe/srs/seind04/start.htm.

National Science Foundation. (2003). *Report of the advisory panel on cyberinfrastructure.* Available online at http://www.nsf.gov/publications key = cise051203. Washington, DC: National Science Foundation Press.

Neeman, H. (2009). *Supercomputing in plain English.* Retrieved April, 2009, from http://www.oscer.ou.edu/education.html

Ong, H., & Farrell, P. (2000). *Performance comparison of LAM/MPI, MPICH, and MVICH on a Linux cluster connected by a gigabit ethernet network* (Tech. Rep.). Kent, OH: Department of Mathematics and Computer Science, Kent State University.

Overton, M. L. (2001). *Numerical computing with IEEE floating point arithmetic.* Philadelphia, PA: Society for Industrial and Applied Mathematics.

Pande, V., Baker, I., Chapman, J., et al. (2003). Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Kollman Memorial Issue, Biopolymers*(68(1)), 91–109.

Panoff, R. (1995). The four A's of computational science education: Application, algorithm. architecture, and active learning. *IEEE Computing in Science & Engineering*, *2*(4).

Paprzycki, M. (2006). Integrating parallel and distributed computing in computer science curricula. *IEEE Distributed Systems Online*, *7*(2).

Pea, R. D. (2002). Learning science through collaborative visualization of the Internet. In *Virtual Museums and Public Understanding of Science and Culture.* Stockholm, Sweden: Nobel Symposium (NS 120).

Peck, C., Hursey, J., McCoy, J., & Pande, V. (2005). Building Internet distributed computing applications using existing scientific cores. *Dr. Dobbs*, *30*(11), 39-41.

Peck, C., Murphy, T., & Gray, P. (2005). Little-Fe: A portable, educational cluster. *HPC Wire*, *14*(42), 4-7.

Peck, C., Murphy, T., Gray, P., & Joiner, D. (2005). New directions for computational science education. *HPC Wire*, *14*(34), 3–6.

Pellergrino, J. W., Chudowsky, N., & Glaser, R. (2001). *Knowing what students know: The science and design of educational assessment* (Tech. Rep.). Washington, DC: National Academy Press.

Pettersson, M. (2005). *Performance counter support, PERFCTR.* Information available at the project web site `http://user.it.uu.se/mikpe/linux/perfctr`.

Pilkey, O. H., & Pilkey-Jarvis, L. (2007). *Useless arithmetic: Why environmental scientists can't predict the future.* New York, NY: Columbia University Press.

President's Information Technology Advisory Committee. (1999). *Report to the President: Information technology research: Investing in our future.* Washington, DC. (Available at `http://www.nitrd.gov/pitac/reports/index.html`)

President's Information Technology Advisory Committee. (2000). *Report to the President: Developing open source software to advance high end computing.* Washington, DC. (Available at `http://www.nitrd.gov/pitac/reports/index.html`)

President's Information Technology Advisory Committee. (2005). *Report to the President: Computational science: Ensuring America's competitiveness.* Washington, DC. (Available at `http://www.nitrd.gov/pitac/reports/index.html`)

Quinn, M. J. (2004). *Parallel programming in C with MPI and OpenMP.* Boston, MA: McGraw-Hill.

Rainey, D., Faulkner, S., Craddock, L., Cammer, S., Tretola, B., Sobral, B., et al. (2007). A project-centric approach to cyberinfrastructure education. In *TG07 - TeraGrid conference.* Indianapolis, IN: TeraGrid.

Roache, P. (1998). *Verification and validation in computational science and engineering.* Albuquerque, NM: Hermosa Press.

Shiflet, A. B., & Shiflet, G. W. (2006). *Introduction to computational science.* Princeton, NJ: Princeton University Press.

Sloan, J. (2005). *High performance Linux clusters.* Sebastopol, CA: O'Reilly and Associates.

Snell, Q. O., Mikler, A. R., & Gustafson, J. L. (1996). *NetPIPE: A network protocol independent performance evaluator* (Tech. Rep.). Ames, IA: Ames Laboratory, Iowa State University.

Society for Industrial and Applied Mathematics. (2001). Report of the working group on CSE education. *SIAM Review, 43(1)*, 163–177.

Spector, D. (2000). *Building Linux clusters.* Sebastopol, CA: O'Reilly and Associates.

Squyres, J. M., & Lumsdaine, A. (2003). A component architecture for LAM/MPI. In *Proceedings of the 10th European PVM/MPI Users' Group Meeting.* Venice, Italy: Springer-Verlag.

Stallman, R., et al. (2001). GNU coding standards [Computer software manual]. Boston, MA. (Included in the GNU `autoconf` distribution; see `ftp://ftp.gnu.org/gnu/autoconf`)

Stevenson, D. E. (1994). Science, computational science and computer science: At a crossroads. *Communications of the ACM, 137*(12), 85-96.

Swanson, C. D. (2003). *Computational science education* (Tech. Rep.). Ames, IA: Krell Institute.

Thompson, C., Ching, D., Gray, P., Helland, B., Meade, T., & Ragan, S. (2007). Computational literacy: Challenges in K–12 education. In *TG07 - TeraGrid conference.* Indianapolis, IN: TeraGrid.

Trucano, T., & Post, D. (2004). Verification and validation in computational science and engineering. *IEEE Computing in Science & Engineering, 6*(5), 8–12.

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2003). *A model curriculum for K–12 computer science: Final report of the ACM K–12 task force curriculum committee* (Tech. Rep.). New York, NY: ACM Press.

Tufte, E. (1983). *The visual display of quantitative information.* Cheshire, CT: Graphics

Press.

Tufte, E. (1990). *Envisioning information.* Cheshire, CT: Graphics Press.

Tufte, E. (1995). *Visual explanations.* Cheshire, CT: Graphics Press.

Tufte, E. (2006). *Beautiful evidence.* Cheshire, CT: Graphics Press.

Turner, D., & Chen, X. (2002). *Protocol-dependent message-passing performance on Linux clusters* (Tech. Rep.). Ames, IA: Ames Laboratory, Iowa State University.

Turner, D., Oline, A., Chen, X., & Benjegerdes, T. (2003). *Integrating new capabilities into NetPIPE* (Tech. Rep.). Ames, IA: Ames Laboratory, Iowa State University.

Turner, P. (2006). *Undergraduate computational science and engineering education* (Tech. Rep.). Philadelphia, PA: SIAM Working Group on CSE Undergraduate Education.

van der Spoel, D., Lindahl, E., Hess, B., & Groenhof, G. (2006). Gromacs user manual [Computer software manual]. (Available online at `http://gromacs.org`)

van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E., & Berendsen, H. J. C. (2005). Gromacs: Fast, flexible and free. *Journal of Computational Chemistry, 26*, 1701–1718.

W. L. DeLano. (2009). *The PyMOL molecular graphics system.* Retrieved January, 2009, from `http://www.pymol.org`

Wilkinson, B., & Allen, M. (2005). *Parallel programming.* Upper Saddle River, NJ: Pearson Prentice Hall.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Yasar, O., Rajasethupathy, K., Tuzun, R., McCoy, A., & Harkin, J. (2000). A new perspective on computational science education. *IEEE Computing in Science & Engineering, 2*(5).

Zellner, A., Keuzenkamp, H., & McAleer, M. (Eds.). (2001). *Simplicity, inference, and modeling.* Cambridge, England: Cambridge University Press.

Zyda, M. (2007). Creating a science of games. *Communications of the ACM, 50*(7), 26–29.

Appendices

Appendix A

LittleFe - Parts Manifests

A.1 Parts Manifests

The following parts manifests (Table A.1 and Table A.2) capture the v3 production series of *LittleFe*, *circa* 2007.

As with any reference design based on digital technology the particular details of the mainboards, disks, and networking components must be revisited every 6-12 months. While it creates extra work, overall this pace of change favors *LittleFe* in the long run since it serves to drive the performance up and the cost down. The basic framework remains constant while the technology it encloses continually evolves.

Table A.1: LittleFe v3 Computer Parts Manifest

Component	Part Number	#	Each	Per Unit	Source
Mainboard	VIA CN10000	6	173.00	1,038.00	Logic Supply
Memory	DDR2 533 memory 1GB	1	122.00	122.00	Logic Supply
Memory	DDR2 533 memory 512MB	5	64.00	320.00	Logic Supply
Power supply	Pico PSU 120W	1	49.00	49.00	Logic Supply
Power supply	Pico PSU 80W	5	39.00	195.00	Logic Supply
Frame	Aluminum ends and rails	1	100.00	100.00	Locally Supplied
Switch	D-Link DSS-8+ 10/100 switch	1	17.00	17.00	NewEgg
Power supply	MeanWell SP-320-12	1	90.00	90.00	PowerGate
Jumpers	1 per motherboard plus 1 uplink	7	2.00	14.00	Locally supplied
Disk drive	Travelstar 7K100 Hitachi	1	100.00	100.00	Directron
CD drive	Panasonic CW-8124-B CD/DVD	1	77.00	77.00	Logic Supply
NIC	Low-profile 10/100 PCI card	1	12.50	12.50	Logic Supply
Well nuts	Feet for the frame	8	1.65	13.20	Ace Hardware
Aluminum	1/2" x 1/2" angle, in feet	12	1.00	12.00	Ace Hardware
Retainers	Hitch pins	8	0.12	0.96	Ace Hardware
Standoffs	Nylon, mainboards and switch	28	0.12	3.36	Ace Hardware
12V Input	Lead, mainboard and switch	7	1.90	13.30	Mouser
110/220VAC	Line input and switch	1	14.00	14.00	Mouser
IDE-IDE	Motherboard to 3.5" IDE cable	1	10.00	10.00	Logic Supply
IDE-LPFF	Motherboard to 2.5" IDE cable	1	10.00	10.00	Logic Supply
Power control	Case front panel switch	6	10.00	60.00	Xoxide.com
Cards	Luan plywood mounting cards	8	0.50	4.00	Locally supplied

Table A.2: LittleFe v3 Traveling Case Parts Manifest

Component	Part Number	Quantity	Each	Per Unit	Source
Case	Pelican 1610	1	173.00	173.00	Commonly available
Cups	Case mounting	8	0.20	1.60	Ace Hardware

Appendix B

LittleFe - Assembly Instructions

B.1 Overview

Assembling *LittleFe* from a parts kit requires only basic knowledge of handtools and computer components. You will need large and small flat and philips screwdrivers, pliers and cable ties to complete the assembly. In addition to these illustrated instructions there is a narrated video of the assembly process available at `http://LittleFe.net`.

B.2 Hardware Assembly

B.2.1 Frame

The frame is assembled from the rails, card-edge guides, and end plates, see Figure B.1.

The rails and card-edge guides are assembled first. Layout the rails on a table lining-up the holes so that the ends with the shorter hole spacing are together and to the right. Lay the card-edge guides on top of the rails so that the large hole is facing away from you, see Figure B.2.

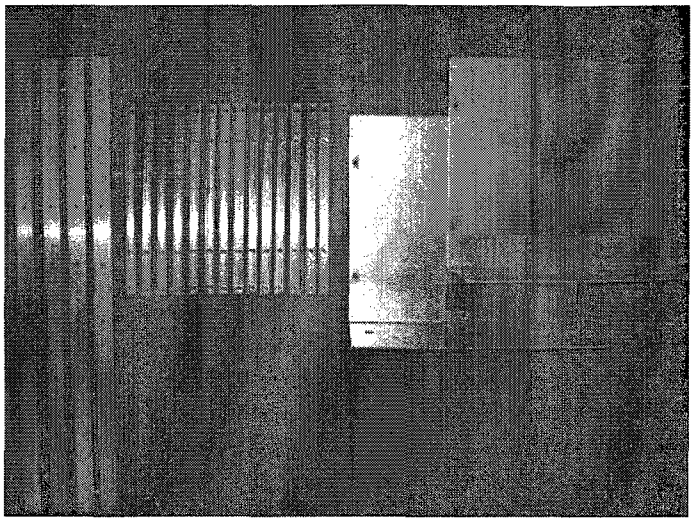


Figure B.1: Rails, card-edge guides, and end plates. Note that only one of the end plates is prepped for the 110/220VAC line input switch and the shorter spacing between the holes in the rails and the end of the rails at the bottom of the picture.

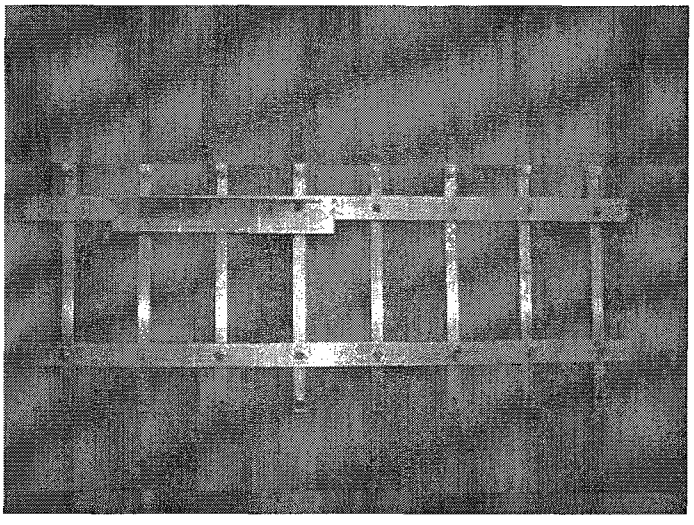


Figure B.2: Assembly of the first set of rails and card-edge guides. Note the shorter hole spacing to the right, the 110/220VAC regulated power supply mounting bracket, and the larger holes at the bottom of the fourth card-edge guide from the left.

Using the flat-head screws and nylon lock nuts mount the card-edge guides to the rails. The bracket which holds the regulated power supply should be placed on the left end of one assembly with the right-most mounting hole on the fourth card-edge bracket from the left. Two of the the card-edge guides have larger holes at the bottom, these should be placed

fourth from the left (this is where the cross-tie bar goes). Before tightening the screws be sure the rails are aligned properly, the easiest way to do this is to use a carpenter's square.

The second rail card-edge assembly is a mirror image of the first, that is the short spacing would be to the left on the table as you assemble it. This ensures that when combined with the end plates that the card-edge guides face each other, see Figure B.3

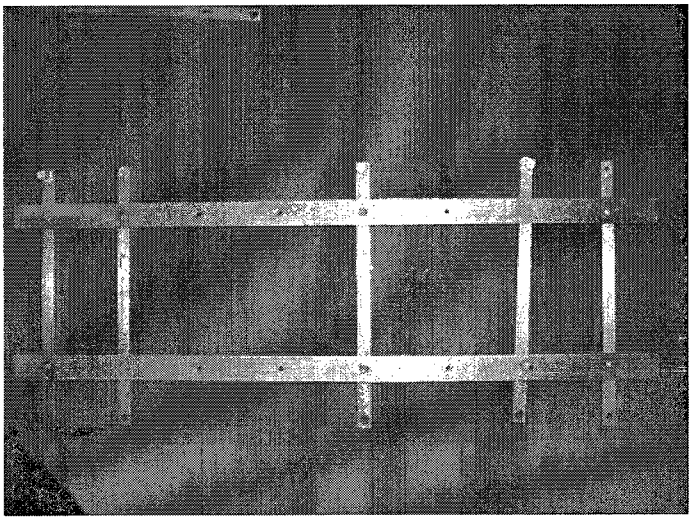


Figure B.3: Assembly of the second set of rails and card-edge guides. Note the shorter hole spacing to the left and the absence of the 110/220VAC regulated power supply mounting bracket.

Next mount the card-edge supports to each card-edge guides, these go in the large hole in the bottom of each card-edge guide, with the threaded end facing out, and are secured with nylon lock nuts. The fourth card-edge from the left on each rail set is for the cross-tie bar, see Figure B.4.

Next the end plates are assembled. This consists of mounting the rubber feet on the top and bottom and mounting the 110/220VAC line input switch. The smaller rubber feet go on the top and the larger feet on the bottom, see Figure B.5. The line input switch mounts with two screws, make sure that the switch is mounted such that it's accessible from the outside of the frame, see Figure B.6.

Now that the rail/card-edge assemblies and end plates are complete the frame can be

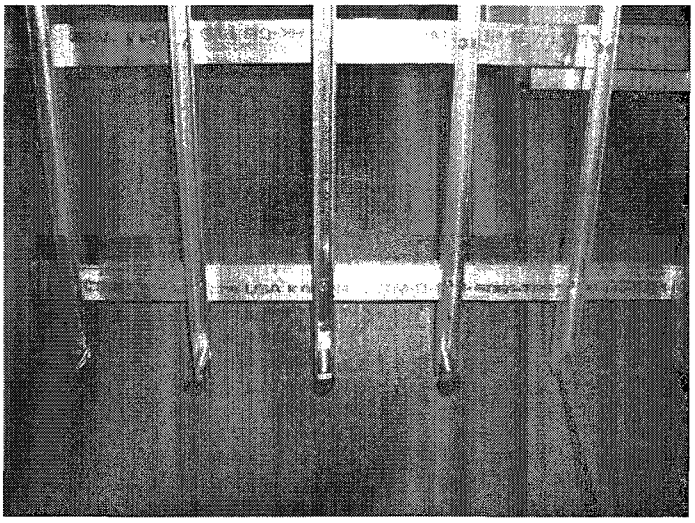


Figure B.4: Card-edge supports and the cross-tie bar mounted to the card-edge guides.

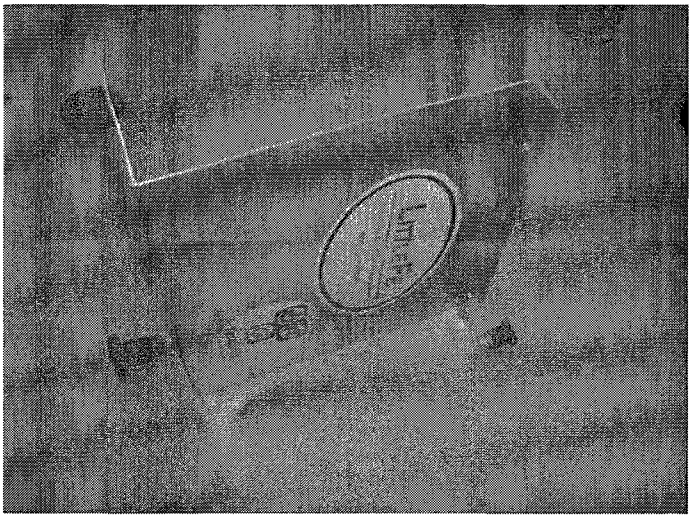


Figure B.5: End plate showing placement of small and large rubber feet.

assembled. Note that the end-plate with the 110/220VAC line input switch should be attached to the end of the rails with the longer hole spacing, see Figure B.7.

The frame is now ready for the installation of the high voltage wiring, low voltage wiring, and 110/220VAC regulated power supply.

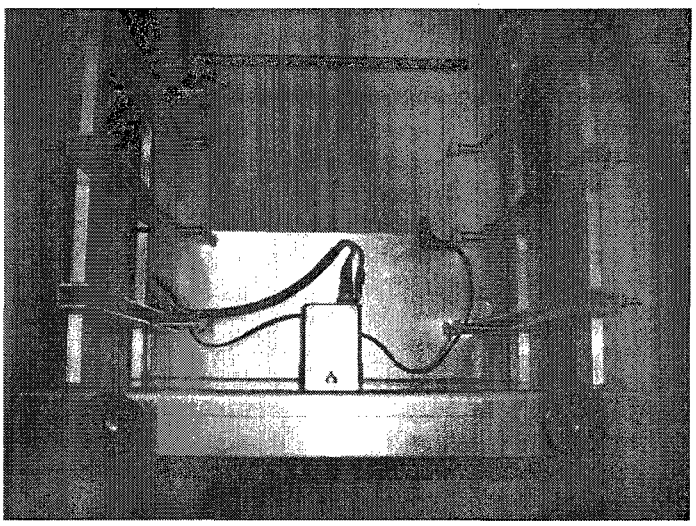


Figure B.6: End plate showing placement of 110/220VAC line input switch.

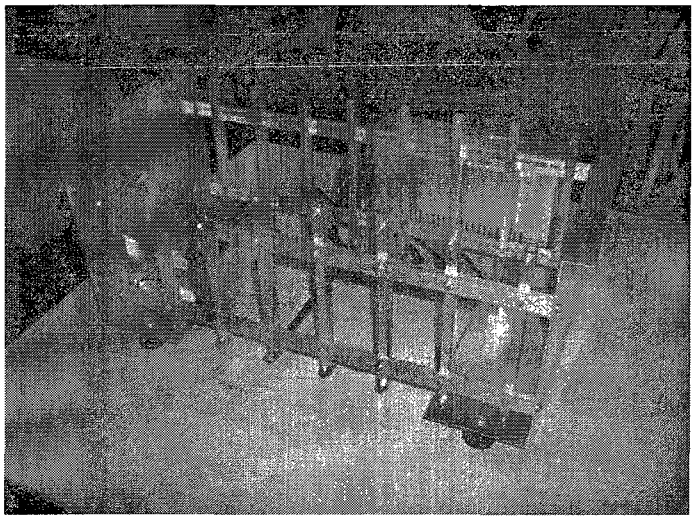


Figure B.7: Completed end plate and rail assemblies.

B.2.2 Wiring

The 110/220VAC regulated power supply is mounted to the bracket with two screws and lock washers. The end with the connection block should face to the right as you look at the frame from the outside, see figure B.8. The 110/220VAC power supply cable should be attached to the line input switch and then to the marked terminal locations on the regulated power supply. Take care to insure that load (white), neutral (black), and ground

(green) are all properly connected, see Figures B.9 and B.11. Routing and securing the 110/220VAC feed line can be seen in Figure B.10.

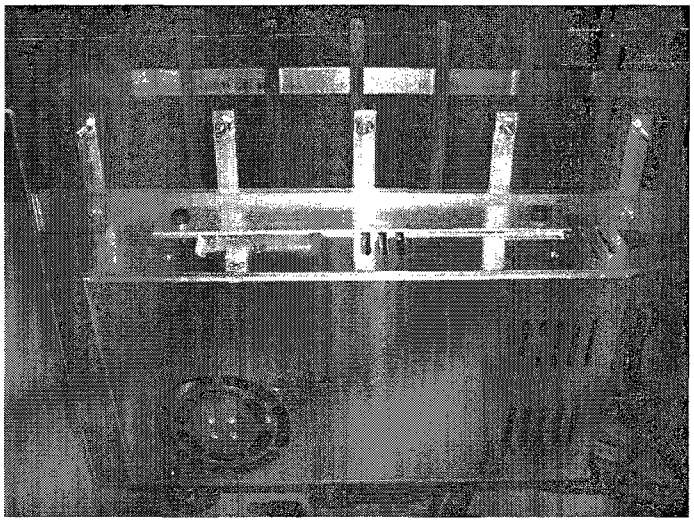


Figure B.8: 110/220VAC regulated power supply mounting.

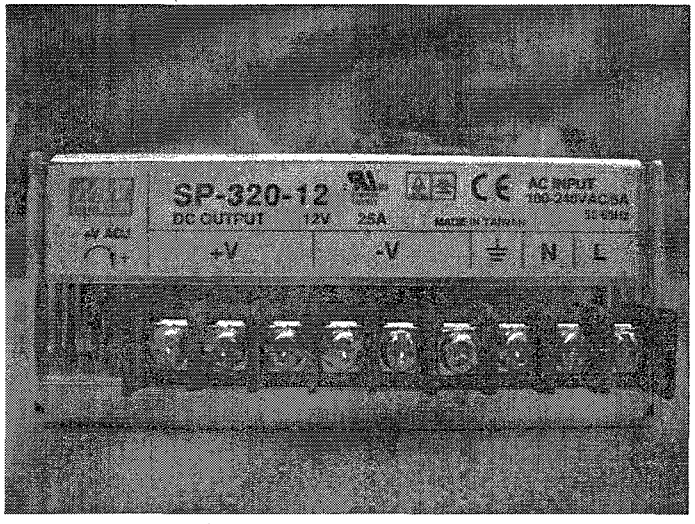


Figure B.9: 110/220VAC regulated power supply connection block showing ground, neutral, and load lugs (the three on the far right).

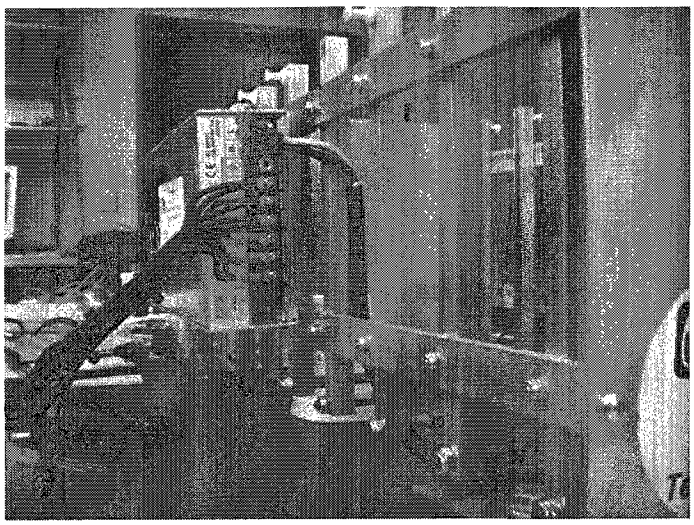


Figure B.10: Routing and mounting the 110/220VAC feed line.

B.2.3 Network

The network switch should be mounted to the card with the indicator lights facing up and the network connections facing down. Before inserting the network card into the frame the network jumpers should be installed starting with port 1, note that the cables are numbered, lf0 should go to port 1, lf1 to port 2, *etc.* See Figure B.11

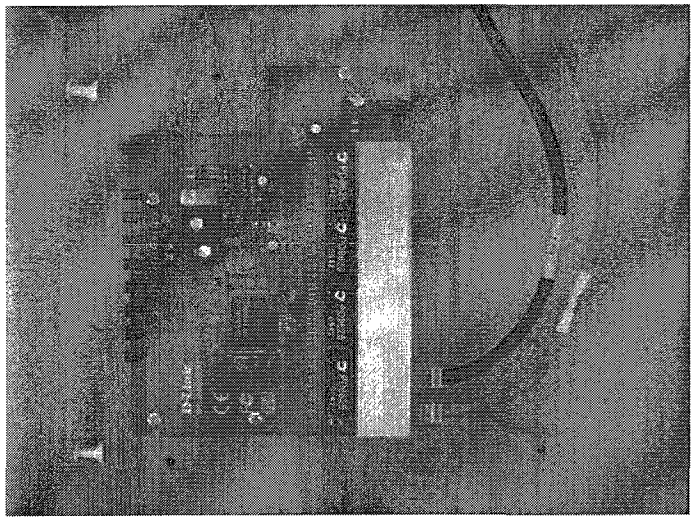


Figure B.11: Network switch mounted to the plate with the first two network jumpers installed.

Place the network card in the frame routing the cables underneath the rails in preparation for securing them to the rails as illustrated in Figure B.12. Secure the cables to each rail as shown in Figure B.13 with nylon cable ties. Take care to ensure that the network jumpers each have a smooth path with no sharp bends or kinks.

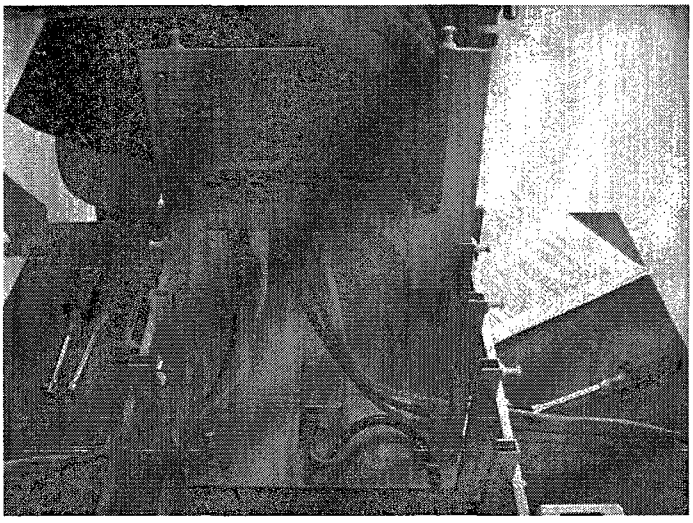


Figure B.12: Routing the network jumpers.

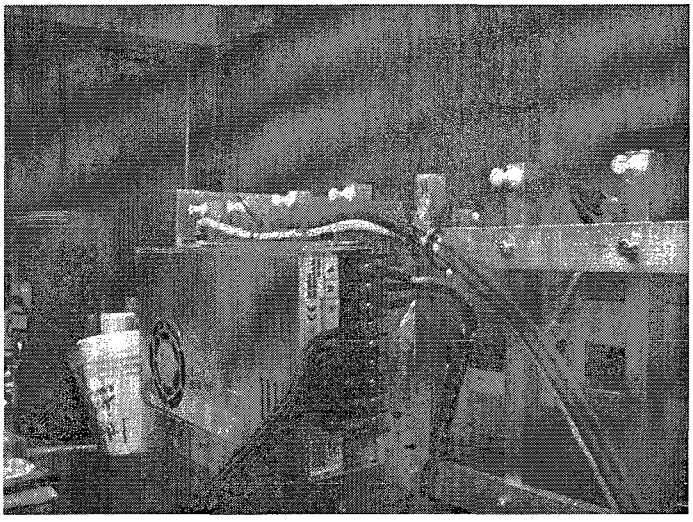


Figure B.13: Securing the network jumpers.

B.2.4 CPU and Disk Cards

The mainboards are mounted to the cards using bolts, 1/8" nylon spacers, and nylon lock nuts. The nuts should go on the card side, the bolt heads on the mainboard side. The corner opposite the power supply connector uses a 1/16" nylon spacer and flat metal washer in conjunction with the angle bracket, this provides a mount for 12VDC input connector, see figure B.14. Note that the nylon spacer should be in contact with the mainboard, then the angle bracket, then the flat metal washer which is in contact with the card.

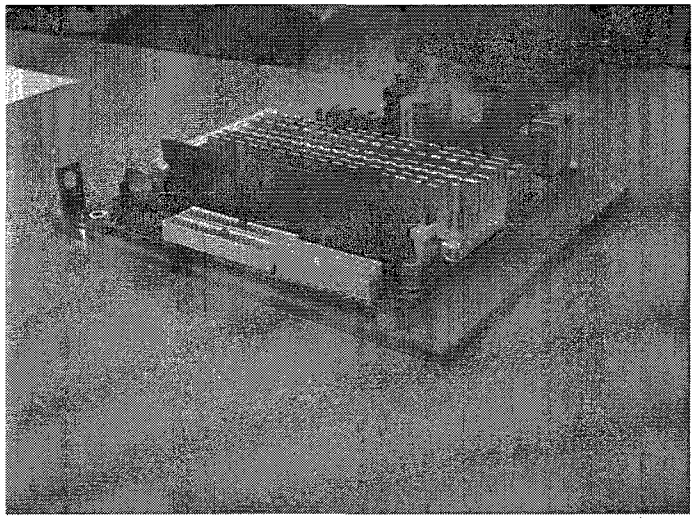


Figure B.14: The angle bracket which holds the 12VDC input feed is highlighted with a green laser on the left-hand side of the figure.

Once the mainboard is mounted to the card the on-board ATX power supply can be inserted into the mainboard connector and the 12VDC input connector can be mounted to the angle bracket, see figure B.15.

The power switch is mounted with a single screw and nylon lock nut to the top of the audio out/PS2 block on the front of the mainboard. The wires are routed around the heat sink and then the connectors are placed onto the header pins located on the back of the mainboard near the angle bracket, see figures B.16 and B.17.

The connectors from the power switch are labeled power switch, HDD LED, reset switch,

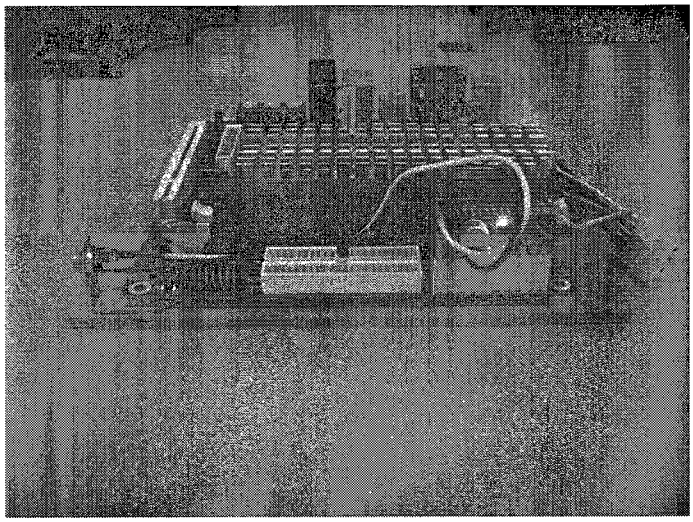


Figure B.15: The on-board ATX power supply is highlighted on the right with a green laser, the 12VDC input connector is on the left in the bracket.

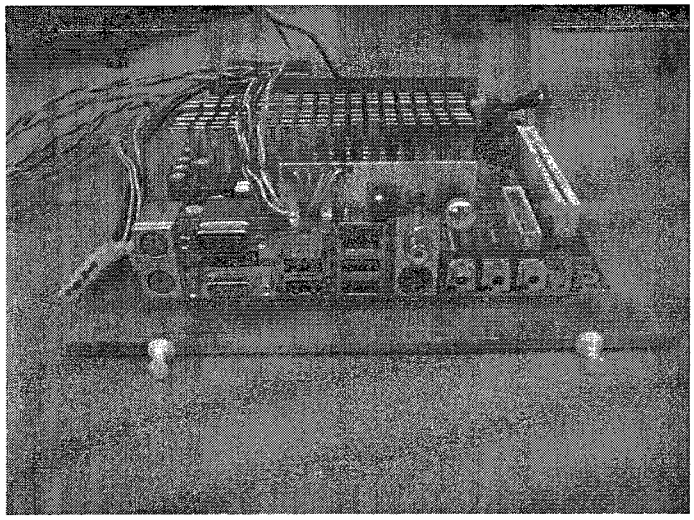


Figure B.16: The power switch mounted on top of the audio/PS2 block.

etc.. See the mainboard manual for a pin-out of the header to which they are connected.

The uplink NIC uses the PCI bus connector on the head-node's mainboard. After removing the RJ-45 socket bezel from the card install it. A very small cable tie can be used to secure the card from wiggling free of the PCI connector.

Cable management is done using nylon cable ties. Re-usable cable ties, *i.e.* ones with a release tab, facilitate easy dissassembly and reassembly which in turn makes it possible to

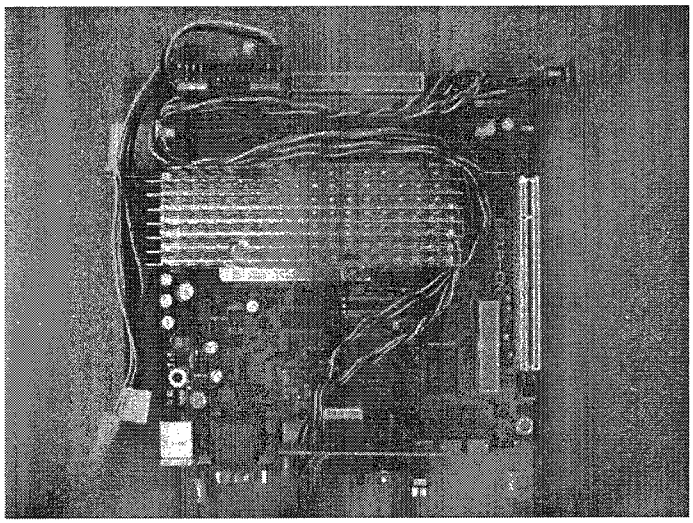


Figure B.17: Overhead view showing the routing of the power switch cables to the header pins located in the upper right-hand corner of the board near the 12VDC input connector.

use *LittleFe* to show students the inner workings of a computational system. The extra Molex connectors are secured along the side of the mainboard with one cable tie on each mounting screw. The wires for the power switch are included in the bundle at the rear of the board, see figure B.18.

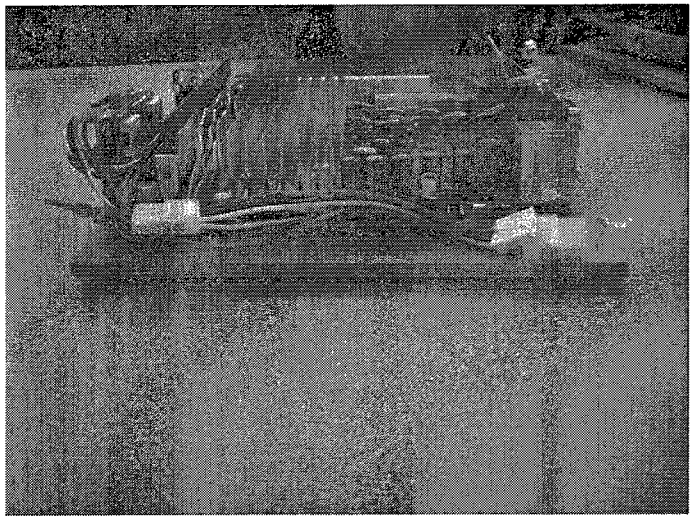


Figure B.18: Detail showing the cable management for the Molex connectors on the side of the mainboard.

Once the on-board ATX power supplies and power switches are mounted the five compute

mainboard cards can be installed in the frame, see figure B.19.

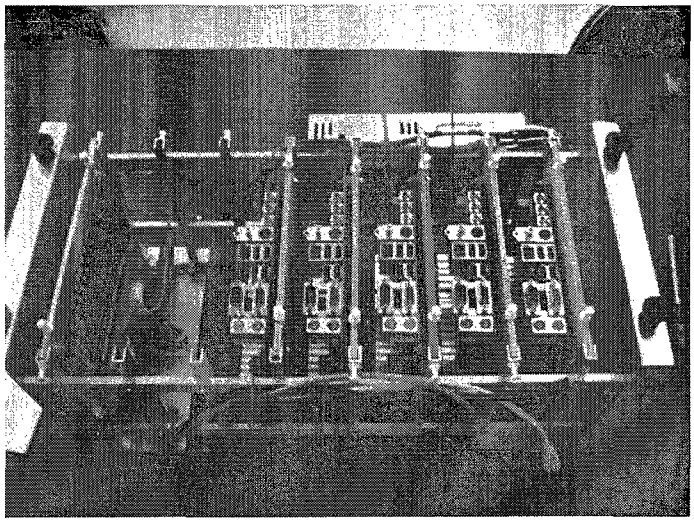


Figure B.19: The five compute mainboard cards installed in the frame.

The CD/DVD drive and disk drive are mounted to the drive card using O rings as illustrated in figures B.20 and B.21. Note the orientation of the drives, this is important for the cabling between the drive card and the head node mainboard card.

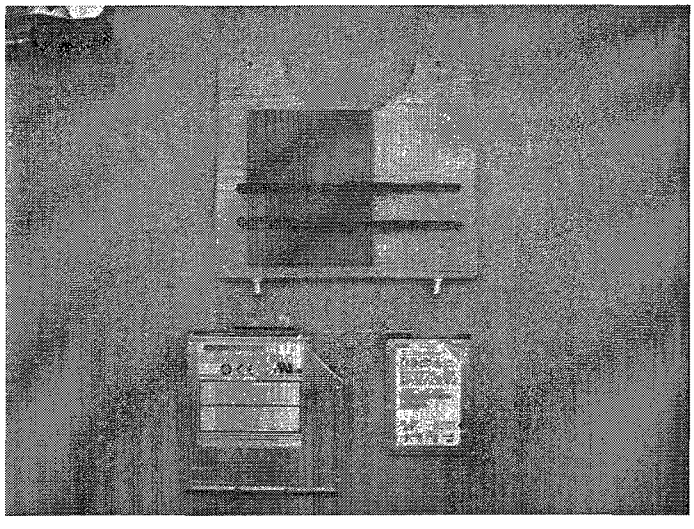


Figure B.20: The CD/DVD drive, disk drive, and disk card with the O rings mounted on it.

Install the drive cables to the drives and then to the head-node mainboard as illustrated in figure B.22.

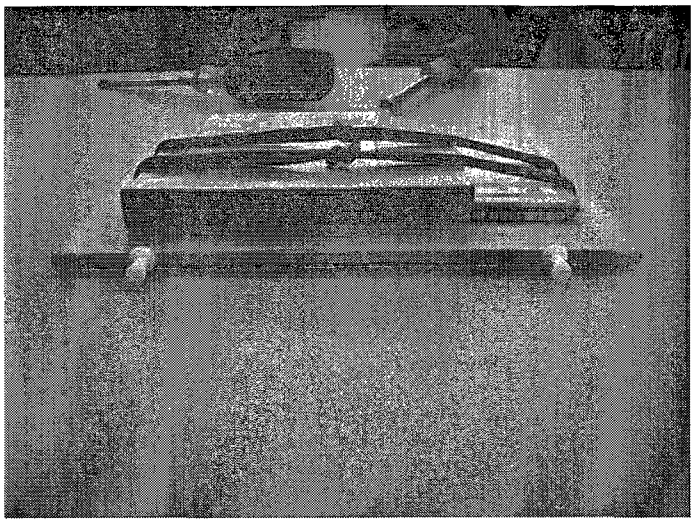


Figure B.21: The drives mounted on the disk card.

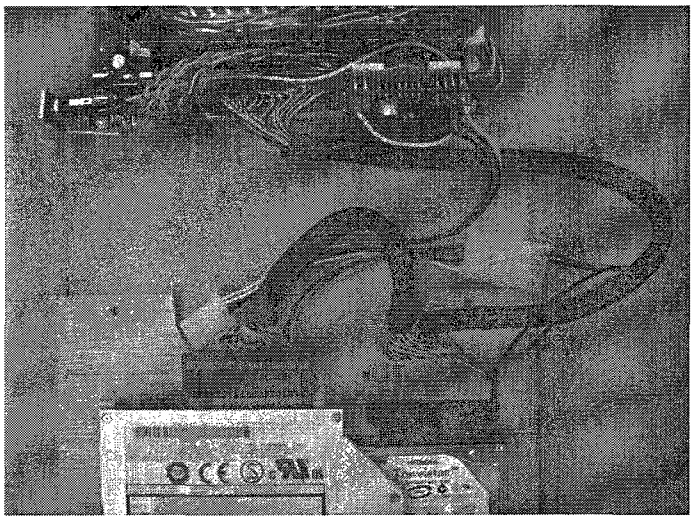


Figure B.22: Drive and power cabling for the disk drive card.

Once the drive card has been assembled and cabled to the head-node mainboard those two cards can be installed in the frame as illustrated in figure B.23.

B.2.5 BIOS Configuration

The five compute nodes should have their BIOSs configured to boot over the LAN using PXE. The head-node BIOS should be set to boot from the IDE disk first and CD/DVD

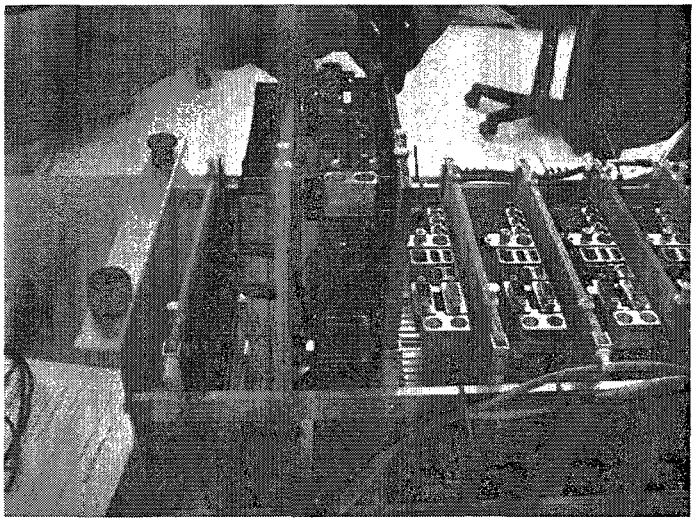


Figure B.23: Installing the head node mainboard card and disk card into the frame. Note the orientation of the disk card.

second. For testing and debugging purposes all of the compute node BIOSs should be set to boot from CD/DVD second.

B.2.6 Testing

A basic test of the system can be performed by booting from an ISO such as the BBCD. While this only tests the head node is does ensure that the disk subsystem is functioning correctly and prepares us for installing the BCCD in the next section. If you have access to a USB CD/DVD drive you can test each of the compute nodes in a similar fashion.

B.3 Software Installation

B.3.1 BCCD

LittleFe runs the Bootable Cluster CD (BCCD) image (P. Gray, 2004). The BCCD project is re-working their web presence as of January, 2009. Currently there are two sources of information and software for the BCCD, the 2.x codeline is available at `http://bccd.net`

and what is known as the NG release or 3.x codeline is available at `http://cluster.earlham.edu/trac/bccd-ng`. When the dust settles the URLs below will be updated with the correct, stable URLs for each codeline.

B.3.2 Liberation

Liberation is the process of taking the BCCD 3.x live ISO and installing the software image onto the disk drive attached to *LittleFe's* head node.

Instructions for performing an initial liberation and subsequent updates can be found at `http://cluster.earlham.edu/trac/bccd-ng/wiki/InstallInstructions`.

B.3.3 Testing

There are a small set of software package tests available for the BCCD 3.x codeline at `http://cluster.earlham.edu/trac/bccd-ng/wiki/Tests`. Users are encouraged to contribute tests for software packages they add to their BCCD installation. This is easy to do with *e.g.* Debian's `apt` environment.

B.3.4 Adding Functionality

Since the BCCD 3.x codeline is built on the Debian Linux distribution it's easy to customize your local installation of the BCCD with Debian's `apt` environment. More information is available with the `man` command.

If you would like to build your own live ISO based on the BCCD 3.x codeline with additional functionality or configuration information it's easy to do so. Step-by-step instructions for building from source are available at `http://cluster.earlham.edu/trac/bccd-ng/wiki/DevelopmentInstructions`.