UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 285

Design of High Performance Computing Software for Genericity and Variability

MALIN LJUNGBERG

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2007

ISSN 1651-6214
ISBN 978-91-554-6837-8
urn:nbn:se:uu:diva-7768

Dissertation presented at Uppsala University to be publicly examined in 2446, 2, Polacksbacken, Uppsala, Friday, April 20, 2007 at 10:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract
Ljungberg, M. 2007. Design of High Performance Computing Software for Genericity and Variability. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 285. 29 pp. Uppsala.
ISBN 978-91-554-6837-8.

Computer simulations have emerged as a cost efficient complement to laboratory experiments, as computers have become increasingly powerful.

The aim of the present work is to explore the ideas of some state of the art software development practices, and ways in which these can be useful for developing high performance research codes.

The introduction of these practices, and the modular designs that they give rise to, raises issues regarding a potential conflict between runtime efficiency on one hand and develoment efficiency on the other. Flexible software modules, based on mathematical abstractions, will provide support for convenient implementation and modification of numerical operators. Questions still remain about whether such modules will provide the efficiency which is required for high performance applications.

To answer these questions, investigations were performed within two different problem domains. The first domain consisted of modular frameworks for the numerical solution of Partial Differential Equations. Such frameworks proved a suitable setting, since several of my research questions revolved around the issue of modularity.

The second problem domain was that of symmetry exploiting algorithms. These algorithms are based on group theory, and make ample use of mathematical abstractions from that field. The domain of symmetry exploiting algorithms gave us opportunities to investigate difficulties in combining modularity based on high level abstractions with low level optimizations using data layout and parallelization.

In conclusion, my investigation of software development practices for the area of high performance computing has proved very fruitful indeed. I have found that none of the concerns that were raised should lead us to refrain from the use of the practices that I have considered. On the contrary, in the two case studies presented here, these practices lead to designs that perform well in terms of usability as well as runtime efficiency.

Keywords: PDE solver, high-performance, coordinate invariant, curvilinear coordinates, symmetry exploiting, generalized Fourier transform, finite difference, expression tempates, feature modeling, variability

Malin Ljungberg, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

ISSN 1651-6214
ISBN 978-91-554-6837-8
urn:nbn:se:uu:diva-7768 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-7768)

List of Publications

This thesis is based on the following publications, which are referred to in the text by their Roman numerals.

I Malin Ljungberg and Michael Thuné: *Mixed C++/Fortran 90 implementation of parallel flow solvers*. Parallel Computational Fluid Dynamics, Trends and Applications: 233–240 (2001).

II Malin Ljungberg: *High Performance Generative Programming with a Fortran 95 Application*. Proceedings of the workshop Parallel/High-Performance Object-Oriented Scientific Computing: (2005), submitted to Scientific Programming.

III Malin Ljungberg, Kurt Otto, Michael Thuné: *Design and Usability of a PDE Solver Framework for Curvilinear Coordinates*. Advances in Engineering Software 37: 814-825 (2006).

IV Malin Ljungberg: *Composable Difference Operators for Coordinate Invariant Partial Differential Equations*. Technical Report 2007-007, Department of Information Technology, Uppsala University (2007).

V Krister Åhlander and Malin Ljungberg: *Generic Programming Aspects of Symmetry Exploiting Numerical Software*. Proceedings of ECCOMAS (2004), Also available as Technical Report 2004-020, Department of Information Technology, Uppsala University (2004).

VI André Yamba Yamba, Krister Åhlander, Malin Ljungberg: *Designing for Geometrical Symmetry Exploitation*. Scientific Programming, 14: 61-80 (2006)

Comments on my participation

In this section I list my main contributions for the manuscript included in this thesis.

I	I was the main contributor.
II	I am the sole author of this manuscript.
III	I was the main contributor. This is a condensed and slightly reworked version of my licentiate thesis.
IV	I am the sole author of this manuscript.
V	I was the main responsible for the generation and presentation of the symmetric geometries. The code development, the analysis, the conclusion as well as the presentation was done jointly.
VI	This manuscript is based on the Master thesis of André Yamba Yamba. I acted as assistant advisor, with particular responsibility for issues concerning software architecture. I also played an important role in converting the Master thesis into a publishable manuscript.

Acknowledgements

First I wish to thank my advisor Michael Thuné who has been my guiding light and inspiration during my many years as a graduate student. I also wish to thank my assisting advisors Kurt Otto and Krister Åhlander, who have very patiently spent many hours answering questions and giving me feedback. All my colleagues at the Department of Scientific Computing make it a wonderful workplace, with a spirit of warmth and friendliness that I will dearly miss. Finally I would like to thank my three children, Benjamin, Salman, and Nora, who make everything worthwhile.

Contents

1. Introduction

Computer simulations have emerged as a cost efficient complement to laboratory experiments, as computers have become increasingly powerful. This explains the growing interest in the field of Computational Science and Engineering. Developing software for simulations of physical processes poses particular challenges, because of the advanced mathematical models that are used when describing the problems to be solved. This is one of the reasons why software development practices that are commonly used within other areas are not well spread within the field of Computational Science and Engineering.

The aim of my work has been to explore the particular needs of the high performance computing community within the area of software development practices. In particular I have looked at how some such practices can be utilized in combination with advanced mathematical models for the solution of Partial Differential Equations (PDEs). Two specific areas have been explored; flexible composable operators for Finite Difference Methods (FDM) on curvilinear structured grids, and symmetry exploiting operators based on the Generalized Fourier Transform (GFT).

In order to approach the question of software development practices for high performance computing, I need to specify what I mean by "software development practices" and by "high performance computing". Both of these are actually very general areas, and I will satisfy myself with dealing with what I consider to be relevant sub-areas.

In this work, when I talk about "high performance computing", I am particularly considering a research environment. New numerical methods and algorithms are developed and tested by numerical analysts. The focus in this context is to show that an algorithm or a method, which quite often is based on a mathematical derivation, is efficient for a certain set of problems. The results are often presented in the form of conference contributions or as articles in scientific journals, before they are eventually picked up by an application area specialist and utilized to solve specific numerical problems.

The difference between the aims of the numerical analyst and those of the application area specialist leads to different needs with regards to software support. For the application area specialist it is crucial to be able to vary parameters and models pertaining to the physical scenario that is being modeled. The numerical analyst, on the other hand, needs software that supports variability with respect to the numerical methods and algorithms. In my work,

I am considering flexible software that caters to the needs of the numerical analyst. It is important to note that "flexibility" in this thesis means flexibility with regard to the kinds of modifications to numerical operators that numerical analysts experiment with.

Within the area of software development practices, I have not included the use of complete development environments, such as the Rational Unified Process, RUP [12]. Most industrial software projects use a process that is integrated with software support for that process. Because of this, their choice of software development process is to a large extent governed by the choice of development environment.

Within the academic community, we are less bound by software development environments, simply because most software projects do not have the money and time required to get and make efficient use of such an environment. There are two sides to this. On one hand, the quality control in a software project may suffer because of the lack of supporting software for processes such as version control, bug tracking and regression testing. On the other hand, without a rigid software development process you get a better opportunity to explore alternative paradigms.

The software development practices that I consider are thus such, that do not require any more software support than a word processor, an editor and a compiler. I have not fallen for the temptation to think that there is no point to have a software development process, unless you have a supporting environment. All software projects do have a development process, but in some cases it is less well described.

The aim of the present work is to explore the ideas of some state of the art software development practices, and ways in which these can be useful for developing high performance research codes.

2. Software development models in high performance computing

The number of software development models is as large as the number of software developers. I discuss models that have an impact on how the software is structured. This means that I do not consider models that focus on how the development work is performed, such as for example Agile Software Development [2].

When the number of lines of code grows, so does the need to divide the software up into manageable units. A successful modularization will allow the developer to consider each module in terms of how it can be used rather than in terms of how it is implemented. Design decisions will be contained, as discussed by Parnas [14], and concepts and operations from the problem domain are instead made visible. This will allow the developer to add new functionality to an existing code without getting entangled into the implementation details of the modules that are already present.

One key to success, when modularizing a code, is to ensure that the functionality contained in each module is easy to understand and easy to use. Finding a good conceptual basis for the modularization of the software may thus be crucial for its usability and maintainability.

Within high performance computing, the goal of development efficiency, through flexible modularity, may be in conflict with the goal of runtime efficiency. It is a challenge to increase flexibility without suffering a decrease in performance. For high performance codes, it is also crucial for the application programmer to be able to perform low level optimization through the selection of an appropriate data layout for a particular problem. Parallelization is another approach that is commonly used within high performance computing, in order to decrease execution time. The questions of flexible modularity and performance give rise to the following research questions:

- Can highly modular flexible codes be written without compromising performance and the ability to perform low level optimizations?
- How do we measure the flexibility of a set of software modules?
- How can parallelization be introduced in a code, which has been modularized based on mathematical abstractions?

The aims of tools such as the Unified Modeling Language, UML [3], and analysis processes, such as object-oriented analysis and design [13], are to ensure that the software modules used will be easy to use and understand. Of

particular importance here is that conceptually related data should be collected in the same module, in order to simplify modification of the software.

The use of modularization based on high level mathematical abstractions for the numerical solution of PDEs has been successfully explored in a number of software efforts. Sophus [9] is a particularly relevant example, since it is based on curvilinear coordinates, just like our efforts. Sophus also supports low level optimizations and parallelizations. Sophus uses Finite Element Methods (FEM), rather than FDM, which means that its problem domain differs slightly from that of the present work.

Overture [6] is a set of object-oriented tools for solving PDEs using FDM. One of the incentives for the present work was our need for a greater flexibility with regard to operator composition, as compared with that offered by Overture. In particular we wanted to be able to select the discretization of each term of an operator expression independently. We also wanted a more efficient direct evaluation of operator expressions than that offered by Overture. These differences are explained in more detail in Paper IV.

As the software industry has gotten more mature, an increasing amount of effort is spent on the update and modification of existing software, and also on the production of sets of related software, such as for example the versions of an operating system for a computer. This has lead to a developing interest in how change and variations are handled [10]. Techniques that are used include feature and variability modeling [5]. The terminology and graphical tools used for feature and variability modeling constitute a language, which enables the developers to define software modules that can be configured and reused in several different contexts. Such a collection of software modules, which can be used to construct several different complete programs, is commonly called a *framework*. Considering the design of a framework for high performance applications leads to the following question.

- How can object oriented analysis and design, together with feature and variability modeling, be used to design flexible, domain specific, high performance computing software?

It is common for a framework to contain different versions of the same module, but with slightly different functionality. For these cases, it is convenient to have configurable modules. The technique of generating software modules based on given parameters is called *generative programming* [5]. Parameterized modules can be custom designed for a particular purpose, and thus enhance the flexibility of the software.

- How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?

Generics, or *templates*, can be used as a method of parameterizing modules, in the languages that provide support for them, such as Java and C++. *Concepts*, which provide an extension of the structure and generative powers of templates, have recently been proposed as an update to the C++ standard [8].

The proposed concepts offer a conceptual enhancement of the C++ language which is a promising modeling option for mathematically based codes.

The Matrix Template Library, MTL [16] provides an example of how powerful generative techniques can be when applied in the area of high performance computing. POOMA (Parallel Object-Oriented Methods and Applications) [4], together with PETE (the Portable Expression Template Engine) [4], on which it is built, uses expression template techniques to achieve efficient evaluation of algebraic expressions on large datasets.

When developing software for a specific problem domain, such as the high performance codes that I am considering, the most important basis for modularization is the relevant domain specific knowledge. Modeling the software on the preexisting entities in the domain is a safe way of ensuring that developers will have a correct intuition for how to put the modules together into a complete code. The need for agreement between modules and domain specific entities is equally true for high performance codes. Because of the scarcity of descriptions of how such models are arrived at, I find myself asking the following question.

- How is modularization done in a highly abstract domain, such as mathematical software?

Considering the fact that a problem of some complexity may contain parts that are best modeled using quite diverse conceptual frameworks, it is not uncommon to end up with modules that are written in different languages. This leads me to wonder the following.

- Is it possible to combine specialized modules, written in different languages, without increasing execution time?

The present trend for computer languages is to offer unambiguous interfaces. This makes it a straightforward task to combine different language modules into complete programs.

The work at hand is an investigation of how these software modeling practices can be used to exploit the general aspects of high performance codes, in order to enhance their performance, in terms of runtime as well development efficiency. In particular, I investigate the seven questions that I have identified:

1. Can highly modular flexible codes be written without compromising performance and the ability to perform low level optimizations?
2. How do we measure the flexibility of a set of software modules?
3. How can parallelization be introduced in a code, which has been modularized based on mathematical abstractions?
4. How can object oriented analysis and design, together with feature and variability modeling, be used to design flexible, domain specific, high performance computing software?
5. How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?

6. How is modularization done in a highly abstract domain, such as mathematical software?
7. Is it possible to combine specialized modules, written in different languages, without increasing execution time?

As stressed in the Introduction, in these questions "flexibility" is considered in regard to the kinds of modifications to numerical operators that numerical analysts experiment with.

3. Results

I performed investigations within two different problem domains, in order to address the research questions. The first domain consisted of modular frameworks for the numerical solution of PDEs. Such frameworks proved a suitable setting, since several of my research questions revolved around the issue of modularity.

The second problem domain was that of symmetry exploiting algorithms. These algorithms are based on group theory, and make ample use of mathematical abstractions from that field. The domain of symmetry exploiting algorithms gave us opportunities to investigate difficulties in combining modularity based on high level abstractions with low level optimizations using data layout and parallelization, which are issues pertaining to research questions 1, 3 and 6.

3.1 Problem Domain I: Modular PDE solver frameworks

The area of modular PDE solver frameworks is huge, and I have limited my scope to solvers using finite difference methods, FDM. These methods require the problem geometry to be discretized, using structured grids. One of the challenges when using FDM is to find a structured grid that fits the domain. The use of curvilinear coordinates is often required, in order to produce a *body fitted grid*. Lately the effectiveness of such grids have been emphasized by for example Knoll et al. [11]. Metric coefficients need to be included when a FDM is defined on a curvilinear structured grid. Paper III and Paper IV describe the development of software modules supporting the inclusion of metric coefficients in FDM.

Two different PDE solver frameworks have been investigated. In Paper I we extend the Cogito [17] framework, and thus investigate inter language modularity. In Papers II, III, and IV, we look at extensions of the TENGO [1] framework. For these papers, the focus is on enhancing usability by introducing modules that simplify the process of discretizing general PDEs.

3.1.1 Paper I

In Paper I, we investigated the integration of Cogito [17], a Fortran 95 library supporting implementation of parallel PDE solvers, with two different time stepping algorithms, implemented in C++. The paper addresses research question 7, which is

7. Is it possible to combine specialized modules, written in different languages, without increasing execution time?

The purpose of the investigation was to see whether the Cogito legacy code could be used by the C++ time stepping module, with the same performance as an implementation using only Fortran 95 modules.

Integration between the C++ module and the relevant Cogito modules, implemented in Fortran 95, was done using wrapper classes, as described by Gray et al. [7]. Time measurements were made for two different PDE problems. The first was an advection simulation, using Leap-Frog time stepping, and the second was a compressible Navier-Stokes problem, using a five step Runge-Kutta time stepping method. Execution time, as well as speedup and sizeup for up to 12 processors were compared for the mixed language and pure Fortran 95 implementations. The differences between the two implementations were in all cases negligible.

In conclusion, Paper I demonstrated that a flexible PDE solver framework can be achieved by integrating specialized modules, which may also be implemented in different languages. This can be done without loss of performance.

3.1.2 Paper II

Paper II describes the development and evaluation of a module for the discretization of general, coordinate invariant, differential equations. This module is written in C++, using generative techniques, and integrated with the TENGO [1] framework, written in modular Fortran 95. Within this context, we address the following research questions

5. How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?
7. Is it possible to combine specialized modules, written in different languages, without increasing execution time?

The *expression template* programming technique [18] is used in the new module in order to provide efficient evaluation of general algebraic expressions. This technique is expanded to include not only algebraic operators, but also discretized differential operators. The module supports efficient direct evaluation of general expressions, as well as automatic formation of a matrix operator representation of such expressions.

We find that the new module

- can easily be integrated into the existing framework,

- increases the usability of the framework by supporting a flexible notation for the discretization of differential expressions, and
- does not draw an "abstraction penalty" in the form of increased execution time, relative to special purpose implementations.

The use of expression template techniques on Fortran 95 data structures, demonstrated in Paper II, provides a unique way of combining the efficient evaluation offered by the former with the powerful legacy implementations that exist in the latter.

3.1.3 Paper III

In Paper III, we study the design and usability of a PDE solver framework for curvilinear coordinates. In particular, we explore the use of feature and variability modeling, combined with object-oriented analysis and design in developing a metric framework. The purpose of the metric framework is to supply relevant metric coefficients, which are to be used when discretizing PDEs on curvilinear structured grids.

Feature and variability modeling are well established analysis tools within product line design [10]. The similarities between products being constructed from modules in a product line on one hand, and PDE solvers being implemented from modules in a PDE solver framework on the other, indicates that feature and variability modeling are suitable tools to use when designing such a framework.

Object-oriented analysis and design is a well established development practice. Our interest lies in how it is used in a highly abstract context, such as the implementation of FDM on curvilinear structured grids. We explore how classes and objects are identified based on abstract concepts from the mathematical domain. By ensuring that the software model is consistent with the mathematical concepts on which it is based, we get a framework that is easy to understand and use.

In order to draw any conclusions regarding the suitability of the development methods explored, it is necessary to establish a way to evaluate the performance of a modular framework, both in terms of execution time and in terms of usability. In Paper III, we compare the execution time of a special purpose implementation to that of an implementation based on the modular framework. Usability is measured by comparing the number of code changes that would be necessary to perform on each of these implementations in order to introduce certain functionality changes. The changes considered are

- switching from an orthogonal to a non-orthogonal metric,
- increasing the order of accuracy from 2 to 4,
- changing from planar to axial symmetry, and
- introducing domain decomposition parallelism.

Out of the research questions that were identified in Chapter 2, Paper III addresses the following:

4. How can object oriented analysis and design, together with feature and variability modeling, be used to design flexible, domain specific, high performance computing software?
6. How is modularization done in a highly abstract domain, such as mathematical software?
2. How do we measure the flexibility of a set of software modules?

The results presented in Paper III consist of

- a description of how feature and variability modeling can be used within the context of PDE solver frameworks,
- a design for a metric framework, to be used when discretizing PDEs using FDM on curvilinear structured grids, and
- an evaluation of the performance of the new framework, with respect to execution time as well as usability.

3.1.4 Paper IV

In Paper IV, I combine the inter language modularity of Paper I with the general difference expressions of Paper II, and the metric framework presented in Paper III. The result is a set of software representations of continuous differential operators. I call these representations the Flexible Operators (FlexOp). They are intended as a tool for the numerical analyst. The FlexOp support flexible discretization of general algebraic expressions, including representations of continuous differential operators, on curvilinear structured grids.

The major benefits of the FlexOp are

- a high level of agreement between the mathematical description of the continuous PDE and the implementation of its discretization,
- flexibility with respect to selection of discretization schemes,
- automated inclusion of metric coefficients when discretizing coordinate invariant differential operators,
- efficient direct evaluation, using expression template techniques, as well as
- support for matrix operator formation, thus enabling efficient evaluation using matrix-vector multiplication.

The research question that is treated in Paper IV is

5. How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?

The design of FlexOp demonstrates one way of answering this question. The FlexOp use parameterized modularity to provide high flexibility and usability, without loss of performance, where performance is measured relative to a single language, special purpose implementation.

3.2 Problem Domain II: Symmetry exploiting algorithms

The idea behind symmetry exploiting algorithms is to use results from the mathematical area of group theory to find more efficient ways of solving PDEs in domains that exhibit symmetries. An example of a symmetry exploiting algorithm is the Discrete Fourier Transform (DFT), which may be used to find a more efficient solution for problems that are periodic. In our work, we study more general symmetries, such as for example a triangle or a cube, and we use the Generalized Fourier Transform (GFT) to find efficient solutions to PDEs.

The mathematical machinery involved when working with GFT is extensive and highly abstract. Our aim has been to find general and flexible, yet efficient, software representations of the mathematical concepts, in order to meet the demands on performance as well as usability.

In Paper V we achieve flexibility through the use of parameterized classes, while a parallel, object-oriented code developed in C is presented in Paper VI.

3.2.1 Paper V

In Paper V, we use a boundary element method to solve an electrostatic problem on a cube. The GFT is used to develop a more efficient solution method on this symmetric domain. In the paper, we outline both the boundary element method and how the GFT is used to find a solution. The GFT based solution is compared with a direct solution of the same problem, with respect to execution time. We verify that the GFT based solution is much faster, particularly for large problems.

We also describe the software representations of the mathematical abstractions used in implementing the GFT code. We find that generic programming, through the use of parameterized classes, is suitable for expressing mathematical abstractions. The highly efficient GFT algorithm is easily implemented based on software representations that contain a high level of abstraction.

In Paper V, we address the following research questions

5. How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?
6. How is modularization done in a highly abstract domain, such as mathematical software?

We demonstrate how a design using parameterized classes, based on highly abstract mathematical concepts, enables us to implement a GFT based algorithm for the solution of PDEs on symmetrical domains. High flexibility can be achieved, without loss of performance, since parameterized classes offer compile time polymorphism.

3.2.2 Paper VI

In Paper VI, we present a software design for GFT based solution methods, similar to that of Paper V, but in this case neither inheritance, nor parameterized classes are used, and the implementation is in C. In Paper VI, we explore how the GFT algorithm can be fine tuned for different contexts, through the use of customized data layout. We also enhance the performance of the algorithm by introducing a low level parallelism.

The research questions addressed are:

1. Can highly modular flexible codes be written without compromising performance and the ability to perform low level optimizations?
3. How can parallelization be introduced in a code, which has been modularized based on mathematical abstractions?

The results are very encouraging, as we find that the high level of abstraction in the software design in no way compromises the ability to introduce low level optimizations and parallelism.

4. Conclusions

In Chapter 2, I identified a number of questions regarding software design for high performance computing software. In Chapter 3, I presented six different investigations that cover aspects of these software design questions. Here I summarize the answers that I found.

1. Can highly modular flexible codes be written without compromising performance and the ability to perform low level optimizations?
3. How can parallelization be introduced in a code, which has been modularized based on mathematical abstractions?

In Paper I, we demonstrated that a flexible PDE solver framework can be achieved by integrating specialized modules, written in different languages. We found that this inter language operatbility could be introduced without loss of performance.

In Paper VI, we described how context dependent low level optimizations, in the form of specialized data layouts, are introduced in a code, which is modularized based on high level mathematical abstractions. I find that the high level of abstraction in the software design in no way compromises the ability to introduce low level optimizations and parallelization.

2. How do we measure the flexibility of a set of software modules?

One of the results presented in Paper III is a usability evaluation of a modular software design. I find that the results of such evaluations provide important input to the software design process, as high performance codes evolve and get increasingly complex.

4. How can object oriented analysis and design, together with feature and variability modeling, be used to design flexible, domain specific, high performance computing software?

In Paper III, we also explored how feature and variability modeling can be used as an aid in the analysis and design of a software frameworks for handling metric coefficients. This framework is highly specialized and is intended to be used in the implementation of PDE solvers, using FDM on curvilinear structured grids.

I find that feature and variability modeling are powerful tools for the analysis of such frameworks, since they allow for the inclusion of variational aspects, which are not covered by an ordinary object-oriented analysis and design process.

5. How can generative techniques be used in modular high performance computing software to enhance flexibility without a loss of performance?

We explored the use of generative techniques in Papers II, IV, and V. We found that they are very suitable for use in high performance computing, because they offer a combination of high flexibility and static polymorphism. Static polymorphism opens up for an increased opportunity of optimization relative to the dynamic polymorphism of pure object-orientation. Generative techniques thus give us polymorphism without the loss of efficiency that would come with the use of virtual functions and inheritance.

6. How is modularization done in a highly abstract domain, such as mathematical software?

In Papers III and V, we describe software designs that are based on the mathematical domains of differential geometry and group theory, respectively. We find that a software model which closely mimics the mathematical framwork leads to highly usable code. Such a model enables the developer to express concepts and algorithms more or less directly in terms of the relevant mathematics. This, in turn, facilitates the development of new algorithms and methods.

7. Is it possible to combine specialized modules, written in different languages, without increasing execution time?

In Papers I and II, we explore the issue of inter language modularity. In particular, we combine specialized modules written in C++ and Fortran 95. We find that such modules can be combined without an increase in execution time, relative to single language implementations. This means that it is possible to use the powerful structure of the C++ language in combination with the many efficient implementations of numerical algorithms that exist in Fortran 95.

Paper II gives an example of this by demonstrating how expression template techniques can be used on Fortran 95 data structures, thus combining the efficient evaluation offered by the former with the powerful legacy implementations that exist in the latter.

Fortran 95 is not designed with interoperability in mind, but the Fortran 2003 standard includes a well specified inter language interface. As high performance computation software gets more involved, the trend is increased use of inter language operability. This trend is backed up by an increased level of standardization of inter language interfaces, such as for example the new Fortran standard [15].

In conclusion, my investigation of software development practices for the area of high performance computing has proved very fruitful indeed. I have found that none of the concerns that I voiced in Chapter 2 should lead us to refrain from the use of the practices that I have considered. On the contrary, in the two case studies presented here, these practices lead to designs that perform well in terms of usability as well as runtime efficiency.

Bibliography

[1] K. Åhlander and K. Otto. Software design for finite difference schemes based on index notation. *Future Generation Computer Systems*, 22:102–109, 2006.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. `http://www.agilemanifesto.org/`.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999. ISBN 0-201-57168-4.

[4] J. Crotinger, J. Cummings, S. Haney, W. Humphrey, S. Karmesin, J. Reynders, S. Smith, and T. J. Williams. Generic programming in POOMA and PETE. In *Selected Papers from the International Seminar on Generic Programming, Lecture Notes in Comp uter Science, Vol. 1766*, pages 218–231, London, UK, 2000. Springer-Verlag.

[5] K. Czarnecki and U. W. Eisenecker. *Generative Programming*. Addison-Wesley, 2000. ISBN 0-201-30977-7.

[6] William D. Henshaw David L. Brown and Daniel J. Quinlan. Overture: An object-oriented framework for solving partial differential equations on overlapping grids. In Michael E. Henderson, Christopher R. Anderson, and Stephen L. Lyons, editors, *Object Oriented Methods for Interoperable Scientific and Engineering Computing*. SIAM, 1999.

[7] M. Gray, R. Roberts, and T. Evans. Shadow-object interface between Fortran 95 and C++. *Computers in Science and Engineering*, 1:63–70, 1999.

[8] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, and A. Lumsdaine. Concepts: Linguistic support for generic programming in C++. In *Proceedings of the 2006 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '06)*. ACM Press, October 2006.

[9] M. Haveraaen, H. A. Friis, and T. A. Johansen. Formal software engineering for computational modelling. *Nordic Journal of Computing*, 6:241–270, 1999.

[10] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse, Architecture, Process and Organization for Business Success*. ACM Press, 1997. ISBN 0-201-92476-5.

[11] D. Knoll, J. Morel, L. Magonlin, and M. Shashkov. Physically motivated discretization methods; A strategy for increased predictiveness. *Los Alamos Science*, 29:188–212, 2005.

[12] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison Wesley, 3rd edition, 2004.

[13] B. Oestereich. *Developing Software with UML*. Addison-Wesley, 1999. ISBN 0-201-39826-5.

[14] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[15] J. Reid. The new features of Fortran 2003. JTC1/SC22/WG5 N1579, ISO/IEC.

[16] J. G. Siek and A. Lumsdaine. The matrix template library: Generic components for high-performance scientific computing. *Computing in Science and Engineering*, 01(6):70–78, 1999.

[17] M. Thuné, E. Mossberg, P. Olsson, J. Rantakokko, K. Åhlander, and K. Otto. Object-oriented construction of parallel PDE solvers. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 203–226. Birkhäuser, 1997.

[18] D. Vandevoorde and N. Josuttis. *C++ Templates: The Complete Guide*. Addison Wesley, 2002. ISBN 0201734842.

Sammanfattning på svenska

Allteftersom datorer har blivit kraftfullare har datorsimuleringar blivit ett kostnadseffektivt komplement till experiment. Detta förklarar det växande intresset för vetenskapliga beräkningar. Att utveckla mjukvara för simulering av fysikaliska processer medför speciella utmaningar på grund av de avancerade matematiska modeller som används vid problemformuleringen. Detta är en av anledningarna till att mjukvaruutvecklingsmodeller som används inom andra områden inte har spridits sig i samma utsräckning inom beräkningsvetenskapen.

Syftet med mitt arbete har varit att undersöka de speciella förutsättningar som råder för mjukvaruutveckling för högprestandaberäkningar. Jag har tittat speciellt på hur några mjukvaruutvecklingsmodeller kan användas tillsammans med avancerade matematiska modeller för att numeriskt lösa Partiella Differentialekvationer (PDE). Två områden utforskas; flexibla, komponerbara operatorer för finita-differensmetoder (FDM) på kurvilinjära strukturerade nät och symmetriutnyttjande operatorer baserade på den generaliserade fouriertransformen (GFT).

Mitt arbete har särskilt gällt mjukvara lämpad för utveckling av nya numeriska metoder. Användaren är alltså en numeriker i en forskningsmiljö. För denna grupp av användare fås hög användbarhet genom hög flexibilitet med avseende på skapandet av nya numeriska metoder. En numeriker behöver mjukvara som gör det lätt att experimentera med variationer av algoritmer och metoder.

Baserat på numerikerns behov formulerar jag här sju frågor om användandet av några utbredda mjukvaruutveckingsmodeller inom högprestandaberäkningar. Generellt rör sig dessa frågor omkring en eventuell konflikt mellan mjukvarans användbarhet för att skriva nya program å ena sidan, och exekveringstiden för dessa program å andra sidan. De sju frågorna är

1. Hur kan man skriva modulära flexibla koder med hög prestanda, och utrymme för lågnivåoptimeringar?
2. Hur mäter man flexibiliteten hos en uppsättning mjukvarumoduler?
3. Hur kan man parallelisera en kod som har modulariserats baserat på matematiska abstraktioner?
4. Hur kan objektorienterad analys och design användas tillsammans med variabilitetsmodellering för att skapa flexibel, domänspecifik mjukvara för högprestandaberäkningar?

5. Hur kan man använda generativa tekniker för att öka flexibiliteten hos modulär högprestandakod, utan att tappa prestanda?
6. Hur modulariserar man mjukvara inom en mycket abstrakt domän som matematisk mjukvara?
7. Är det möjligt att kombinera specialiserade moduler, skrivna i olika språk, utan ökad exekveringstid för programmet?

Sex olika artiklar används för att besvara ovanstående frågor. De fyra första av dessa artiklar handlar om flexibla, komponerbara operatorer för finita-differensmetoder på kurvilinjära strukturerade nät, medan de två sista handlar om symmetriutnytjande operatorer baserade på den generaliserade fouriertransformen.

Frågorna 1 och 3 besvaras av Papper I och IV. Papper I visar hur man kan bygga ett flexibelt ramverk för numerisk lösning av PDE, genom att kombinera specialiserade moduler, skrivna i olika programmeringsspråk. Prestandan för ett program skrivet utifrån ett sådant ramverk är densamma som för ett specialskrivet program, som använder bara ett programmeringsspråk.

I Papper IV, visar vi hur kontextberoende lågnivåoptimeringar, i form av speciella dataformat, kan introduceras i en kod som är modulariserad baserat på matematiska abstaktioner. Vi finner att denna typ av modularisering kan kombineras med de eftersträvade lågnivåoptimeringarna.

Fråga 2 besvaras i Papper III, där vi bland annat presenterar en utvärdering av användbarheten av en modulär mjukvarudesign. Denna typ av utvärdering blir en allt viktigare del av mjukvaruutvecklingsmodellen när högprestandakoder blir större och mer sammansatta.

Papper III besvarar också fråga 4, genom att presentera hur en mjukvaruutvecklingsmodell baserad på objektorienterad analys och design, tillsammans med variabilitetsmodellering, används för att utveckla ett ramverk för hantering av metriska koefficienter. Detta mycket specialiserade ramverk är ett stöd vid utvecklingen av numeriska PDE-lösare, baserade på finita-differensmetoder på kurvilinjära strukturerade nät.

Variabilitetsmodellering är ett kraftfullt verktyg vid analysen av den här typen av ramverk, eftersom den ger möjlighet att inkludera variabilitetsaspekter vid modelleringen. Dessa aspekter täcks inte av traditionell objektorienterad analys och design.

De generativa tekniker, som är ämnet i fråga 5, behandlas i Papper II, IV och V. Generativa tekniker är passande för mjukvara för högprestandaberäkningar, eftersom de erbjuder en kombination av hög flexibilitet och statisk polymorfism. Statisk polymorfism ger större möjlighet till optimeringar än den dynamiska polymorism arvsmekanismen erbjuder. Genom att använda generativa tekniker kan vi alltså få den flexibilitet som polymorfismen erbjuder utan den effektivitetsförlust som vi skulle få om vi implementerade denna polymorfism med hjälp av arv och virtuella funktioner.

Fråga 6 rör modularisering av matematisk mjukvara. Denna fråga besvaras av Papper III och V, som beskriver modularisering av programvara för differ-

entialgeometri respektive gruppteori. Vi finner att mjukvarumodeller som tätt följer det matematiska ramverk som de är baserade på ger mycket användbar kod. Sådana mjukvarumodeller ger utveckaren möjlighet att uttrycka koncept och algoritmer mer eller mindre direkt i termer av de matematiska begrepp som används vid algoritm- och metodutvecklingen.

Papper I och II undersöker möjligheten att kombinera mjukvarumoduler skrivna i olika språk. Vi kombinerar specialiserade moduler skrivna i C++ och Fortran 95. Vi finner att sådana moduler kan kombineras utan att exekveringstiden ökar, i jämförelse med en implementering skriven i bara ett språk. Detta betyder att det är möjligt att kombinera den kraftfulla strukturen i ett objektorienterat språk, som C++, med de många effektiva bibliotek som redan finns skrivna i Fortran.

Papper II ger ett specifikt exempel genom att beskriva hur uttrycksmallar (expression templates) kan användas på datastrukturer definierade i Fortran 95. På detta sätt kan man i en och samma kod både använda uttrycksmallarnas effektiva evaluering och ha tillgång till de många kraftfulla implementeringar av numeriska metoder som finns i Fortran.

Sammanfattningsvis finner jag att min utvärdering av mjukvaruutveckingsmodeller för högprestandaberäkningar har givit god avkastning. De frågeställningar som jag identifierade har alla blivit besvarade på ett tillfredsställande sätt. Jag har inte funnit någon konflikt mellan mjukvarans användbarhet för att skriva nya program å ena sidan, och exekveringstiden för dessa program å andra sidan. Tvärtom, inom de två områden som jag utforskat resulterade de utprovade mjukvaruutveckingsmodellerna i program som visade sig vara mycket användbara både vad det gäller flexibiltet och effektivitet.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 285

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-7768

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2007