Proposal to Investigate Software Tools for Atmospheric and Related Sciences

Cecelia DeLuca
Steven W. Hammond
Richard D. Loft

December 3, 1999

Project Summary

In this document we propose to undertake a four-pronged research and development program to identify utilities and software engineering techniques that will enable the creation or adaptation of high performance scientific software frameworks for the atmospheric and related sciences. We are motivated by the growing inter-disciplinary and multi-institutional nature of modeling efforts, the volatility of the high performance computer market, and the increasing complexity of parallel computing systems with multiple memory hierarchies. These three trends are increasing the complexity of model development and our research is focused on finding the means to manage it.

Recently, the Computation Science Section (CSS) has investigated the task of applying modern software engineering practices to the problem of developing high performance scientific software systems. We find software frameworks to be a notably successful and promising approach. Frameworks consist of modular, integrated software components that can be layered and allow a user to readily synthesize applications. Our preliminary finding is that high performance frameworks such as Overture and PETSc have been successfully created for specific domains. So far we see no reason to suppose that a workable software framework cannot be created similarly for modelers in the atmospheric and related sciences.

As a result of our investigations, we propose that the CSS undertake a four-pronged research and development program. This effort is complementary to efforts to create a next generation Climate Simulation Model (CSM) and responsive to the findings of the Code Assessment Panel. The four proposed CSS focus areas are:

- Assess and prioritize common software infrastructure needs of the NCAR community. This assessment will be a collaborative effort with the model development and algorithm research communities to ensure relevance. The assessment will identify candidate common software components and the specific type of functionality required. It will use demand for functionality and estimates of implementation costs to help prioritize the components. The assessment will result in a prioritized list of shared components, a set of draft API's and an estimate of the staff resources required to create them.

- Investigate systems of modular and reusable software components that have been developed elsewhere for applicability to our scientific discipline. Because of the cost of developing our own modeling software components, it is important to identify and evaluate existing software that suits our needs. As a means of evaluation, we propose implementing and profiling important algorithms within in a subset of these existing frameworks.

- Evaluate utilities for improved software reliability and collaboration. Such tools include version control systems, software distribution utilities and web based code repositories.

- Demonstrate, via a pilot project, that CSS can create a modular, high-performing, reliable modeling framework. A pilot project will provide a good testing ground for ideas and will demonstrate feasibility. We propose that the pilot project be the redesign of the related packages SPHEREPACK and FFTPACK into a form which can be incorporated into a framework. This choice is motivated by several factors. First, these packages have a very large user base. Second, they provide CSS a local focus to a general problem. Third, it removes CSS and SCD exposure regarding stewardship and modernization of all existing software assets.

Retasking the CSS to develop this expertise will require some reorganization of responsibilities within the section and the division. In particular, we suggest that current activities within CSS related to user code migration to message passing systems should be gradually transitioned to the Technical Consulting Group. We also believe that it makes sense to consolidate the maintenance and execution of benchmarks within CSS by forming a new full-time Software Engineer II Benchmark Librarian position. Finally, this project will require assistance from a software engineer with website design experience.

Contents

Project Description

1 Introduction

Modeling efforts at the National Center for Atmospheric Research (NCAR) have become broad inter-disciplinary and multi-institutional efforts. Current coupled climate models incorporate atmosphere, land, ocean, and sea ice components.[1] A proposed space weather model will extend from the surface of the earth to the surface of the sun.[2] At other institutions, a coupled watershed, mesoscale and general circulation model,[3] and models that integrate earth system models with economic models[4] are being developed.

Researchers face formidable political, scientific and technical challenges in coordinating these distributed collaborations. From a software engineering point of view, these challenges include software interoperability, software version control, validation, and testing, and data set integration. These areas must be addressed if such projects are to achieve even minimal scientific goals; i.e. get any simulations done at all, regardless of performance.

The current High Performance Computing (HPC) environment only increases the complexity of building complex scientific models. The HPC market is volatile, with no single dominant vendor. It is characterized by rapid reversals of fortune and fluctuating levels commitment to HPC by vendors. The prevalent architecture, a parallel cluster of shared memory multiprocessor nodes built with commodity parts, is perhaps the most complex computing system ever created. Multiprocessor nodes typically have deep memory hierarchies and complex caching schemes. The programming model is typically virtual shared memory[5] or a hybrid of shared and distributed memory[6]. Performance programming may require a careful combination of caching optimizations, shared memory directives, and message passing calls between nodes to achieve the best possible performance.

The diversity of architectural details such as cache sizes, on- and off-node bandwidths, latencies, and the availability of features such as efficient one sided message passing calls or the ability to overlap communication with computation, dictates a flexible software design to achieve performance portability. For example, tuning an application in response to these factors may involve changing the global data decomposition, local arrangement of data, method of parallelization or it may involve switching to a completely different set of numerical algorithms.

One approach to managing this complexity in models and machines is to base code on data abstractions which enable global data decomposition and local data placement to be modified; this may require a sophisticated software design strategy. Constructing modular code that readily supports the exchange of different model components is another fruitful approach. Some progress has been made on automatically tuning algorithms, but this work is still in its infancy [32]. In short, programming these machines for performance portability and modularity is often time-consuming and hard.

In the commercial sector, creating large, complex software systems has been a major driving force in the development of modern software engineering practices. It stands to reason that these practices should be considered part of a solution to the challenge of developing complex, high performance scientific models. The software engineering approaches that we believe are applicable cut across all areas of software development including tools, processes, structure, and standards:

---

[1] e.g., NCAR Climate Simulation Model (CSM) and Parallel Climate Model (PCM).

[2] A Prototype Implementation of a TeraFlop-Class Predictive Space Weather Model, 1999 NSF Knowledge and Distributed Intelligence (KDI) award, Univ. of Michigan/NCAR High Altitude Observatory.

[3] Multi-Scale, Multi-Resolution Modeling, NSF National Partnership for Advanced Computing Infrastructure (NPACI) thrust area, UCLA.

[4] e.g., The MIT Integrated Framework, part of the Joint Program on the Science and Policy of Global Change.

[5] e.g. SGI Origin 2000

[6] IBM SP Power3, Sun HPC

1. Follow formal software design processes. These processes include the production of formal design documents and periodic design reviews. Design processes encourage effective collaboration and tend to produce code that is well thought out, well designed, well documented, and well tested.

2. Modularize, generalize, and standardize code to promote reuse. Good modular design organizes code around the natural data structures, functions and functional hierarchies implied by a problem domain. Modules, or classes in object-oriented terms, are largely self-contained black boxes that hide internal detail, and thus complexity, from the outside world. Generalization makes modules likely to be useful in multiple contexts. Standardization of module interfaces promotes interoperability. Taken together, modular design, generalization and standardization promote code reuse. Code reuse makes efficient use of programmer resources and improves software reliability, since software is tested in a variety of situations.

3. Layer software. Layering is a way of organizing software on a larger scale than modularization. Lower level software modules are general in function and machine specific in implementation. Higher level modules are specific in function and machine independent in implementation. Layered software allows scientists to focus on high level applications, while software engineers, computer scientists and mathematicians focus on general lower level methods. This results in an efficient allocation of labor resources. Code written on top of layered software is more portable, since the machine specifics are localized within a managable set of software functions.

4. Use modern software development tools. These include common code repositories to encourage code sharing, versioning systems that enable collaborative development, issue tracking tools to collect and preserve information related to codes, automatic configuration tools to facilitate software installation and automated test suites to assist in regression testing. Collectively, software development tools increase staff productivity and improve code quality.

Not all of these techniques may be appropriate for our problem domain. We are aware, for example, that some object-oriented software systems have had performance problems. We have also identified frameworks as a promising mechanism to achieve large scale software integration. Part of the work proposed here will be a detailed analysis of other efforts to determine what has worked and what has not worked in related objected-oriented software design projects. We will then verify these results with our own pilot project.

We hypothesize that object-oriented software design and frameworks could ultimately provide a general modeling infrastructure for high performance scientific software development in the atmospheric and related sciences. We have identified four activities that will test our hypothesis which are described in the sections that follow. These are:

1. Understand the current and future scientific software needs of the modeling community and distill them into meaningful software component descriptions.

2. Investigate related efforts to create scientific software frameworks.

3. Experiment with tools that foster software development collaborations. Here, we intend to create a web-based code repository, as well as an NCAR-wide web forum to discuss software design, algorithms, interfaces, standards and related issues.

4. Develop a pilot project to test ideas by converting SPHEREPACK and FFTPACK functionality into a prototype parallel software framework, built on top of a utility layer that includes message passing.

The discussion of our proposed four-pronged research and development effort is preceded with some basic material on object-oriented software design and software frameworks. Finally, we end this proposal with a discussion of proposed tasks with staffing requirements and a timeline for this investigation as well as some conclusions and recommendations.

We believe that the tasks outlined in this report make great strides toward improving the quality of the software development efforts at NCAR, protecting our existing software assets, reducing the burden of model development through module code design and code reuse, and helping the institution to develop a long-term strategy for developing the needed complex models of the future. In addition, it is an element of a response to the NSF Code Assessment Panel[7]. Finally, the work being proposed here is well aligned with the recently released draft NASA Cooperative Agreement Notice from the HPCC program focused on "Increasing Interoperability and Performance of Grand Challenge Applications in the Earth and Space Sciences". Proceeding with this proposed research will favorably position NCAR to seek additional funding to help support this work.

2 Background

In this section we provide some background information and discuss some related efforts in object-oriented software design and frameworks. As part of the work proposed here we will examine these efforts in greater detail for applicability to our problem domain.

2.1 Object-Oriented and Object-Based Code Design

Object-oriented software design organizes data and methods into functional components to make the process of developing and maintaining large, complex software systems more manageable. In object-oriented programming, data, related attributes, and methods for manipulating the data are collected into modules or "classes." A specific case, or "instantiation," of a class is called an object. For example, a vector class might include a class attribute of length, and a class method called `get_length`. A particular instantiation of that class might be a vector object with a length of 5. To access the length of the vector, the vector method `get_length` would need to be called. A powerful feature of object-oriented programming is that it hides or "encapsulates" the internal complexity of objects. The programmer using a vector object would not need to know how its data was laid out in memory, or how its length was obtained.

Object-based programming is a looser interpretation of object-oriented programming[29] and may include some procedural elements for high performance or convenience. An object-based implementation of the vector class described above might access the vector length by dereferencing the vector class structure directly, without calling a class method.

Reverse communication is another technique which incorporates elements of object-based design. It is used to code several PDE solver libraries in such a way that the solver does not need to know how the data objects that it works on are formed, or the details of a preconditioner [14]. This generalization allows the solver to be used in a wider range of applications.

Object-based and object-oriented design methodologies have been used extensively over the past decade in fluid dynamics on parallel platforms. Such codes have been written in a variety of languages, including C[24], FORTRAN 90[19], and even Fortran 77[15] as well as C++[3, 13, 10]. Studies show that the object-based approach can separate the numerical scheme from parallelization procedures, which helps to increase portability and readability. This can be done while still attaining good performance [9, 20, 22]. Object-based code in fluid dynamics applications can improve reliability [24], encapsulate optimizations [8] and facilitate load balancing in adaptive grid computations[21] and mesoscale modeling[4].

[7]Report of the National Center for Atmospheric Research Code Assessment Panel, July 27-29, 1999.

Object-based software has also been used successfully in other computational areas pertinent to earth science models, such as sparse matrix computations [1, 17].

2.2 Frameworks

Like modules, frameworks help to organize complex codes, though at a larger scale. They provide a way of sharing not just a single module, but a collection of integrated modules. A technical definition of a framework is that it "provides a family of interrelated classes that can be reused, mainly through composition, for the rapid construction of various programs in a particular application domain."[6]

A simple way to structure a framework is as layers of modules. Basic modules, which for climate models might include a set of communication routines, math utilities, and I/O utilities are built as generally as possible without compromising performance (i.e., select critical sections may be written procedurally). A collection of these modules may be modeled as a bottom layer which can be used to build another layer of more complicated components. Multiple layers may be supported.

Frameworks have been classified into "called" and "calling" categories[30]. "Calling" frameworks dictate the flow of control in an application, controlling and invoking a sequence of methods. "Called" frameworks provide interrelated methods or libraries of methods that can be invoked by an application. One can think of such a framework as having multiple interrelated libraries embedded in it. In our pilot project, we propose to investigate a "called" framework which would allow researchers freedom in structuring their codes and which would also allow legacy codes to use modules within it.

Some popular examples of layered frameworks are POOMA for general scientific computation[27], and Overture for solving PDEs[2], both of which are designed for parallel platforms. Some domain-specific frameworks in the sciences and engineering are for simulation of aerospace propulsion systems[25], ecosystem simulations[5], and real-time radar signal processing[7]. Locally, layered architectures can be found in the Parallel Ocean Program (POP) model from Los Alamos and the Weather Research and Forecast (WRF) model at NCAR[18].

Investigation of layered frameworks is essential since layering can be employed to simplify code development and to help insulate the scientific efforts from the volatility in the HPC market. One can put vendor or architecture specific code in a lower layer of a framework, thus isolating these details from the majority of the model developers and users.

3 Proposed Research and Development Activities

In the sections that follow we discuss the four areas of research and development proposed here.

3.1 Assess Internal Software Needs

Our first line of inquiry centers around identifying code that could be shared and strategies for sharing it. Code reuse is critical to conserving resources, and improves the quality of code through collaborative development and testing. Shared code should be as general and modular as possible; generalization increases the likelihood that a particular piece of code will be useful in multiple contexts, and modularization reduces dependencies between the module and other parts of the code, making it easier to swap. Identifying shareable general purpose routines also allows scientists to offload some of their code development. Mathematicians and software engineers can build and optimize standard methods, so that scientists can concentrate on advancing models scientifically.

There are a wide variety of modeling efforts at NCAR, ranging from the newly constructed modular, layered WRF model[18] to more traditionally architected codes such as the Thermosphere-Ionoshere-Mesosphere-Electrodynamics GCM (TIME-GCM)[28]. Up to now software engineers in different modeling groups at NCAR and its collaborating institutions have developed code in relative isolation from each other. There are some grass roots efforts where this is changing but for the most part it appears that developers must seek still seek code in diverse locations, forge their own contacts with other groups, or create new utility, mathematical and physics routines.

We propose to assess and prioritize common software infrastructure needs of the NCAR community. This assessment will be conducted in collaboration with with the model development and algorithm research communities to ensure relevance. The assessment will identify candidate shared software components and the specific type of functionality required. It will use demand for functionality and estimates of implementation costs to help prioritize the components. The assessment will result in a prioritized list of shared components, a set of draft API's and an estimate of the staff resources required to create them. In addition, we will create and maintain a web forum for discussing common code needs and coding practices to promote institutional sharing of ideas.

3.1.1 Sample Modules and Module Libraries

Over the past two months we have met with several of the model development groups at NCAR to discuss common software needs. In these preliminary meetings we have found that shareable software falls mainly into two categories. The first is a collection of fairly straightforward utilities and the second is set of complex mathematical routines. The utilities are prime candidates for "productizing" because they are modular, commonly used, and easy to incorporate. In addition, they account for a good deal of the code in current models and they are relatively easy to write or already written. Making these utilities available as shared libraries will shrink model code size as well as reduce the burden and time for model development. Complex mathematical routines, such as solvers, are motivated by a different set of factors. These are difficult to develop, especially when trying to make them both mathematically robust and scalable across a range of complex machine architectures. Because their development demands much time and advanced knowledge of mathematics, computer science and software engineering, it is essential to avoid redundant efforts.

The lists below are examples of code that could be shared. They were compiled based on input from the PCM and CSM groups.

Utilities

A communication library. Many message passing utility libraries which wrap and extend the MPI standard are in use at NCAR and elsewhere. Examples include *GCOM* in use at the United Kingdom Meteorological Office (UKMO) and Norwegian Met. Office (DNMI) which is a wrapper for MPI and Cray T3E SHMEM libraries. The *rpncomm* library developed at Canadien Recherche en prévision numérique (RPN/CMC) provides a set of common tools for building distributed-memory atmospheric models such as data transposition.

A simple math utility library. These utilities might include methods such as the calculation of orbital parameters and solar declination angle, a calendar manager for tracking model time, and machine size independent global reduction operators.

A computational utility library. Computational utilities might include code for allocating and freeing memory or timing and performance metrics.

A set of common I/O modules. Most larger models use some form of history and restart files. I/O routines parse output fields and write them to history files. Restart files preserve the current state of a model so that it can resume after an interruption. The history writing module must write to a convention in order to interact predictably with post processing graphics and diagnostics, and collectively deciding upon that convention would be part of designing a shared module. It is a particularly good time to begin work on such a module, since the netCDF library[26] for array-oriented data access on which many existing I/O modules are based is in the process of becoming both parallel and increasingly modular.[8]

Mathematical Libraries

FFTs: one-, two-, and three-dimensional FFTs. FFTs are ubiquitous, with applications including spectral dynamical algorithms and data analysis.

Differential Operators: routines for computing common differential operators such as divergence, vorticity, gradients, and the Laplacian of scalar and vector functions. Also included are utility programs for computing associated Legendre functions and Gauss points and weights. These general functions are the building blocks of the fluid equations and spectral transforms.

Linear Solvers: solves linear and nonlinear elliptic PDEs using multigrid iteration. Current applications include pressure solvers in anelastic models or Helmholtz problems arising in 3D semi-implicit time discretisations.

Interpolations: methods for transferring between one, two, three and four dimensional arrays defined on uniform and nonuniform orthogonal grids. Regridding might occur in a flux coupler linking multiple model components, or for prolongation and restriction (interpolation) within a multigrid iteration scheme.

A natural source for shared mathematical libraries is to look at those written for use in the atmospheric and related sciences by Paul Swartztrauber and John Adams of NCAR's Computational Science Section. This a focus of the pilot project described in Section 3.4.

3.2 Assess Related External Efforts

3.2.1 Survey of Frameworks and Object-Based Approaches

The second thrust of our research and development program will be an investigation of reusable software components created elsewhere for their applicability to our discipline. If there are existing approaches or software upon which we can build we will do that rather than reinventing the wheel. The approaches and software packages we plan to examine include those mentioned in Section 2 such as POOMA, PETSc, Overture, and DAGH (Distributed Adaptive Grid Hierarchy)[23].

We have already begun to examine PETSc. It is a suite of tools designed to allow users to write sophisticated parallel applications which involve the solution of partial differential equations. PETSc does not attempt to be a complete parallel mathematical software library, but rather focuses on providing some specific components. This is accomplished by providing a collection of increasingly complex functionality that spans various levels of abstraction. It include more data objects such as distributed matrices and vectors, in addition to Krylov subspace methods, preconditioners and linear and nonlinear solvers. These routines are built on low level libraries like BLAS, LAPACK and MPI. Use of the higher level objects allows the application programmer to work directly with the mathematical abstractions rather than being forced to deal with low-level communication, details. One potential use of PETSc is in the sparse matrix operations of the interpolation scheme in the CSM flux coupler.

3.2.2 Survey of Other Institutions

Finally, as part of our investigation, we will also investigate software structures, tools and practices at meteorological centers such as the ECMWF, Goddard Institute for Space Studies, Max Planck Institute, the UKMO, DNMI and RPN.

[8] Private communication, Dave Fulker of NCAR Unidata

3.3 Evaluate Collaboration and Software Development Utilities

As mentioned above, the model development efforts centered at NCAR are increasingly interdisciplinary and multi-institutional. No one organization has all the requisite resources and expertise. Given this reality, we believe that it is imperative to build an infrastructure or supporting environment to support these geographically distributed collaborative model development efforts. This infrastructure should encourage code sharing, communication, and make it as easy for people at disparate locations to collaborate as if they were just down the hall. Thus, the third avenue of research and development is an investigation into software utilities to be integrated into an environment for improved software reliability and enhanced collaboration. The elements of such an environment would ideally include distributed repositories, easy-to-use version control, issue-tracking tools, automatic configuration tools, and automated test suites. A very long-term goal might be a fully integrated environment in which all of these tools interact intelligently through web interfaces. For now, simply evaluating alternatives and adopting tools which are open source and flexible, and therefore likely to persist, is probably the best strategy.

We have identified several open source version control systems, such as tkCVS, cvsweb and Bonsai, which provide graphical user interfaces for CVS (the latter two via the web) and propose to evaluate them within the CSS. Primary goals for any new system would be the ease with which current version trees already in CVS or RCS could be imported, and retaining all capabilities, such as limiting editing capabilities to certain users, that are included in a package like CVS.

There are numerous open source software repository packages available. One which we would like to evaluate further is Repository In a Box (RIB), developed at the National High Performance Software Exchange (NHSE) Technical Team at the University of Tennessee. The advantage of RIB is that it allows the created repository to interoperate with other software repositories that use the Basic Interoperability Data Model (BIDM), the IEEE standard for interoperable software cataloguing on the Internet. This includes the extensive netLib repository. A second repository which we would like to examine is NUCM, developed at the Software Engineering Research Laboratory (SERL) at the University of Colorado[31]. NUCM is distributed and can track the dependencies of its contents, both valuable capabilities to have for complicated, collaborative software such as coupled climate models.

Software Dock from SERL[12] is a prototype of a web-based auto-distribution system which allows a user on a remote system to install software packages, obtain component updates, reconfigure software configurations or remove them in a self-consistent way. This may be helpful in the distribution of complex models.

The final element to be investigated is a web site with a web forum for discussing software standards and common software needs across NCAR.

3.4 CSS Pilot Project

Over the past two decades CSS staff has invested many tens of FTE years to develop software libraries of numerical routines particularly focused on the numerical problems arising in atmospheric and related science research. These libraries include FFTPACK, SPHEREPACK, FISHPACK, MUDPACK, and REGRIDPACK. These are all documented on the NCAR/SCD web site. This collection represents a wealth of mathematical knowledge applicable to current modeling efforts and is an invaluable software asset. They continue to be used by multiple modeling groups such as the NCAR TIME-GCM group, which uses a MUDPACK routine as its solver. Several packages, including FFTPACK and FISHPACK, have been accessed more than 100,000 times over the past decade, with most of the accesses occurring over the past few years. They are primarily written in Fortran 77 (MUDPACK is partially coded using Fortran 90), for vector supercomputers and shared-memory processors. Although several of the "PACKs" offer shared memory support, none support distributed memory. Thus there are no currently no versions optimized for message passing or deeply cached architectures. Equivalent libraries with portable performance across a wide range of platforms would be enormously valuable tools.

Therefore, we propose the fourth avenue of research and development. We propose to demonstrate, via a pilot project, that CSS can create a modular, high-performance, reliable modeling framework. This pilot project will provide a good testing ground for ideas and will demonstrate the feasibility of creating layered, object-based scientific code for the atmospheric and related sciences. We propose that the pilot project be the redesign of the FFTPACK and SPHEREPACK packages into a form which can be incorporated into a framework. This choice is motivated by several factors. First, these packages have a very large user base. Second, they provide CSS a local focus to a general problem. Third, it removes CSS and SCD exposure regarding stewardship and modernization of all existing software assets. Finally, it will take great strides toward protecting these invaluable software assets. It will give us necessary hands-on experience in this area of software design and credibility to move forward with more ambitious projects.

3.4.1 Abstract Data Types and Associated Classes

Part of our proposed pilot project will be an investigation into the use of abstract data types in common numerical routines used in climate and weather models. Within the modified versions of FFTPACK and SPHEREPACK we will compare and contrast different abstract data types for ease of use and performance. Unlike simple arrays, the data in abstract data types may not be contiguous, and is not necessarily associated with a particular size or distribution. This flexibility could lead to performance gains. On cache-based architectures like the IBM-SP Power3, data layout in memory can be critical for performance. Data arrangement is a function of algorithm, machine topology and memory hierarchy, global data decomposition, and local data arrangement, such as axis order and whether multiple data fields are interleaved. Basing code on data structures in which local and global data arrangement are readily changeable, and in which local data does not need to be contiguous, may make it easier to organize data to take advantage of a particular architecture.

Research into abstract data types in earth science models is not new. For example, the WRF model is based on an abstract data type which enables a researcher to easily change the parallel decomposition of a finite difference model[18]. Future work might include adapting WRF ideas for spectral models and incorporating the ability to specify the arrangement of data in local memory.

Global Arrays (GA)[16] are an example of an abstract data type used as part of the NWChem toolkit[11]. GAs provide a portable interface through which each process in a MIMD parallel program can asynchronously access logical blocks of a physically distributed matrix. The structures were designed to provide the performance advantage of message-passing while maintaing the ease of use inherent in shared memory programming.

Abstract data types may prompt the introduction of associated computational classes. Data might be associated with grid classes and mapping classes which would specify the relation of grids to the underlying machine. This helps to decouple the specification of how data is distributed from the data object, making it easier to remap the algorithm. This strategy is followed in[7].

Particular algorithms may suggest their own abstract type. For example, a data object associated with a model based on an adaptive grid might look quite different from one associated with a model based on spherical coordinates. A framework for the atmospheric and related sciences might include multiple abstract data types.

3.4.2 Future Work

We also would like to explore packaging promising new algorithms for general use. For example, a spherical filtering alternative to the spectral transform method was recently developed within the

CSS. It reduces the number of communication-intensive Legendre transforms at each time step, and thus could improve performance. In the area of iterative elliptic solvers, 3D semi-implicit time-stepping schemes in the WRF mesoscale model and the Spectral-Element Atmospheric Model (SEAM) will both require a simple object-oriented reverse communication interface to Krylov subspace methods such as the conjugate gradient (CG), generalised conjugate residual GCR and generalised minimal residual (GMRES) iterations. To construct both scalable and robust solvers, parallel preconditioners are mandatory (based on FFTPACK 5.0, MUDPACK or more sophisticated packages).

4 Staffing Requirements and Schedule

In recent years the role of the CSS has shifted away from library development and the associated basic research to that of a consulting group, providing expertise to modeling groups who require advanced knowledge of computer architecture, software engineering, computer science, numerical methods, and parallel computing. In order for CSS to undertake the research and development project proposed here it will require a refocusing on library development and some reorganization of responsibilities within the division. We suggest that the responsibility for user training, porting codes to the new machines, and all related acitivities be transitioned to the Technical Consulting Group.

We also believe that it makes sense to consolidate the maintenance and execution of benchmarks within CSS by forming a new full-time Software Engineer II Benchmark Librarian position. Finally, this project will require assistance from a software engineer with website design experience to assist in the construction and maintenance of a shared code repository and the web forum.

5 Conclusions and Recommendations

We have presented a plan for investigating the form and composition of a software infrastucture for the atmospheric and related computational sciences. We estimate that the scope of this investigation will require the entire CSS section's involvement for a period of about four months.

We have identified tasks to begin work on immediately in each of our lines of inquiry. We will continue to talk with modeling groups at NCAR and elsewhere to determine how they are responding to increased complexity in their simulations and computing environment, to compile our findings, and to refine our ideas based on their input.

In order to support the gradual introduction of shared code, we will immediately create a website with a code repository. The website will also contain links to discussion forums for common software needs and software development issues, and could be an initial venue for an NCAR-wide discussion on issues related to a common history buffer module. We can begin the repository with some simple math utilities.

Pending acceptance of this proposal we will provide the following deliverables in May 2000:

- a prioritized list of candidate shared software needs for modeling efforts across NCAR,
- a report evaluating related frameworks and research efforts aimed at using frameworks in their model development efforts,
- a summary of our findings and experience with software utilities for improved software reliability and collaboration including version control systems, software distribution utilities and web based code repositories, and

- a prototype redesign of the related packages SPHEREPACK and FFTPACK into a form which can be incorporated into a framework for high performance computing.

We believe that this investigation is a sensible and modest short-term step towards constructing a robust software infrastructure at NCAR. Our investigations will help to clarify what the problem areas are, identify successful software strategies already adopted by modeling groups, assess the extent to which code might be shared, and evaluate the feasibility of structuring shared code as a software framework.

6 List of Reviewers

The following is a list of people who consented to review this document.

Robert Bond, Associate Group Leader, Digital Radar Technology, MIT Lincoln Laboratory

Byron Boville, Section Head of Climate Modeling Section, Climate and Global Dynamics, NCAR

Tony Craig, Software Engineer, Climate and Global Dynamics, NCAR

Ben Foster, Software Engineer, High Altitude Observatory, NCAR

Al Kellie, Director, Scientific Computing Division, NCAR

John Michalakes, Visiting Software Engineer (Argonne National Laboratory), Mesoscale and Microscale Meteorology, NCAR

Bernie O'Lear, Associate Director, Scientific Computing Division, NCAR

Pete Peterson, Deputy Director, Scientific Computing Division, NCAR

Alexander Wolf, Director, Software Engineering Research Laboratory, University of Colorado

References

[1] Baley, S., Gropp, W., McInnes, L., and Smith, B. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[2] Bassetti, P., Brown, D., Davis, K., and Henshaw, W. Overture: an object-oriented framework for high performance scientific computing. In *Proceedings of ACM/IEEE SC98: y 10th Anniversary. High Performance Networking and Computing Conference*, page 801, Los Alamitos, CA, 1998. IEEE Comput. Soc.

[3] Buffat, M., Yudiana, I., and Leribault, C. Parallel simulation of turbulent compressibe flows with unstructured domain partitioning. perfromance on t3d and sp2 using oop. In *Proceedings of the Parallel CFD '96 Conference*, pages 76–83, Amsterdam, Netherlands, 1996. Elsevier.

[4] Buttner, L. and Gerlach, J. Load balancing and object-orientation: a case study from mesoscale flow modeling. In *Proceedings of Euromicro Workshop on Parallel and Distributed Processing*, pages 76–83, Los Alamitos, CA, 1995. IEEE Comput. Soc. Press.

[5] Campps, A.M.C. and Hill, D.R.C. An agent-based framework for visual-interactive ecosystem simulations. In *Transactions of the Society for Computer Simulation*, pages 139–152, USA, 1998. Soc. Comput. Simulations.

[6] Casais, E. An experiment in framework development. *Theory and Practice of Object Systems*, 1(4):269–280, 1995.

[7] DeLuca, C., Heisey, C.W., Bond, R., and Daly, J.M. A portable, object-based parallel library and framework for real-time radar signal processsing. In *Scientific Computing in Object-Oriented Parallel Environments. First International Conference, ISCOPE '97*, Berlin, Germany. Springer-Verlag.

[8] Eldred, M. S., Outka, D. E., Bohnhoff, W., and Witkowski, W. R. Optimization of complex mechanics simulations with object-oriented software design. *Computer Modeling in Engineering*, 1(3):323–352, 1996.

[9] Emerson, D.R., Ecer, A., Periaux, J., and Satofuka, N. An object-oriented programming paradigm for parallel computational fluid dynamics on distributed memory parallel computers. In *Proceedings of the Parallel CFD '97 Conference*, pages 561–568, Amsterdam, Netherlands., 1997. Elsevier Science.

[10] Forde, B.W.R, Foschi, R.O., and Steimer, F.S. Object-oriented finite element analysis. *Computers and Structures*, 34(3):355–374, 1990.

[11] Gracio, D. Ecce-nwchem client-server distributed computing model. *http://www.emsl.pnl.gov:2080/docs/ecce/about/slides/ecceNWChemSlides/index.htm*, 1998.

[12] Hall, R.S., Heimbigner, D., van der Hoek, A., and Wolf, A. The software dock: A distributed, agent-based software deployment system. In *Technical Report CU-CS-832-97*, Boulder, CO., 1997. Department of Computer Science, University of Colorado.

[13] Hatakeyama, M., Watanabe, M., and Suzuki, T. Object-oriented fluid flow simulation system. *Computers and Fluids*, 27(5-6):581–97, 1998.

[14] Heroux, M.A., Brebbia, C.A., Howard, D., and Peters, A. A reverse communication interface for 'matrix-free' preconditioned iterative solvers. In *Applications of Supercomputers in Engineering II*, pages 207–213, Southampton, UK., 1991. Comput. Mech. Publications.

[15] Hogstrom, K. and Thune, M. Portability and data structures in scientific computing - object-oriented design of utility routines in fortran. In *Parallel Computing: From Theory to Sound Practice. Proceedings of EWPC '92, the European Workshops on Parallel Computing*, pages 585–588, Amsterdam, Netherlands. IOS Press.

[16] J. Nieplocha and R. Harrisonand R. Littlefield. Global arrays: A portable 'shared-memory programing' model for distributed memory computers. In *Proceedings Supercomputing 94*. IEEE Computer Society Press, 1994.

[17] Knopp, J. On the benefits of abstract data types in sparse matrix computations. In *Technology of Object-Oriented Languages and Systems TOOLS 17. Proceedings of the Seventeenth International Conference*, pages 35–44, Englewood Cliffs, NJ. Prentice Hall.

[18] Michalakes, J., Dudhia, J., Gill, D., Klemp, J., and Skamarock, W. Design of a next-generation regional weather research and forecast model. In *Draft manuscript*.

[19] Norton, C.D., Decyk, V., and Slottow, J. Applying fortran 90 and object-oriented techniques to scientific applications. In *Object Oriented Technology. ECOOP '98 Workshop Reader*, pages 462–463, Berlin, Germany., 1998. Springer Verlag.

[20] Ohta, T. Design of a data class for parallel scientific computing. In *Scientific Computing in Object-Oriented Parallel Environments. First International Conference, ISCOPE '97*, pages 211–217, Berlin, Germany. Springer-Verlag.

[21] Oliker, L. and Biswas, R. Efficient load balancing and data remapping for adaptive grid calculations. In *SPAA '97. 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 33–42, New York, NY. ACM.

[22] Olsson, P., Rantakokko, J., and Thune, M. Software tools for parallel cfd on composite grids. In *Parallel Computational Fluid Dynamics. Implementations and Results Using Parallel Computers*, pages 725–731, Amsterdam, Netherlands. Elsevier.

[23] Parashar, M. and Browne, J.C. *System Engineering for High Performace Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive mesh Refinement.* to be published in Structured Adaptive Mesh Grid Methods, IMA Volumes in Mathematics and its Applications, 1999.

[24] Petridis, M., Knight, B., and Edwards, D. A design for reliable cfd software. In *Reliability and Robustness of Engineering Software II. Proceedings of the Second International Conference*, pages 3–17, Southampton, UK. Comput. Mechanics Publications.

[25] Reed, J.A and Afjeh, A.A. Software tools for parallel cfd on composite grids. In *An object-oriented framework for distributed computational simulation of aerospace propulsion systems*, pages 149–163, Berkeley, CA. USENIX Assoc.

[26] Rew, R.K. and Davis, G.P. Netcdf: An interface for scientific data access. In *Computer Graphics and Applications*, pages 76–82, Los Alamitos, CA., 1990. IEEE.

[27] Reynders, J.V.W. III and Dubois, P.F. The pooma framework. *Computers in Physics*, 12(5):453–459, 1998.

[28] Roble, R.G and Ridley, E.C. A thermosphere-ionosphere-mesosphere-electrodynamics general circulation model (time-gcm): equinox solar cycle minimum simulations (30-500 km). *Geophysical Research Letters*, 21(6):417–420, 1994.

[29] Rumbaugh, J., Blaha, M., Premerlani, W., F. Eddy, and W. Lorenson. *Object Oriented Modeling and Design*. Prentice-Hall, 1991.

[30] Sparks, S., Benner, K., and Faris, C. Managing object-oriented framework reuse. *IEEE Computer, Theme Issue on Managing Software Development*, 29(9):53–61, September 1996.

[31] van der Hoek, A., Heimbigner, D., and Wolf, A. A generic, peer-to-peer repository for distributed configuration management. In *Proceedings of the 18th International Conference on Software Engineering*. Berlin, Germany, March, 1996.

[32] Whaley, R. and Dongarra, J. Automatically tuned linear algebra software. In *Supercomputing '98 Proceedings (Electronic Publication)*. IEEE.