DISSERTATION

Titel der Dissertation

Context Adaptivity for Selected Computational Kernels with Applications in Optoelectronics and in Phylogenetics

Verfasser

Mag. Hannes Schabauer

angestrebter akademischer Grad

Doktor der technischen Wissenschaften
(Dr. techn.)

Wien, April 2010

Studienkennzahl lt. Studienblatt:	A 786 881
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Dipl. Ing. Dr. Wilfried Gansterer, MSc

Contents

Summary

Keywords: high performance programming, parallel and distributed computing, high throughput computing, generalized complex symmetric eigenvalue problems (EVPs), phylogenetics, optoelectronics.

Introduction

Computational kernels are the crucial part of computationally intensive software, where most of the computing time is spent; hence, their design and implementation have to be accomplished carefully. Two scientific application problems from optoelectronics and from phylogenetics and corresponding computational kernels are motivating this thesis. In the first application problem, components for the computational solution of complex symmetric EVPs are discussed, arising in the simulation of waveguides in optoelectronics. LAPACK and ScaLAPACK contain highly effective reference implementations for certain numerical problems in linear algebra. With respect to EVPs, only real symmetric and complex Hermitian codes are available, therefore efficient codes for complex symmetric (non-Hermitian) EVPs are highly desirable. In the second application problem, a parallel scientific workflow for computing phylogenies is designed, implemented, and evaluated. The reconstruction of phylogenetic trees is an NP-hard problem that demands huge scale computing capabilities, and therefore a parallel approach is necessary.

One idea underlying this thesis is to investigate the interaction between the context of the kernels considered and their efficiency. The context of a computational kernel comprises model aspects (for instance, structure of input data), software aspects (for instance, computational libraries), hardware aspects (for instance, available RAM and supported precision), and certain requirements or constraints. Constraints may exist with respect to runtime, memory usage, accuracy required, etc..

Methodology

The concept of context adaptivity is demonstrated to selected computational problems in computational science. The method proposed here is a meta-algorithm that utilizes aspects of the context to result in an optimal performance concerning the applied metric. It is important to consider the context, because requirements may be traded for each other, resulting in a higher performance. For instance, in case of a low required accuracy, a faster algorithmic approach may be favored over an established but slower method. With respect to EVPs, prototypical codes that are especially targeted at complex symmetric EVPs aim at trading accuracy for speed. The innovation is evidenced by the implementation of new algorithmic approaches exploiting structure. Concerning the computation of phylogenetic trees, the mapping of a scientific workflow onto a campus grid system is demonstrated. The adaptive implementation of the workflow features concurrent instances of a computational kernel on a distributed system. Here, adaptivity refers to the ability of the workflow to vary computational load in terms of available computing resources, available time, and quality of reconstructed phylogenetic trees.

Contributions

Context adaptivity is discussed by means of computational problems from optoelectronics and from phylogenetics.

For the field of optoelectronics, a family of implemented algorithms aim at solving generalized complex symmetric EVPs. Our alternative approach exploiting structural symmetry trades runtime for accuracy, hence, it is faster but (usually) features a lower accuracy than the conventional approach. In addition to a complete sequential solver, a parallel variant is discussed and partly evaluated on a cluster utilizing up to 1024 CPU cores. Achieved runtimes evidence the superiority of our approach, however, further investigations on improving accuracy are suggested.

For the field of phylogenetics, we show that phylogenetic tree reconstruction can efficiently be parallelized on a campus grid infrastructure. The parallel scientific workflow features a moderate parallel overhead, resulting in an excellent efficiency.

Acknowledgments

At this point, I would like to thank some people for making this dissertation project possible. First of all, I want to thank my family and friends for various kinds of support. I am very grateful for the opportunity to work in the Research Lab Computational Technologies and Applications[1], having the possibility to elaborate this dissertation. I especially thank W. Gansterer, C. Pacher, H. Schmidt, and A. Sunderland for close and fruitful collaborations. "Danke" also to W. Gansterer for supervising and C. Überhuber for refereeing this dissertation. Thanks to R. Cerveny for improving my written English.

Moreover, I very much appreciate further collaborations, discussions, or just chatting about various things with my colleagues, including (but not limited to!) S. García Carbajal, N. Finger, A. Hess, R. Hochreiter, A. Janecek, S. Kilianová, M. Mücke, E. Schikuta, C. Vömel, M. Voracek, R. Weidlich, C. Wiesinger. *THX*!!

The work has mainly been performed under the CPAMMS project (grant number: FS397001) in the research focus area "Computational Science" at the University of Vienna[2]. A research visit to the UK including the utilization of the cluster computer HPCx has been supported by the HPC Europa2 project (project number: 228398) with the support of the European Commission – Capacities Area – Research Infrastructures[3].

This dissertation is dedicated to my father Dipl. Ing. Helmut Schabauer (†2001).

[1] see the webpage of the Research Lab CTA, `http://rlcta.univie.ac.at/`

[2] see the webpage of the CPAMMS project, `http://cpamms.univie.ac.at/`

[3] see the HPC Europa2 webpage, `http://www.hpc-europa.eu/`

Chapter 1

Introduction

The presented dissertation discusses context adaptivity for selected computational kernels with applications in optoelectronics and in phylogenetics on state-of-the-art hard- and software infrastructures. This chapter gives an introduction consisting of the disambiguation, motivation, problem setting, and goals.

Motto

The 9th International Conference on Computational Science (ICCS) took place in Baton Rouge (USA), in 2009. The annually changing head note for 2009 was "*Compute. Discover. Innovate*". This statement emphasizes the important role of computational science for all sciences now, thus it shall serve as motto of this work.

1.1 Disambiguation

In the following, we discuss some basic terms that are most relevant for this work.

Computational science

The general field of this work is "computational science", which we are going to sketch. Computational science is often credited as the third pillar in research, besides theory and experiment [PV05, Loa96, Lev89, Wil89]. In Figure 1.1, we observe science as a triangle, with theory, experiment, and computation at its vertices. Each vertex represents a style of research and provides a window through which we can look [Loa96, p.xviii]. For computational science, the concerted actions of scientists, engineers, mathemati-

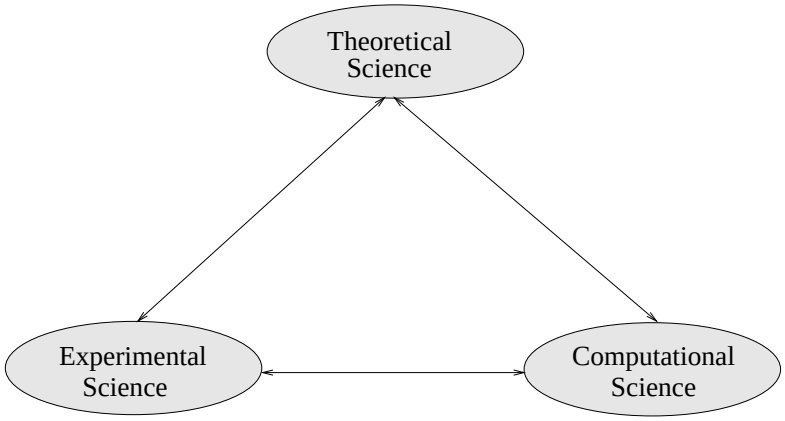

Figure 1.1: Science can be seen as a triangle, with theory, experiment, and computation at its vertices [Loa96].

cal scientists, and computer scientists are required [Ste94]. The processed problems are complex and their solutions are desirable. Scientists have to collaborate, no single discipline has the solution [Ste94].

The President's Information Technology Advisory Committee (PITAC) was an advisory committee publishing advanced technology reports in the area of high performance computing, large-scale networking, cyber security, and high assurance software and systems design[1]. The PITAC define computational science in a 2005 report:

> "*Computational science is a rapidly growing multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems.*" [BLB+05]

See Figure 1.2 for a visualization of their view on computational science, where computational science is based on (∗) computer and information science, (∗) algorithms and modeling and simulation software, and (∗) computing infrastructure. Computers are to be seen as complete scientific instruments, rather than as servants to experimental science [Wil88].

Context and adaptivity

"Context adaptivity" is one important topic of this work, therefore we discuss basic meanings of these two terms. The Oxford encyclopedic English dictionary [HA91] gives general meanings of "context" and "adaptivity". Underlined parts are most relevant for this work.

[1] see PITAC webpage, `http://www.nitrd.gov/Pitac/index.html`

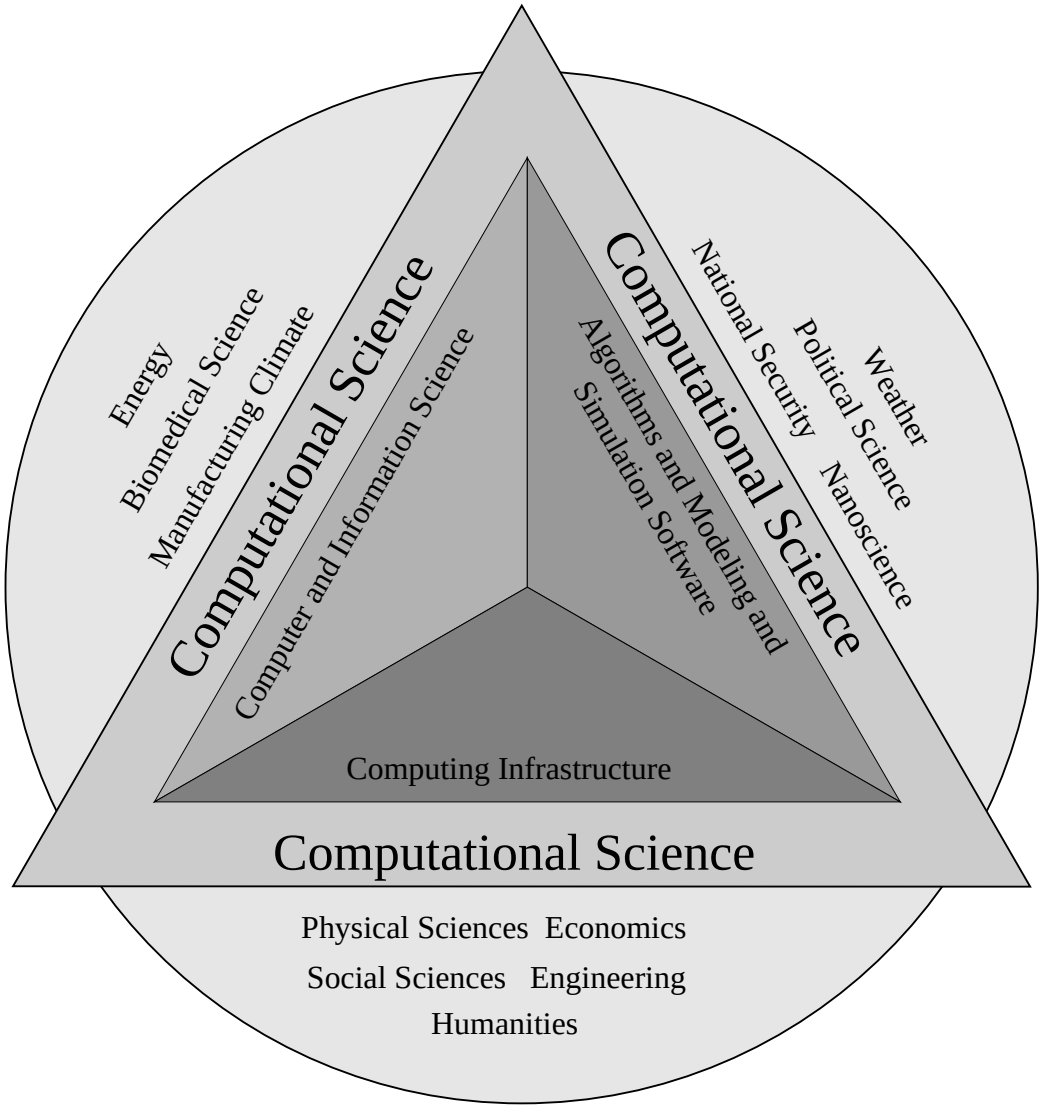

Figure 1.2: PITAC's view of computational science [BLB+05] fusing three distinct elements: (∗) algorithms and modeling and simulation software developed to solve science, engineering, and humanities problems, (∗) computer and information science that develops and optimizes the advanced system hardware, software, networking, and data management components needed to solve computationally demanding problems, (∗) and the computing infrastructure that supports both the science and engineering problem solving and the developmental computer and information science.

context **1** the parts of something written or spoken that immediately precede and follow a word or passage and clarify its meaning. **2** the circumstances relevant to something under consideration.

to adapt **1** **a** fit, adjust (one thing to another) **b** make suitable for a purpose. **c** alter or modify (esp. a text). **d** arrange for broadcasting etc. **2** *intr. & refl.* become adjusted to new conditions. **adaptive** *adjective.*

A more detailed discussion of context adaptivity for numerical software can be found in Chapter 3.

Computational kernel

"Computational kernel", "numerical kernel", or similar technical terms can be found in papers dealing with computational science, although a general definition of it is hard to find.

In [VD00], R. Vuduc and J. Demmel define a numerical kernel as "performance-critical library subroutines". The authors enumerate some examples of kernels and their applications, including sparse matrix-vector multiply in the solution of linear systems, Fourier transforms in signal processing, discrete cosine transforms in JPEG image compression, and sorting in database applications. Furthermore, they denominate the BLAS as a "practical example of a widely-used kernel standard".

R. Whaley argues in his dissertation [Wha04, p.2] that "it is important that each generation of increasingly powerful computers have well optimized computational kernels, which in turn allow for efficient execution of the higher-level applications that use them.". In a later section, he classifies the BLAS routine `_gemm` as a "level 3 BLAS computational kernel". In a bioinformatics paper about classification problems in genome and proteome analysis, V. Atalay and R. Cetin-Atalay design a computational kernel for classification problems that require specific motif extraction and search from sequences.

On the basis of these publications, we define a computational kernel for numerical software as follows.

Definition 1.1.1 (Computational kernel)**.** Computational kernels in numerical software are performance-critical parts, where most of the runtime, memory consumption, or further performance critical computer resources are consumed.

In comparison with above citations, we observe computational kernels in a broader sense; for the discussed below generalized complex symmetric

eigensolver, we denominate each essential part as a computational kernel, including (∗) factorization, (∗) transformation from generalized to standard EVP, (∗) tridiagonalization, etc.; for the phylogenetics application, we denominate dominating tree reconstruction methods inside the tree phase as separate instances of a single computational kernel.

1.2 Motivation

We see the motivation for this dissertation threefold. (*a*) Computer simulations of computationally expensive processes, here we discuss computational kernels arising in two applications from optoelectronics and phylogenetics, should always be faster and faster. However, it should be emphasized that the highest possible accuracy is not always required. In some cases, for example, where accuracy requirements are well known, it may be possible to trade reduced accuracy requirements for a higher speed. This work contributes to this central idea. (*b*) Distributed computing on state-of-the-art hardware offers a huge peak performance, but the sustained performance is often low. The inherent complexity of cluster computers and grid systems hampers the development of efficient algorithms and their implementations. Resulting from the complexity in hardware, the design of numerical algorithms and their efficient implementation have to be done carefully. (*c*) Reusability and portability are important aspects of modern software. In many cases, it may be possible to build new algorithms on top of existing library routines – for the domain of numerical linear algebra, in particular BLAS, LAPACK, and ScaLAPACK. The linear algebra codes developed for this thesis are done on top of these building blocks in order to inherit most of their excellent properties.

Grand challenges

When discussing computational kernels, we should not only keep in mind *what* can be computed, but also *why* we compute something; that is, applications are motivating the development of better computational kernels. The underlying classes of scientific applications are commonly referred to as grand challenges. Pursuant to Hoare and Milner, grand challenges have to fulfill the following criteria [HM05]: a grand challenge can be a major scientific and engineering project that is undertaken by large international efforts spread over 10 or 15 years; a grand challenge project requires the support of the general scientific community as well as dedicated commitment from those who engage in it; a grand challenge project requires understanding from

the national funding agencies whose participation enhances the international reputation of those who engage in it.

For example, in CFD, the need for the computational power of an Exaflop/s has already been reported in 1989 [Lev89]. In this article, the grand challenges include (∗) quantum chemistry, statistical mechanics, and relativistic physics, (∗) cosmology and astrophysics, (∗) CFD, (∗) materials design and superconductivity, (∗) biology, pharmacology, genome sequencing, genetic engineering, protein folding, enzyme activity, and cell modeling, (∗) medicine and modeling of human organs and bones, and (∗) global weather and environmental modeling; see also [Ste94] for a complementary list. See [Gus97, Ric95] for concrete examples of grand challenges, for instance, protein folding and climate modeling. As a lot of computing resources is needed, precise goals and a way of measuring progress toward those goals is essential [Gus97].

Grand challenges, as well as other scientific software, demand efficient implementations of core algorithms. In [DS00], J. Dongarra and F. Sullivan list a top 10 of algorithms "with the greatest influence on the development and practice of science and engineering in the 20^{th} century". Included algorithms (chronological order of development) are (∗) Metropolis algorithms for Monte Carlo, (∗) simplex method for linear programming, (∗) Krylov subspace iteration methods, (∗) the decompositional approach to matrix computations, (∗) the Fortran optimizing compiler, (∗) QR algorithm for computing eigenvalues, (∗) quicksort algorithm for sorting, (∗) fast Fourier transform (FFT), (∗) integer relation detection, and (∗) the fast multipole method. The numerical solution of EVPs frequently uses the QR algorithm and sometimes the Krylov subspace methods. In a referring article in SIAM News, B. Cipra states that "*Eigenvalues are arguably the most important numbers associated with matrices – and they can be the trickiest to compute.*" [Cip00]. Therefore, the importance of EVPs for computational science can be indicated.

Applications in computational science

The primary driver for scientific computing are the applications that demand huge computational power. Simulations of natural phenomena can never have enough computational power, sometimes they cannot be observed, due to very long (or short) durations or phenomena that lie in the past. By way of example, the past natural phenomenon of an asteroid impact was simulated using the SAGE code [GWMG04], see Figure 1.3 for a visualization of this impact. Simulation refers to the application of computational models to the study and prediction of physical events or the behavior of engineered systems [OBF+06]. As an example demanding a computationally very ex-

pensive computer simulation, a statement of the aircraft manufacturer Airbus is quoted. In the 2007 February/March edition of the Airbus letter one can read as follows[2].

> "*For example, by 2012, elements of the commercial aircraft aerodynamic design process, such as defining data for use in the loads analysis process, that currently takes up to 350 days could be reduced to just 36 days by replacing existing physical wind tunnel testing-based methods with high fidelity, computer-based simulation. In cases such as this, the time consuming process of model manufacture and testing will be replaced by enhanced, sophisticated numerical simulation methods that are able to generate and manipulate data significantly faster than any of the computational fluid dynamics (CFD) methods available today.*" [Air07]

Applications in computational science are typically very resource-demanding and diverse. Domains of computational science range from scientific investigations of the biochemical processes of the human brain and the fundamental forces of physics shaping the universe, to analysis of the spread of infectious disease of airborne toxic agents in a terrorist attack, to supporting advanced industrial methods with significant economic benefits, such as rapidly designing more efficient airplane wings computationally rather than through expensive and time-consuming wind tunnel experiments [BLB+05]. This 2005 PITAC report about computational science recommends to (∗) create a new generation of well-engineered, scalable, easy-to-use software suitable for computational science that can reduce the complexity and time to solution for today's challenging scientific applications and can create accurate simulations that answer new questions; (∗) design, prototype, and evaluate new hardware architectures that can deliver larger fractions of peak hardware performance on scientific applications; and (∗) focus on sensor- and data-intensive computational science applications in light of the explosive growth of data [BLB+05].

More power

The theoretical peak performance of computer systems is often very hard to reach, in other words, the sustained performance is usually much smaller than the theoretical peak performance [CFP08]. Moreover, it is very hard to write software which efficiently utilizes the latest generation of computer systems.

[2]The Airbus letter is not a scientific journal, but nevertheless the article evidences the huge demand for computational power from a concrete big company.

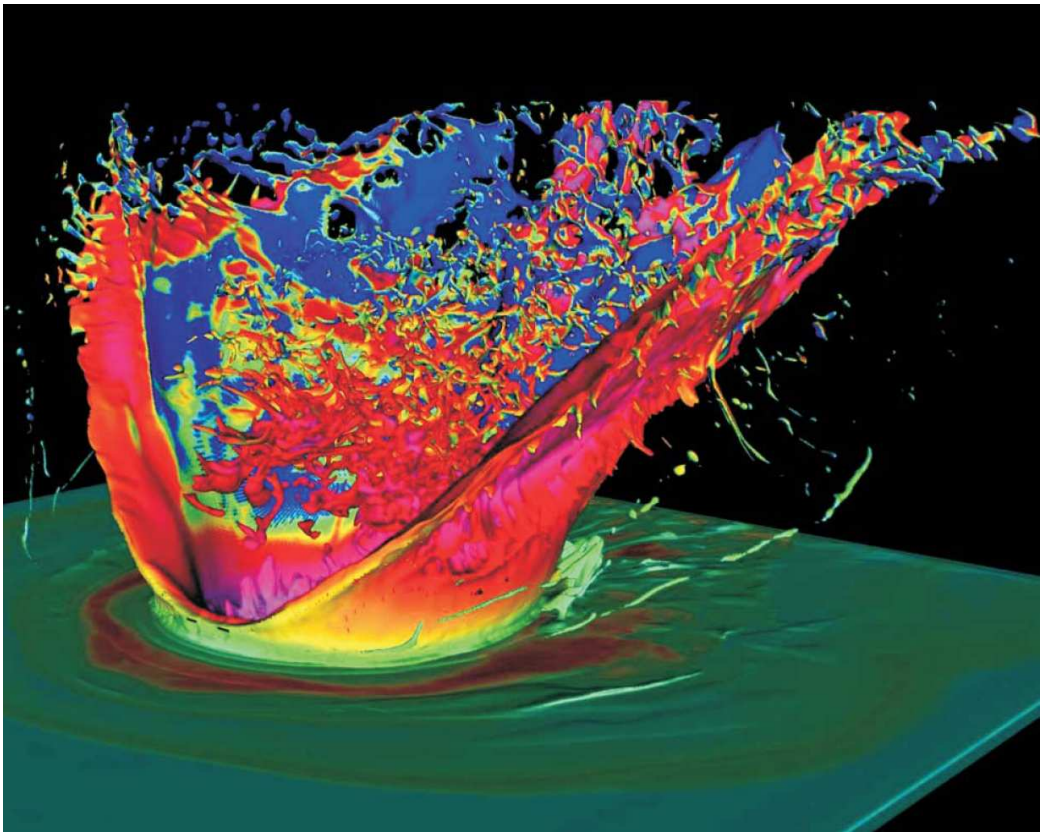

Figure 1.3: Simulated impact of the 10-km-diameter asteroid that struck the Yucatan peninsula 65 million years ago and presumably triggered the worldwide extinction of the dinosaurs and many other taxa. Shown here 42 seconds after impact, the expanding column of debris from the asteroid and crater is about 100 km high. Colors indicate temperature: the hottest material (*red*) is at about 6000 K, and the coolest (*blue*) has returned to ambient temperature; source: [PV05], see also [GWMG04] for details about this simulation. The simulation was performed using the SAGE code, implementing continuous adaptive mesh refinement.

> "*It is not uncommon for applications that involve a large amount of communication or a large number of irregular memory accesses to run at 10% of peak performance or less. Were this gap to remain fixed, we could simply wait for Moore's law to solve our problems; however, it is growing.*" [DBC+06]

The authors of the above quote argue that *"it is getting more difficult to achieve the necessary high performance from platforms because of the specialized knowledge in numerical analysis, mathematical software, compilers, and computer architecture that is required, and because rapid innovation in hardware and systems software quickly makes performance-tuning efforts obsolete"*. See Section 2.3.4 for performance definitions covering numerical aspects (accuracy) and temporal aspects (runtimes).

The typical configuration of computers for scientific computing has undergone repeated changes in recent decades. Parallel and distributed environments have become very important, constituting the most important systems of today. As computers are changing, algorithms and software need to be capable of optimally adapting to these heterogeneous, potentially distributed computer systems [Las06]. Computational science can never have enough computing power. Today's fastest computers are already capable of achieving Petaflop/s, Exaflop/s systems are already being planned [Thi08].

The multilayer process beginning with a research topic in an application discipline, leading to a solution involving large scale computational efforts, can be observed in terms of a model outlined by W. Gansterer [Gan06]. It consists of the following parts: (a) Underlying real world phenomenon, (b) analytical models, (c) computational models and algorithms, (d) software and middleware, and (e) hardware. The focus of this work is on the efficient implementation of core parts of numerical software, that is, on computational kernels.

1.3 Problem Setting

In this thesis, context adaptivity is discussed for two application problems from optoelectronics and from phylogenetics. The first application problem requires the solution of generalized complex symmetric EVPs. The most widely used method to solve such EVPs is to treat them as general complex EVPs, as done in the routine `zggev` (LAPACK) that abstains from utilizing its special algebraic properties. The new methods investigated in this thesis aim at utilizing aspects of the context and requirements of the application problem in order to maximize performance. Both serial and parallel

approaches are discussed. Parallel codes for the first application problem feature relatively high levels of communication and can therefore be regarded as fine-grained parallelization.

The second application problem is the reconstruction of phylogenetic trees. The corresponding computational kernel is constituted by the maximum likelihood method to compute phylogenetic trees. In this case, the challenge arises in the dynamic mapping of the workflow featuring multiple instances of the same computational kernel to available computers. We identify an embarrassingly parallel computational problem in the core part computing phylogenies, hence the second application problem features a coarse-grained parallel approach.

Concerning generalized complex symmetric EVPs, new computational methods are investigated. Serial and parallel solver components based on BLAS, LAPACK, and ScaLAPACK (MPI-based) codes are discussed. The principal research question is here: how and to what extent can generalized complex symmetric EVPs be solved more efficiently than with existing approaches? Moreover, how can accuracy be traded for computational efficiency? Applied methods exploit structure and include complex symmetric indefinite factorization for the reduction of generalized to standard EVPs, tridiagonalization for the reduction to tridiagonal form, and the solution of the resulting tridiagonal EVP (including backtransformation of eigenvectors). Concerning phylogenetics, we discuss a parallel approach implemented on a campus grid infrastructure. The research question is here: how can phylogenetic tree reconstruction efficiently be parallelized utilizing a coarse grained approach? All individual methods are discussed separately in detail.

1.4 Goals

The primary goal of computational kernels (which are solving computational problems) is to achieve optimal performance in terms of certain performance metrics. Performance can be measured, for example, in one of the following metrics: achieved accuracy, runtime, memory consumption, or electrical power consumption. One goal is a discussion of context adaptivity in general and its application to optoelectronics and phylogenetics. The main goal for the developed computational kernels of the generalized complex symmetric eigensolver is a faster runtime (serial and parallel) than existing approaches for general EVPs, while still satisfying given accuracy requirements. The goal of the parallel scientific workflow for phylogenetic tree reconstruction is a proof-of-concept implementation and evaluation on a campus grid infrastructure.

Synopsis. The remainder of this dissertation is organized as follows. Chapter 2 comprises fundamentals which are relevant for the following chapters. In Chapter 3, the concept of context adaptivity is introduced. Chapter 4 describes two applications which comprise computational kernels discussed later. Chapters 5 and 6 discuss implemented computational kernels for solving generalized complex symmetric EVPs and for computing phylogenetics. We conclude with results and perspectives in Chapter 7.

Chapter 2

Background

This chapter covers background information being relevant for Chapters 5 and 6. It presents the knowledge base for understanding these chapters. See [SIU08] for a reference to the international system of units (SI).

2.1 Notation

Matrices, vectors, and scalars

Capital letters A, B, etc. denominate matrices. I_n is a unit matrix (also called identity matrix) of order n, Λ is a diagonal matrix having all its eigenvalues along the diagonal. $\mathbb{R}$ denotes the set of real numbers, and $\mathbb{C}$ denotes the set of complex numbers. Minuscules x, y, etc. denote column vectors (for instance, eigenvectors), except k and l which are used as indices or loop variables; x_k is the k^{th} entry of x. i denotes the imaginary unit throughout this work, λ is an eigenvalue. e_1, e_2, etc. are corresponding column vectors of the unit matrix I.

Further symbols

A modulo function is denoted as " mod ", ceiling and floor function are denoted as "$\lceil\ \rceil$" and "$\lfloor\ \rfloor$", respectively. The transpose of a matrix is signified by a superscript "$^\top$", the conjugate transpose by an elevated "*". The end of a proof is marked by a "$\square$", an approximate value is indicated by "$\approx$". The mapping from one matrix to another is given as "$\mapsto$".

Submatrices

Lower and upper bounds of dimensions in matrices and vectors are specified as `lo:hi`. The same syntax is used in Fortran [ABH+09, Fre07] and GNU Octave [EBH08]. For example, `x(1:n)` denotes an array consisting of elements 1 to n. The first dimension of an $n \times m$ matrix varies over the rows, the second dimension over the columns. For example, `A(2:3,4:6)` defines a submatrix of matrix A, which spans over rows 2 and 3, columns 4, 5, and 6.

Function names

Function names in programming languages are typesetted in teletyper font, for instance "`zsytr1`". LAPACK has a comprehensive naming scheme, indicating the data type, matrix type, and operation of the corresponding function [ABB+99, p.12]. For example, `zgeev` identifies the eigensolver (`ev`) for general (`ge`) matrices of data type double complex (`z`). ScaLAPACK features a similar naming scheme, where the letter `p` heads the function name [BCC+97, p.30]. For example, `pzheev` (ScaLAPACK) corresponds to the eigensolver for Hermitian standard EVPs of data type double complex. Original routines from BLAS, LAPACK, or ScaLAPACK mentioned in this thesis are, unless clear from the context, followed by "(BLAS)", "(LAPACK)", or "(ScaLAPACK)", respectively. Our new routines are named according to the LAPACK / ScaLAPACK naming scheme.

2.2 Numerical Linear Algebra

Numerical linear algebra commonly deals with its standard problems linear systems of equations, least squares problems, eigenvalue problems, and singular value problems [Dem97, p.2]. Chapters 5 and 6 comprehend numerical linear algebra codes, more precisely, eigenvalue problems (EVPs).

2.2.1 Matrix Fundamentals

Below definitions, involving some abbreviations and rephrasings, are taken from the textbooks "Linear Algebra – An Introduction" by R. Bronson and G. Costa [BC07], "Applied Numerical Linear Algebra" by J. Demmel [Dem97], "Numerical Linear Algebra" by L. Trefethen and D. Bau [TB97], "Accuracy and Stability of Numerical Algorithms" by N. Higham [Hig96], "Matrix Analysis" by R. Horn and C. Johnson [HJ85], and from "CRC Concise Encyclopedia of Mathematics" by E. Weisstein [Wei03].

Definition 2.2.1 (Symmetric matrix)**.** A matrix A is symmetric if it equals its own transpose: $A = A^\top$.

Definition 2.2.2 (Conjugate transpose matrix)**.** The conjugate transpose matrix or adjoint matrix of A, written A^*, is obtained by taking the transpose of A and conjugating each entry.

Definition 2.2.3 (Hermitian matrix)**.** A Hermitian matrix or self-adjoint matrix of A equals its conjugate transpose matrix: $A = A^*$.

Definition 2.2.4 (Tridiagonal matrix)**.** A tridiagonal matrix is a square matrix having nonzero entries only on its diagonal, subdiagonal, and superdiagonal.

Definition 2.2.5 (Hessenberg matrix)**.** A lower Hessenberg matrix is a square matrix with only zero entries above the first superdiagonal; an upper Hessenberg matrix is a square matrix with only zero entries below the first subdiagonal.

Definition 2.2.6 (Diagonal matrix)**.** A diagonal matrix is a square matrix having only zeros as non-diagonal elements.

Definition 2.2.7 (Triangular matrix)**.** A lower triangular matrix is a square matrix with all its nonzero entries on the diagonal and below; an upper triangular matrix is a square matrix with all its nonzero entries on the diagonal and above.

Definition 2.2.8 (Unit triangular matrix)**.** A unit triangular matrix is a triangular matrix with solely ones on the diagonal.

Definition 2.2.9 (Unitary matrix)**.** A unitary matrix A satisfies the condition $A^*A = AA^* = I_n$. A unitary matrix with only real entries is called an orthogonal matrix.

Definition 2.2.10 (Normal matrix)**.** Let A be a square matrix. If $A^*A = AA^*$, A is a normal matrix. All unitary and Hermitian matrices are normal.

Definition 2.2.11 (Norm)**.** A vector norm is a function $\|\cdot\| : \mathbb{C}^n \mapsto \mathbb{R}$ that assigns a real-valued length to each vector. The vector norm has to satisfy the following three conditions: $(*)$ $\|x\| \geq 0$ with equality if and only if $x = 0$, $(*)$ $\|\alpha x\| = |\alpha|\|x\|$ for all $\alpha \in \mathbb{C}$, $x \in \mathbb{C}^n$, and $(*)$ $\|x + y\| \leq \|x\| + \|y\|$ for all x,$y \in \mathbb{C}^n$ (the triangle inequality). The norm concept can be generalized to matrices; in this work we focus on two norms specified below.

Let x be a vector of size n. The 2-norm of a vector is defined as $\|x\|_2 = \sqrt{(|x_1|^2 + \cdots + |x_n|^2)} = \sqrt{(x^*x)}$. The spectral norm of a matrix A is defined as $\|A\|_2 = \max\{\sqrt{\lambda}\}$, where λ is an eigenvalue of A^*A.

Definition 2.2.12 (Positive definite matrix)**.** A matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if $x^\top Ax > 0$ for all nonzero $x \in \mathbb{R}^n$.

Definition 2.2.13 (Indefinite matrix)**.** A matrix $A \in \mathbb{R}^{n\times n}$ is indefinite if $x^\top Ax < 0$ for some $x \in \mathbb{R}^n$.

Definition 2.2.14 (Matrix factorization)**.** A factorization of the matrix A is a representation of A as a product of several "simpler" matrices, which make the problem at hand easier to solve.

Definition 2.2.15 (Cholesky factorization)**.** If $A \in \mathbb{R}^{n\times n}$ is real symmetric positive definite, then there exists a unique lower triangular $L \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that $A = LL^\top$; alternatively, there exists a unique upper triangular $U \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that $A = U^\top U$. If $A \in \mathbb{C}^{n\times n}$ is complex Hermitian positive definite, then there exists a unique lower triangular $L \in \mathbb{C}^{n\times n}$ with positive diagonal entries such that $A = LL^*$; alternatively, there exists a unique upper triangular $U \in \mathbb{C}^{n\times n}$ with positive diagonal entries such that $A = U^*U$.

2.2.2 Similarity Transformations

The following definitions are taken from [Dem97, TB97], the Householder transformation was introduced in a paper by A. Householder [Hou58].

Definition 2.2.16 (Similarity transformation)**.** If a square matrix Q is nonsingular, then the map $A \mapsto Q^{-1}AQ$ is called a similarity transformation of A. Two matrices A and B are similar if there is a similarity transformation relating one to the other, that is, if there exists a nonsingular Q such that $B = Q^{-1}AQ$. If Q is nonsingular, then A and $Q^{-1}AQ$ have the same characteristic polynomial, eigenvalues, and algebraic and geometric multiplicity. The eigenvectors are transformed by Q^{-1}.

Definition 2.2.17 (Unitary similarity transformation)**.** Unitary transformations are transformations by unitary matrices. Such transformations do not alter eigenvalues of the applied matrix.

Definition 2.2.18 (Householder transformation)**.** A Householder transformation (or reflection) is a matrix of the form $H = I - 2vv^\top$ where $\|v\|_2 = 1$. $H = H^\top$ and $HH^\top = (I - 2vv^\top)(I - 2vv^\top) = I - 4vv^\top + 4vv^\top vv^\top = I$, so H is a symmetric, orthogonal matrix.

The subsequent definition follow [GSPF08]. Basics of complex orthogonal similarity can be found, for example, in [HJ91, p.477].

Definition 2.2.19 (Complex orthogonal transformation)**.** Here, a complex orthogonal transformation (COT) is of the shape

$$G = \frac{1}{t^2 + s^2} \begin{pmatrix} t & s \\ -s & t \end{pmatrix},$$

where t, $s \in \mathbb{C}$. $G^\top G = I$, therefore $G^\top = G^{-1}$ and $GAG^\top$ is a similarity transformation of A.

2.2.3 Eigenvalue Problems (EVPs)

Section 2.2.3 is dedicated to basics of EVPs and follow [BDD+00], [TB97, Lecture 24], [Hig96].

Definition 2.2.20 (Diagonalizability)**.** A matrix $A \in \mathbb{C}^{n\times n}$ is diagonalizable (non-defective) if it can be written as $A = V\Lambda V^{-1}$.

Definition 2.2.21 (Generalized complex symmetric EVP)**.** Let A and B be square n by n matrices, A, $B \in \mathbb{C}^{n\times n}$ are symmetric (non-Hermitian), x a nonzero n by 1 vector (a column vector), and λ a scalar, such that $Ax = \lambda Bx$. Then λ is called an eigenvalue, and x is called a (right) eigenvector; (λ, x) is an eigenpair.

The definition of the standard EVP is identical to the generalized EVP where $B = I_n$.

Definition 2.2.22 (Standard complex symmetric EVP)**.** Let A be a square n by n matrix, $A \in \mathbb{C}^{n\times n}$ is symmetric (non-Hermitian), x a nonzero n by 1 vector (a column vector), and λ a scalar, such that $Ax = \lambda x$. Then λ is called an eigenvalue, and x is called a (right) eigenvector; (λ, x) is an eigenpair.

Every matrix $M \in \mathbb{C}^{n\times n}$ is similar to a complex symmetric matrix [HJ85]. In contrast to a real symmetric matrix, a complex symmetric matrix A is not necessarily diagonalizable. Nevertheless, structural symmetry is of great interest for the development of space- and time-efficient algorithms. Obviously, nearly half of the information in a complex symmetric matrix is redundant, and efficient algorithms should be able to take advantage of this fact in terms of memory requirements as well as in terms of computational effort. The utilization of this purely structural property in the absence of important mathematical properties of Hermitian matrices requires a trade-off in numerical stability. In order to perform a symmetry preserving similarity transformation, the transformation matrix $Q \in \mathbb{C}^{n\times n}$ needs to be complex orthogonal (but not unitary), that is, it has to satisfy $Q^\top Q = I_n$.

2.3 Scientific Computing

Scientific computing is part of what has become known as computational science, it is primarily concerned with the development, implementation, and use of numerical algorithms and software [Hea97, p.xv].

2.3.1 High Performance Computer Systems

An overview of current high performance computer systems can be found in [DSSS05, p.53], including (but not limited to) multicomputers or multiprocessors, SMPs, distributed shared memory (DSM), distributed memory, tightly integrated MPP, commodity clusters including Beowulf-class systems, and constellations. In the following, we define some relevant terms.

Definition 2.3.1 (Distributed system)**.** A distributed system is a collection of individual computing devices that can communicate with each other [AW04].

In terms of communication, distributed systems vary from *loosely coupled* (coarse-grained, slow communication) to *tightly coupled* (fine-grained, fast communication) systems [Kle85]. The term parallelization refers to the task of designing and implementing algorithms and a piece of software capable of running on a parallel architecture, in the existence of a serial solution.

Definition 2.3.2 (Cluster)**.** A cluster is a parallel computer system comprising an integrated collection of independent nodes, each of which is a system in its own right capable of independent operation and derived from products developed and marketed for other stand-alone purposes [DSSS05].

One commonly mentioned type of a cluster is a "Beowulf", however the frequently cited paper by T. Sterling et al. [SSB+95] does not mention the term "cluster" but calls the discussed architecture a "network of workstations (NOW)"; it contains no custom components and is a full commodity off the shelf configuration. The authors of this article implemented a "single user multiple computer" featuring 16 Intel DX4 CPUs, 256 Mbyte RAM, 8 Gbyte hard disk storage, connected by two Ethernet networks each running at 10 Mbit capacity. They conclude by stating that interprocessor communication proved to be the most interesting aspect, but the attached network was inadequate under certain loads.

Definition 2.3.3 (Supercomputer)**.** Supercomputers are technologically the most powerful computers available for scientific and engineering calculations at any given time [Wil88].

Figure 2.1: Cluster Jaguar, a supercomputer ranked as #1 in the 2009/11 TOP500 list, clocked 1.759 Petaflop/s achieved sustained performance, housed at ORNL (USA).

Computational grids

Definition 2.3.4 (Computational grid)**.** A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [FK01].

Grid computing infrastructure can be categorized ranging from local facilities (cluster grids), to campus-wide (campus grids) and global (global grid) facilities [Gen02]. Computational grids are designed to support high-end computing, see Definition 2.3.4. Due to additional complexity and costs, local and campus-wide facilities should be utilized before trying to access global grids.

Middleware constitutes a key enabling technology for distributed systems [HRGB07]. According to RFC 2768 [ASC+00], the question "What is middleware?" cannot easily be answered, since the middleware of yesterday may become the fundamental network infrastructure of tomorrow; however, it is defined as "the services needed to support a common set of applications based on a distributed network environment."[ASC+00].

Leading grid software projects include Condor[1] [TTL05, LLM88] (aiming at High Throughput Computing (HTC)), Globus Toolkit[2] [Fos06, FK97] (a software toolkit used for building grids), and gLite[3] [LFF+06, LHA+04] (provides a framework for building grid applications) [RSW+08]. In Section 6.6, we discuss the utilization of Condor to implement a parallel scientific workflow in the field of phylogenetics.

It is very important to point out that, mainly due to communication costs, not all computations can be performed efficiently on all types of grids. In the worst case, the response time may even increase with an increasing number of utilized nodes or CPU cores. Therefore, the choice of the best computational infrastructure depends on each individual problem type or algorithm and the decision has to be taken carefully. I. Foster and C. Kesselman, who participated in the development of the Globus Toolkit, distinguish in [FK01] between five major classes of grid applications, including distributed supercomputing, high throughput computing, on demand computing, data intensive computing, and collaborative computing (see Table 2.1). The main (parallel) contributions of the presented work correspond mainly to the classes distributed supercomputing and high throughput computing (see Chapter 6).

[1] see the Condor project webpage, `http://www.cs.wisc.edu/condor/`
[2] see the Globus Toolkit webpage, `http://www.globus.org/toolkit/`
[3] see the gLite webpage, `http://glite.web.cern.ch/glite/`

Class	Description
Distributed supercomputing	Very large problems needing lots of CPU, memory, etc.
High throughput	Harness many otherwise idle resources to increase aggregate throughput
On demand	Remote resources integrated with local computation, often for bounded amount of time
Data intensive	Synthesis of new information from many or large data sources
Collaborative	Support communication or collaborative work between multiple participants

Table 2.1: Five major classes of grid applications [FK01].

The TOP500 project

The TOP500 project[4] was launched in 1993 to provide a reliable basis for tracking and detecting trends in high performance computing. Twice a year, a list of the sites operating the world's 500 most powerful computer systems is compiled and released by its authors H. Meuer, E. Strohmeier, J. Dongarra, and H. Simon [Meu08]; this list is called "the TOP500 list". Along with names and sites, one can look up characteristics of present and past computers, hence these lists are an excellent source for the inspection of development of high performance computer systems. Table 2.2 lists the currently fastest supercomputers at the time of November 2009; accordingly, Jaguar[5] (see Figure 2.1 for a photo) is the fastest system, followed by Roadrunner and Kraken XT5. In this report, maximum performance is the measured performance achieved on running the Highly Parallel LINPACK (HPL) benchmark [DLP03]. HPL is a portable C code which generates, solves, checks and times the solution process of a random dense linear system of equations on distributed-memory computers. Theoretical peak performance and power consumption are additionally taken into the TOP500 list.

In addition to the individual lists of the fastest supercomputers, the TOP500 webpage allows the user to generate figures to depict the development under certain aspects, including vendors, countries, geographical regions, continents, architecture, and others. Hence, trends of these characteristics can be analyzed over time. Regarding the architecture of supercomputers, one can easily identify the tremendous development of parallel

[4]see the webpage of the TOP500 project, `http://top500.org/`

[5]see the webpage of Jaguar, `http://www.nccs.gov/jaguar/`

#	Computer	Country	Vendor	Cores	$\mathbf{R_{max}}$	$\mathbf{R_{peak}}$	Power
1	*Jaguar*	USA	Cray	224162	1759.00	2331.00	6950.60
2	*Roadrunner*	USA	IBM	122400	1042.00	1375.78	2345.50
3	*Kraken XT5*	USA	Cray	98928	831.70	1028.85	n.s.
4	*JUGENE*	Germany	IBM	294912	825.50	1002.70	2268.00
5	*Tianhe-1*	China	NUDT	71680	563.10	1206.19	n.s.
6	*Pleiades*	USA	SGI	56320	544.30	673.26	2348.00
7	*BlueGene/L*	USA	IBM	212992	478.20	596.38	2329.60
8	*BlueGene/P*	USA	IBM	163840	458.61	557.06	1260.00
9	*Ranger*	USA	Sun	62976	433.20	579.38	2000.00
10	*Red Sky*	USA	Sun	41616	423.90	487.74	n.s.
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
100	*BlueGene/P*	USA	IBM	16384	47.73	55.71	126.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
124	*Cluster Platform 3000*	Austria	HP	4144	39.26	49.73	n.s.
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
500	*BladeCenter JS21*	UK	IBM	2800	20.05	28.00	n. s.

Table 2.2: TOP500 supercomputer sites at the time of November 2009. **Cores** corresponds to the total number of CPU cores in the system; $\mathbf{R_{max}}$ (see also Definition 2.3.16) is the maximum sustained performance of a computer (measured in Gigaflop/s) achieved in the HPL benchmark; $\mathbf{R_{peak}}$ (see also Definition 2.3.15) is the theoretical peak performance, determined by counting the number of floating-point additions and multiplications (in full precision) that can be completed during a period of time; **Power** is the electrical power consumption of the computer system, measured in kilowatt.

systems. Figure 2.2 evidences the massive dominance of different kinds of parallel systems for the last decade. Furthermore, we identify cluster architectures as presently most common systems. All current supercomputers feature a parallel architecture, hence an ample number of CPU cores is available for computation. There is a strong demand for numerical software that can optimally utilize parallel architectures.

2.3.2 Scientific Applications

Computationally expensive scientific applications are the driver for scientific computing (see also Section 1.2).

> "*While hardware performance has been growing exponentially – with gate density doubling every 18 months, storage capacity every 12 months, and network capability every 9 months – it has become clear that increasingly capable hardware is not the only requirement for computation-enabled discovery. Sophisticated software, visualization tools, middleware and scientific applications created*

> *and used by interdisciplinary teams are critical to turning flops, bytes and bits into scientific breakthroughs.*" [Cou07]

Advances in the computational capabilities of high-performance architectures make it possible to address increasingly challenging computational problems. At the same time, it is becoming considerably more difficult to build software that achieves high performance on these systems [SSG+98].

2.3.3 Software for Numerical Linear Algebra

The software we need for solving computational problems in linear algebra consists of software that is ready to run (numerical software environments), and numerical software packages which can be used as building blocks of self-made software.

Linear algebra software environments

Numerical software environments are ready to run right after installation. Matlab is a popular commercial environment for numerical computations, confer a textbook written by C. Moler, one of the founders of Matlab [Mol04]. It provides an advanced high-level programming language and a rich choice of functions for algorithm development, data analysis and visualization, and numeric computation. GNU Octave is a high-level language, primarily intended for numerical computations. It provides a command line interface for solving linear and nonlinear computational problems numerically, and for performing other numerical experiments using a language being mostly compatible with Matlab [EBH08]. We have been using GNU Octave for rapid prototyping of our Fortran codes.

Linear algebra software packages

Numerical software packages constitute the toolkit for a programmer of numerical software, many numerical software environments were developed using such packages.

[DW95] gives an overview of popular scientific linear algebra software. Accordingly, a list of dense linear algebra libraries includes EISPACK [DM84], LINPACK [DS84], LAPACK [ABB+99, Dem91, ABD+90], and BLAS [BDD+02, BLA01, DCHD90, DCHH88, DCHH85, LHKK79]. Additionally, at the time this article was written, ScaLAPACK [BCC+97, BCC+96] was in the works. EISPACK was released in the 1970's and is a collection of Fortran subroutines solving EVPs. LINPACK is a collection of Fortran subroutines to analyze and solve linear equations and linear least squares problems. Both EISPACK and

LINPACK have been largely superseded by LAPACK. BLAS and LAPACK are still protagonists in dense linear algebra and are therefore discussed later in more detail.

The Netlib[6] software repository was created in 1984 to facilitate quick distribution of public domain software routines for use in scientific computation [DGG+07]. On browsing a descriptive version of the software list, we search for codes aiming at dense EVPs. We identify ARPACK [LSY98], EISPACK, and LAPACK. Taking a closer look, we observe that none of these packages include complex symmetric eigensolvers. Out of these three packages, LAPACK contains the latest codes with regular updates; furthermore, requests to the Netlib repository reveal that LAPACK is the most popular package in the list with about 70 million accesses, ScaLAPACK comes next with about 23 million accesses (February 2010).

The BLAS is a set of kernel routines for linear algebra that has been defined in the BLAS Technical Forum standard [BDD+02, BLA01]. The BLAS serves as a foundation for major linear algebra packages that are built on top of it. Due to the crucial importance of the BLAS for numerical linear algebra, versions especially tuned for certain architectures have been developed by the individual hardware manufacturers – for example the AMD Core Math Library (ACML), Intel Math Kernel Library (MK), Engineering Scientific Subroutine Library (ESSL) by IBM, the Sun Performance Library, HP Mathematical Software Library (HP MLIB), and CUDA SDK by Nvidia. Furthermore, the Goto BLAS [GG08] is very well tuned to a couple of architectures, currently including x86, x86_64, IA64, Power, SPARC, and Alpha families of CPUs[7]. The purpose of the Automatically Tuned Linear Algebra Software (ATLAS) project is to provide portably optimal linear algebra software[8] [WPD01]. It currently provides a complete BLAS and a small subset of LAPACK. Some of the mentioned packages include additional numerical routines beyond BLAS; for example, Sun Studio 12 additionally includes the functionality of LAPACK 3.1.1, Sparse BLAS, NIST Fortran Sparse BLAS 0.5, SuperLU 3.0, and ScaLAPACK 1.8; furthermore, Fast Fourier transform (FFT) routines, direct sparse solver routines, and Interval BLAS routines are included[9]. Packages built on top of BLAS include, inter alia, LAPACK, Scalable LAPACK (ScaLAPACK), and the Scalable Library for Eigenvalue Problem Computations (SLEPc) [HRTV07, HRV05].

The Linear Algebra Package (LAPACK) is a library of Fortran subroutines for solving the most commonly occurring problems in numerical linear

[6] see the Netlib webpage, `http://www.netlib.org/`

[7] see the Goto BLAS FAQ, `http://www.tacc.utexas.edu/?id=368`

[8] see the ATLAS FAQ, `http://math-atlas.sourceforge.net/faq.html`

[9] see the webpage of Sun Studio, `http://developers.sun.com/sunstudio/`

algebra. It has been designed to be efficient on a wide range of modern high-performance computers [ABB+99, p.3]. It contains routines to solve systems of linear equations, linear least square problems, eigenvalue problems, and singular value problems. LAPACK routines are written so that as much as possible of the computation is performed by calls to the BLAS [ABB+99, p.4]. The codes are organized hierarchically, where routines that solve complete problems are called "driver" routines; for example, `zgeev` (LAPACK) solves a general standard EVP. "Computational" routines perform individual computational tasks, for example, `zgehrd` (LAPACK) reduces a complex general matrix to upper Hessenberg form. "Auxiliary" routines perform either subtasks of block algorithms, some commonly required low-level computations, or a few extensions to the BLAS. Most routines are available for the data types real, double precision, complex, and double complex (sometimes called complex∗16).

2.3.4 Performance Evaluation

In general, implementations of numerical algorithms should be as fast (the less consumed time, the better) as possible[10] and as accurate as possible. The faster and the more accurate a piece of code performs, the better.

Numerical aspects

While integer numbers 1, 2, 3, etc. can always be represented exactly (applying the adequate data type), this is not the case with real numbers. Digital representations of real numbers are always reduced to a finite subset of digits (mostly in the binary system), periods are usually not used in the computer system. As a consequence of using the binary system to represent numbers, some numbers representable without a period in the decimal system, cannot be exactly stored in the computer system[11]. The following definitions covering accurateness are cited from [Hig96].

Definition 2.3.5 (Absolute error)**.** $E_{\text{abs}}(\hat{x}) = |x - \hat{x}|$, where $\hat{x}$ is an approximation to a real number x.

Definition 2.3.6 (Relative error)**.** $E_{\text{rel}}(\hat{x}) = \frac{|x-\hat{x}|}{|x|}$, where $\hat{x}$ is an approximation to a real number x.

[10] In [Dem97, p.5], speed is even attributed to algorithms.

[11] for instance 0.1 (decimal system) $\approx 0.00011001100110011\ldots$ (binary system)

Definition 2.3.7 (Componentwise relative error)**.** A relative error that puts the individual relative errors on an equal footing is the componentwise relative error

$$\max_k \frac{|x_k - \hat{x}_k|}{|x_k|} \ .$$

Definition 2.3.8 (Accuracy)**.** Accuracy refers to the absolute or relative error of an approximate quantity.

Definition 2.3.9 (Precision)**.** Precision is the accuracy with which the basic arithmetic operations $+,-,*,/$ are performed, and for floating point arithmetic is measured by the unit roundoff (or machine precision).

Two difficulties are connected with limited precision floating point numbers: digital numbers cannot be arbitrary large or small, and the gap between two numbers cannot be arbitrary small. The latter property is very important for all kind of computations, where it is desired to compute as accurately as possible.

Definition 2.3.10 (Cancellation)**.** Cancellation is what happens when two nearly equal numbers are subtracted. It is often, but not always, a bad thing. On the other hand, it is not unusual for rounding errors to cancel in stable algorithms, with the result that the final computed answer is much more accurate than the intermediate quantities.

Definition 2.3.11 (Machine precision)**.** In a computer system, machine epsilon ϵ is the distance from 1.0 to the next larger floating point number.

Definitions of the machine precision (also called machine epsilon, machine precision, and unit roundoff) that can be found in literature, differ slightly (see, for instance, "Numerical computation 1 – Methods, Software, and Analysis" by C. Überhuber [Ü97a, p.140], "Matrix Computations" by G. Golub and C. Van Loan [GL96, p.61], [Dem97, p.12], and "LAPACK Users' Guide" by E. Anderson et al. [ABB+99, p.78]).

We intentionally chose the definition of N. Higham, because it is very concise. After each computation performed by a computer, the accuracy of a result potentially decreases. Therefore, it is important to measure the performance of an algorithm in terms of the accuracy of its result. Accuracy is not limited by precision, because arithmetic of a given precision can be used to simulate arithmetic of arbitrarily high precision. In other words, it is possible to handle any desired accuracy with the limited precision of the underlying hardware and software platform.

Parameter	Single	Single Ext.	Double	Double Ext.
p	24	≥ 32	53	≥ 64
E_{max}	$+127$	$\geq +1023$	$+1023$	$\geq +16383$
E_{min}	-126	≤ -1022	-1022	≤ -16382
$bias$	$+127$	unspecified	$+1023$	unspecified
E width (bits)	8	≥ 11	11	≥ 15
Format width (bits)	32	≥ 43	64	≥ 79

Table 2.3: Parameters for the representation of numbers following the IEEE/ANSI 754-1985 standard. Single, single extended, double, and double extended are selectable formats; p is the number of significant bits, E is an exponent, a biased exponent $e = E + bias$.

The ANSI / IEEE standard 754-1985 [IEE85] defines a family of commercially feasible ways for new systems to perform binary floating-point arithmetic. The definitions incorporate floating-point formats, rounding, operations, the handling of special numbers including infinity, exceptions and traps. Already ahead of its official publication in 1985, it was a de facto standard in the field of floating-point numbers and arithmetic [Ü97a, p.133]. Fortran meets this standard to the extent a processor's arithmetic supports it [For03]. As computing is mainly done with floating-point numbers, this is the foundation for estimating the numerical accuracy of algorithms on a computer system. The following definitions are taken from [IEE85].

Definition 2.3.12 (Binary floating-point number)**.** A bit-string characterized by three components: a sign, a signed exponent, and a significand. Its numerical value, if any, is the signed product of its significand and two raised to the power of its exponent.

A number is represented as $\pm 2^E(b_0.b_1b_2 \ldots b_{p-1})$, where the exponent E is any integer ranging from E_{min} to E_{max} and a binary digit $b_k = 0$ or 1. p equals the number of significant bits (precision). A biased exponent $e = E + bias$ is used to make the exponent's range nonnegative. Furthermore, $\pm\infty$ and NaN ("Not a Number") are representable. A summary of format parameters of the ANSI / IEEE 754-1985 standard for binary floating-point arithmetic is given in Table 2.3. Please notice that at this point we mention only a small fraction of the definitions and specifications described in the standard.

LAPACK determines the machine precision at runtime, by calling the auxiliary routine `slamch` for single precision or `dlamch` for double precision. A special version for complex numbers, which might be called `clamch` or `zlamch` is unavailable, as complex numbers are commonly represented as a pair of real

Machine and arithmetic	Machine precision
Cray-1 single	4×10^{-15}
Cray-1 double	1×10^{-29}
DEC VAX G format, double	1×10^{-16}
DEC VAX D format, double	1×10^{-17}
HP 28 and 48G calculators	5×10^{-12}
IBM 3090 single	5×10^{-7}
IBM 3090 double	1×10^{-16}
IBM 3090 extended	2×10^{-33}
IEEE single	6×10^{-8}
IEEE double	1×10^{-16}
IEEE extended (typical)	5×10^{-20}

Table 2.4: Machine precisions of various computer systems, taken from [Hig96, p.41].

numbers. Table 2.4 lists exemplary assignments of machine precisions on different machines. "Low level" numerical algorithms should incorporate these system-dependent numbers, in order to maximize the accuracy of the implemented algorithm. For instance, the LAPACK routine `zlarfg` (generates a complex elementary reflector H) calls `dlamch` to optimize the computation of its returned elimination vector x.

Temporal aspects

Time measurement is an important aspect for evaluating the performance of computational software. `cpu_time` is the Fortran intrinsic function to measure elapsed time. It is defined in the Fortran 95 standard, and may therefore be unavailable in older compilers. A high-resolution solution capable of time measuring in serial and parallel systems is the MPI subroutine `MPI_Wtime`, which can be used both in new and even old Fortran 77 compilers.

In parallel systems, one has to take care what to measure. One could be interested in the fastest of all started processes, in the slowest, or in an average of all of them. On timing parallel systems, it is often desirable to synchronize the individual processes right before the time measurement starts, and before it ends. Thus, termination of the code segment on all processes can be guaranteed. In MPI, `MPI_Barrier` fulfills this task. See [LKJ03] for a timing template explaining issues about timing in detail.

Subsequent definitions for basic performance quantification are taken from [Ü97a].

Definition 2.3.13 (Response time)**.** The time referred to as response time, elapsed time, or sometimes wall-clock time, passes between the launching of the command which starts a particular computational task and its completion, the response of the computer.

Definition 2.3.14 (Execution time)**.** In order to compare different algorithms for solving a given problem on a single computer, an execution time is defined as $\mathrm{T} := \mathrm{t}_{\mathrm{end}} - \mathrm{t}_{\mathrm{start}}$.

Definition 2.3.15 (Peak performance)**.** An important hardware characteristic, the peak performance $\mathrm{P}_{\mathrm{max}}$ of a computer, specifies the maximum number of floating-point (or other) operations which can theoretically be performed per time unit (usually per second).

Definition 2.3.16 (Empirical floating-point performance)**.** The floating-point performance characterizes the workload completed over the time span T as the number of floating-point operations executed in T:
$\mathrm{P}_{\mathrm{F}} = \frac{\text{number of executed floating-point operations}}{\text{time in seconds}}$.

Definition 2.3.17 (Relative speedup)**.** Relative speedup $S_r(n)$ can be defined as $\frac{T_{p1}}{T_n}$, where T_{p1} is the execution time of a parallel code on a single processor or processor core, and T_n is the execution time of the same code on n processors (or processor cores).

Definition 2.3.18 (Absolute speedup)**.** Absolute speedup $S_a(n)$ can be defined as $\frac{T_{s1}}{T_n}$, where T_{s1} is the execution time of the fastest serial code for a fixed problem on a single processor or processor core, and T_n is the execution time of the parallel code for solving the same problem on n processors (or processor cores).

Definition 2.3.19 (Parallel efficiency)**.** The efficiency $E(n) = \frac{S(n)}{n}$, where S_n is the speedup on n processors or processor cores.

The absolute speedup is usually smaller than the relative speedup, as the parallel program may contain unnecessary overhead if executed on a single processor.

2.3.5 Parallelization Toolkit

All modern supercomputers feature some kind of parallel architecture, see Figure 2.2 for a visualization depicting current supercomputer architectures. We find that clusters are clearly dominating the share of parallel architectures in the TOP500 with a share of currently about 83% of all supercomputer architectures.

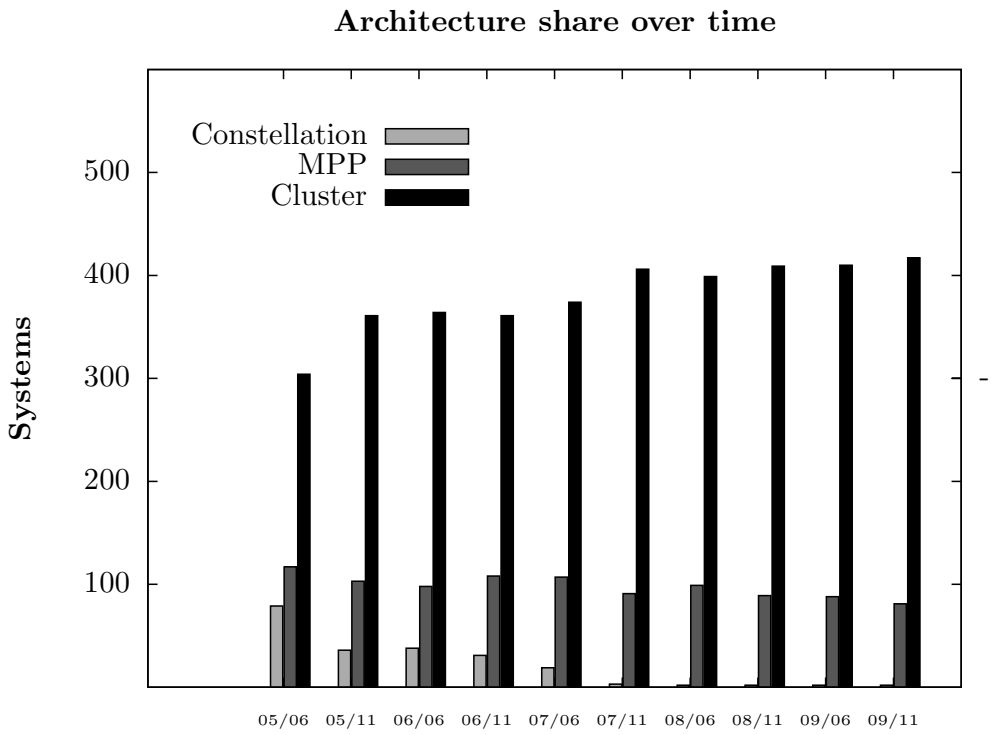

Figure 2.2: Architectures of the top 500 supercomputers from June 2005 to November 2009, including massive parallel processing (MPPs), clusters, and constellations; data taken from TOP500 webpage.

Communication libraries

Message passing is the form of communication of computing elements considered here: a combination of architecture, programming systems, and program formulation are required. Details of this paradigm can be found in [AS88]. Such low-level communication libraries enable the programming of codes for parallel architectures. MPI (Message Passing Interface) [SOHL+98] is the protagonist in message passing, and one of its most popular implementations is MPICH2[12]. BLACS (Basic Linear Algebra Communication Subprograms)[13] [Don91, DW93] needs a low-level communication library and attempts to provide ease of use and portability for distributed environments. BLACS is based on some message primitives, as available, for example, in MPI and PVM. It is mainly used as communication library for ScaLAPACK.

Task parallelism and data parallelism

Parallel applications require the exploitation of task or data parallelism. The following two definitions are taken from [CFK+94].

Definition 2.3.20 (Task parallelism)**.** In a task-parallel (also known as control or process parallel) programming paradigm the program consists of a set of (potentially dissimilar) parallel tasks that interact through explicit communication and synchronization.

Definition 2.3.21 (Data parallelism)**.** In a data-parallel programming paradigm the program consists of a series of operations that are applied identically to all or most elements of a large distributed data structure.

Both styles of parallelism may coexist. For example, in [SSHG93], a mixed compilation approach is discussed, where the compiler makes trade-offs between task and data parallelism. Most parallel-programming systems are based on either task parallelism or data parallelism, but integrating both in one model is attractive because task and data parallelism each has strengths and weaknesses [BH98].

Parallel programming languages

As Fortran was the first widely used high-level programming language on supercomputers [PZA86], early parallel languages or language extensions were also mostly based on Fortran. In [MRZ98, Lov93], the authors enumerate

[12] see the MPICH2 webpage, `http://www.mcs.anl.gov/research/projects/mpich2/`

[13] see the BLACS webpage, `http://www.netlib.org/blacs/`; source and binaries for MPI, PVM, IBM SP, and Intel (on top of NX) are available from the webpage.

IVTRAN, Kali, CM-Fortran, Vienna Fortran, and a few other languages as predecessors of the High Performance Fortran (HPF) parallel programming language [For97, KLS+94]. OpenMP is a shared memory application programming interface (API) whose features are based on prior efforts to facilitate shared memory parallel programming. OpenMP is not a programming language, it is notation that can be added to a sequential program in Fortran, C, or C++ to describe how the work is to be shared among threads that will execute on different processors or cores and to order accesses to shared data as needed [CJvdP08, p.8].

The DARPA[14] High Productivity Computing Systems (HPCS) program is focused on providing a new generation of economically viable high productivity computing systems for national security and for the industrial user community[15]. Their report [DGH+08] extensively discusses history, tools, models, and languages of HPCS. Partitioned Global Address Space (PGAS) languages provide each process direct access to a single globally addressable space. PGAS languages include the parallel programming languages Unified Parallel C (UPC) [CCB99], Co-Array Fortran [NR98], and Titanium [YSP+98]. The HPCS language project aims at inventing new languages that facilitate the creation of parallel, scalable software. These languages are Chapel [CCZ07] by Cray, X10 [CGS+05] by IBM, and Fortress [ACF+07] by Sun. Furthermore, the report tells that an HPCS system must also support C, C++, and Fortran with MPI. While latter languages, together with the PGAS languages, can be regarded as state-of-the-art approaches for parallel programming, the PGAS languages constitute trends for future languages.

2.3.6 High Performance Programming

The programming of scientific applications is a complex task, not only because of the computational problem to be solved, but also because of crude software tools and low-level programming [DSSS05]. There is no framework that can be used to write general portable-performance programs [Gat00, p.16]. The development of new codes in scientific computing should consider the following aspects. This list is just a collection of ideas compiled by ourselves and is by no means complete.

Code reuse Only well-established and stable code should be used as foundation of new codes. LAPACK and ScaLAPACK codes shall be used for linear algebra codes whenever possible, while codes from many other packages should be treated with caution.

[14] Defense Advanced Research Projects Agency (USA), see `http://www.darpa.mil/`

[15] see the HPCS webpage, `http://www.highproductivity.org/`

Source code portability If needed, carefully programmed sources can easily be ported to other programming languages. Pointers should be avoided if possible, as they usually complicate debugging and porting.

Compiler switches Two setups of compiler switches are essential. (∗) A setup for code testing is needed during the code development phase. It includes switches to include at least debugging symbols and array boundary checks. Optimization should be turned off. (∗) Another setup to achieve maximum performance is needed for the finished code. This setup embraces settings to achieve the maximum performance of the code on the target system, for example, the utilization of a 64-bit architecture and optimization switches.

Debugging A debugger like valgrind[16] [NS07, NS03] should be used to accelerate debugging. Especially the inspection of assignment of non-initialized variables is crucial.

[16] see the webpage of valgrind, `http://valgrind.org/`

Chapter 3

Context Adaptivity

As already sketched in Chapter 1, parallel architectures of various kinds have appeared, especially during the last few years[1]. These architectures offer a huge peak performance, but the achieved sustained performance frequently remains behind. Especially for numerical software that involves a lot of communication or a large number of irregular memory accesses, it is not uncommon to run at 10% efficiency or less [DBC+06]. The challenge of maximizing achieved performance is aggravated by current trends to very heterogeneous but promising platforms including multi-core CPUs [DGFK07] like the CELL processor [KBD08, BLK+07] and hardware accelerators like GPUs [Man08].

Context adaptivity aims at being a concept for addressing the issue of achieving the optimal performance of numerical software in different contexts, for instance, in terms of hardware.

3.1 Disambiguation

The linguistic meaning of the term "context adaptivity" has been outlined in Chapter 1, referring to [HA91]. Accordingly, a context is observed as "*the circumstances relevant to something under consideration*", while adaptive means "*become adjusted to new conditions*". Consequently, context adaptivity may be observed as the *ability to become adjusted to new conditions relevant to something under consideration.* G. Coulouris et al. define context for the domain of mobile and ubiquitous computing [CDK05, p.683] in the following way: "*The context of an entity (person, place or thing, whether electronic or otherwise) is an aspect of its physical circumstances of rele-*

[1]Confer the TOP500 webpage, `http://top500.org/` (statistics, charts, and development)

vance to system behaviour. That includes relatively simple values such as the location; the time; the temperature; the identity of an associated user, e.g. one operating a device, or of users nearby; the presence and state of an object such as another device, e.g. a display.". An analysis of definitions of the term context, arising in various other domains, is given in [BB05]. Similar terms context awareness [CDK05, p.683], [CDM03] and context sensitivity [McE97] are not discussed here.

For the domain of services computing, context adaptivity is defined as "*the ability of knowing and modifying the user context to ensure the best provisioning of e-service*" [MMMP04]. Obviously, it is not possible to adhere to this definition, concerning the domain of numerical software. Furthermore, context adaptivity (as a term) has been used in other domains, such as web information systems engineering [CDM03] and the video coding standard H.264 [WSBL03]. For the domain of scientific computing, W. Gansterer defines an algorithm context adaptive "*if it features some form of 'intelligence' to automatically adapt itself to performance critical properties of its context*" [Gan06]. As this definition is focussing on algorithms, a new definition covering mainly numerical software is necessary. We define context and context adaptivity of numerical software as follows.

Definition 3.1.1 (Context)**.** The context of numerical software includes the motivating application problem, the underlying mathematical models, potentially other software components (such as libraries, middleware, etc.), and the available hardware. This context may impose requirements or restrictions on the numerical software, for example, in terms of runtime, accuracy, or memory consumption.

Definition 3.1.2 (Context adaptivity of numerical software)**.** Context adaptivity of numerical software is a comprehensive concept to utilize and adapt to aspects of the context in order to optimize its performance. Here, performance may be measured in various different metrics, for example, in terms of execution time, achieved accuracy, or electrical power consumption. Context adaptivity can be achieved through meta-algorithms which automatically select optimal concrete algorithms, algorithmic variants and optimal settings for a given computational problem in a given context.

Context adaptivity is applied by describing a meta-algorithm (confer, for example, [Bä94]) for choosing appropriate concrete algorithms for a given context and desired properties. For example, in the case that a generalized complex symmetric EVP should be solved (∗) as fast as possible (∗) on a single CPU core (∗) with relatively low accuracy requirements, the meta-algorithm may choose our routine `zsygvn` (see Chapter 5) to solve this EVP.

The "intelligence" of such a meta-algorithm may be designed by declaring an optimization problem featuring an optimal performance as target function, subject to given constraints. The paper [DBC+06] refers to which algorithm or library code to apply to the problem as an "intelligent switch". The implementation of this meta-algorithm is beyond the scope of this thesis and therefore not discussed at this point.

3.2 Related Work

In the following, we distinguish between papers that discuss context adaptivity (or related topics) as an abstract concept, and exemplars for context adaptivity.

3.2.1 Abstract Concept

We mainly see two papers important for the construction of this work that we summarize as follows.

(∗) W. Gansterer discusses context adaptive algorithms in scientific computing in his habilitation thesis [Gan06] and establishes a 5-layer model for computational science. These layers are constituted by (*1*) underlying "real world" phenomenon, (*2*) analytical models, (*3*) computational models and algorithms, (*4*) software and middleware, and (*5*) hardware. Whereas the focus of the work of W. Gansterer is on algorithms, the presented work focuses on aspects of implementations of numerical algorithms. Hence, this work focuses on the layers 3, 4, and 5.

(∗) The self-adapting numerical software effort (SANS) [DBC+06] is a collaborative effort between different projects that deal with the optimization of software at different levels in relation to the execution environment[2]. In this paper, the following approaches are introduced: generic code optimization, LAPACK for clusters, SALSA, fault-tolerance linear algebra, and an optimized communication library. All of these approaches are also relevant for the considerations done in this presented work. These approaches are covered by the proclaimed SANS featuring the components application, analysis models, intelligent switch, numerical components, database, and modeler. Analogies between context adaptivity, as discussed in this thesis, and the latter paper, exist mainly as follows. The intelligent switch can be observed as a task of the meta-algorithm, and numerical components relate to computational kernels.

[2] see SANS webpage, `http://icl.cs.utk.edu/iclprojects/pages/sans.html`

Both papers emphasize the challenges of new concepts for the development of numerical application software in the light of different contexts.

3.2.2 Related Projects

The ATLAS (Automatically Tuned Linear Algebra Software) project[3] is an ongoing research effort focussing on applying empirical techniques in order to provide portable performance [WPD01]. It generates an optimized BLAS, automatically tuned to the target platform. Automatical Empirical Optimization of Software (AEOS) is a technique that was proposed by R. Clint Whaley to optimize floating point kernels [Wha04, WPD01]. An adaptive approach for computing fast Fourier transforms (FFTs) is denominated as FFTW[4]. In [FJ98], an adaptive FFT program that tunes the computation automatically for any particular hardware is proposed; see [VD00] for a paper about experiences with FFTW. SPIRAL is a code generator for linear digital signal processing (DSP) transforms[5]. For a specified transform, SPIRAL automatically generates code which is tuned to the given platform [PMJ+05]. Achieving this goal is reached by solving an optimization problem, applying different algorithmic and implementation choices.

ATLAS, Spiral, and FFTW are code generators that aim at maximizing the performance on different hardware architectures. They are mostly focused on the hardware context, but other aspects of context could also be integrated.

3.3 Goals

The goals of context adaptivity are the maximal performance of numerical software in terms of speed and accuracy, and the minimal resource allocation subject to given constraints; see Section 2.3.4 on how to measure performance, at which both numerical and temporal aspects are relevant. In most cases, the primary goal is the fastest solution of a specified job definition at an acceptable (or higher) accuracy.

Main goals

Consequently, main goals are (∗) optimal runtime performance, (∗) optimal or acceptable accuracy of results, and (∗) further performance aspects of resource allocations including RAM, network bandwidth, hard disk storage,

[3] see ATLAS webpage, `http://math-atlas.sourceforge.net/`
[4] see FFTW webpage, `http://www.fftw.org/`
[5] see SPIRAL webpage, `http://www.spiral.net/`

Main goals
Runtime performance
Accuracy
Further performance aspects (RAM, network, ...)

Table 3.1: Main goals of numerical software.

etc.. In the course of these goals, properties of the context (for instance, accuracy and speed) may sometimes be traded for each other. Main goals are to be found in Table 3.1, trading is covered in Section 3.5.

Accompanying goals

Numerical software is usually difficult to develop, due to its complex mathematical background (see for instance [RB96]). Therefore, common aspects of software engineering like portability, usability, robustness, and reusability (see for example [dAAL+04, Pre01, CNYM00]) are even more crucial for numerical software than for "ordinary" software.

Achieving stable performance on all architectures is an accompanying goal for numerical software. We denominate this goal architecture adaptivity, in accordance with [KU93] that discusses algorithms capable of adapting to different hardware platforms; see also [Gat00] for a thesis on portable high performance programming. This thesis presents a framework that aims at bridging the gap between performance and portability, by a mechanism that selects from a set of semantically equivalent variants to create a variant policy. One approach to attain serial architecture adaptivity in numerical software incorporates the utilization of the portable numerical libraries BLAS and LAPACK, respectively an implementation especially dedicated to the target platform, for example, Intel MKL, AMD ACML, IBM ESSL, ... In the parallel case, ScaLAPACK is one of the protagonists in achieving parallel architecture adaptivity. In addition to architecture adaptivity, the parallel scalability of the algorithm and its parallel implementation should be considered. Table 3.2 summarizes accompanying goals that are desirable for both serial and parallel software being favorable for numerical software. Architecture adaptivity and parallel scalability are aspects related to runtime performance.

3.4 Fabric of Context

This section discusses aspects of the context. The following items can be regarded as key aspects of the context, so they are non-tradeable constraints.

Accompanying goals
Portability
Reusability
Robustness
Usability
Architecture adaptivity
Parallel scalability

Table 3.2: Accompanying goals of numerical software.

Structure of input data

Exploiting structure of matrices for algorithms is relevant to the choice of applicable methods, see [GL96, p.16]. For example, the matrix-matrix multiplication in the BLAS features subroutines for the following types of matrices: general (`_gemm`), symmetric (`_symm`), Hermitian (`_hemm`), and triangular (`_trmm`). Each of these matrix types supports the data types single precision, double precision, complex, and double complex. Corresponding to the matrix type and precision, individual implementations are used, resulting in potentially different performance results.

One important class of matrices are sparse matrices. A matrix is sparse if many of its entries are zero. The interest in sparsity arises because its exploitation can lead to enormous computational savings and because many large matrix problems that occur in practice are sparse [DER86, p.1]; confer also [Vö03]. The Portable, Extensible Toolkit for Scientific Computation (PETSc) [BBE+08, BGMS97] is a suite of data structures and routines for the scalable (parallel) solution of scientific applications[6]. It supports about 30 different types of sparse matrices (confer online manual pages, "MatType"[7]). A further aspect for regarding the structure of input data is the handling of values that are smaller than a specified threshold. For example, the Cholesky decomposition of a sparse matrix is reported to apply a threshold in the range of $10^{-6} \dots 10^{-8}$ in order to treat smaller entries as zero and therefore decrease computational costs [SKDO07]; in another paper, a block tridiagonalization approach utilizing sparse symmetric matrices is discussed, where effectively-sparse entries are treated as zeros such that the eigenvalue error remains bounded [BGW04].

[6] see the PETSc webpage, `http://www.mcs.anl.gov/petsc/petsc-as/`

[7] see the PETSc online documentation, `http://www.mcs.anl.gov/petsc/petsc-2/documentation/index.html`

Computing platform

Numerical application software executes on platforms consisting of performance-critical hard- and software, both of them constitute a key aspect for the context of the system. Platforms may be very heterogeneous, aggravating the efficient mapping of algorithms to the computational platform [DL06, Las06]. Performance-critical hardware components include, but are not limited to, CPU, RAM, and caches. Detailed fundamentals of computer-design and performance quantification can be found in the textbook "Computer Architecture – A Quantitative Approach" by Hennessy and Patterson [HP07]. Most critical parts of the software platform comprise utilized computational libraries. For the field of numerical linear algebra, this concerns especially the implementation of the BLAS, as further libraries, for example LAPACK, heavily rely on its performance.

Temporal requirements

This constraint is the maximum time consumed by the implementation needed to complete a desired task, for example, the solution of an EVP. This time can be specified in terms of CPU time, elapsed time, or any other way of expressing a temporal restriction. For each computational problem, potentially more than one applicable algorithm exists, and for each algorithm multiple possible implementations can be realized. Each implementation may have a different performance in terms of temporal requirements.

Precision and accuracy

Provided precisions of representable numbers are defined in the IEEE standard 754-1985 [IEE85] (confer Section 2.3.4). Depending on the computational problem and its order, a different accuracy may be required. Common precision types on programming language level are single precision and double precision, arbitrary precisions are supported utilizing additional software like ARPREC[8]. ARPREC includes routines to perform arithmetics with an arbitrarily high level of precision, including many algebraic and transcendental functions [Bai05].

Required accuracy constitutes the constraint in terms of exactness of results on the application level. For instance, in [SO96, p.149], the authors argue that a reduced accuracy in the Hartree-Fock method is still adequate for most purposes. In [GSPF08], reduced accuracy in the computation of eigenpairs is discussed. Table 3.3 lists covered aspects of the context.

[8] see the High-Precision Software Directory, `http://crd.lbl.gov/~dhbailey/mpdist/`

Context
Structure of input data
Computing platform
Temporal requirements
Precision and accuracy requirements

Table 3.3: Elements comprising the context of numerical software.

3.5 Methodology

Methodology denominates all proposed techniques available to achieve context adaptivity. The subsequent methods are a framework of contingent possibilities, their realization may be very complex. In the following, these approaches are suggested, see Table 3.4 for the list of discussed methods.

Algorithmic variants

For each specific job definition, more than one algorithmic solution with distinct performance characteristics may exist. See [PTVF07, Knu98] for taxonomies and prototypical implementations of job definitions, including some algorithmic variants. Context adaptivity is achieved by selecting the algorithmic variant that fits best to the given context.

Implementation variants

One algorithm may be realized by multiple implementations featuring potentially different performance characteristics. One such variant is called blocked code. In case of reflectors for the annihilation of entries in matrices, block reflectors can be applied instead of elementary reflectors. A major advantage of block reflectors is that they can be computed using matrix multiplication for most of the work, and that they can be applied using matrix multiplication for all of the work [SP88]. It is often preferable to partition a matrix into blocks and to perform the computation by matrix-matrix operations on the blocks [DCHD90]. See [Ü97a, p.267] for a comparison of performance for different blocked matrix-matrix multiplications, evidencing the attractiveness of blocked codes. As a further example, a blocked Bunch-Kaufman (BK) factorization was proven to outperform its unblocked counterpart.

The utilization of block algorithm variants has been discussed for the case of LAPACK in [AD89]. Massively parallel environments that cannot guarantee error-free execution may demand an approach based on an implementation variant. For instance, fault tolerance in matrix multiplication has

Methodology
Algorithmic variants
Implementation variants
Parallelization of serial codes
Trading

Table 3.4: Methodology in context adaptivity for numerical software.

been covered in [GvdGKQO01, PKD97]. As different implementations of the same algorithm may perform in different ways in different contexts, the best one may be chosen.

Parallelization of serial codes

Serial codes may be parallelized: a variant executable on a parallel architecture that utilizes aspects of the parallel platform (mainly multi-core or multi-processor execution) may be identified and implemented. For some codes, efficient parallel variants are known, but some codes may not be feasibly parallelizable at all. Due to the huge importance of parallel architectures, ongoing efforts aiming at parallelization of serial codes have been undertaken; see, for example, [Bis04, DFF$^+$03, GGK03, dV94] for textbooks on parallel computing. Parallel codes are a combination of algorithmic and implementation variants.

Trading

For some implementations, multiple properties of the context may be traded for each other. Let a job definition require a relatively low accuracy; in case of the existence of multiple algorithmic variants and implementations with different performance characteristics, accuracy may be traded for speed. In other words, a faster algorithm could be less accurate, but still fulfill the required accuracy. Flexible properties of the context can be traded for each other in order to optimize the performance of the numerical software. Table 3.5 lists examples of tradeable properties.

Tradeable properties
Consumed time
Achieved accuracy
RAM occupation
Hard disk occupation
Network load
... further types of resource allocation

Table 3.5: Examples of tradeable properties in context adaptivity for numerical software.

Chapter 4

Application Problems

Application problems from science and engineering are drivers for the development of improved algorithms and numerical software. This chapter includes applications of the topics discussed in subsequent chapters. Section 4.1 describes guided-wave multisection devices that initially motivated our investigations of generalized complex symmetric EVPs, see Chapters 5 and 6; Section 4.2 describes basics of phylogenetics that motivated our investigations elaborated in Chapter 6.

4.1 Guided-Wave Multisection Devices

This section is partly based on our paper "Tridiagonalizing Complex Symmetric Matrices in Waveguide Simulations" [GSPF08].

Guided-wave multisection devices are optoelectronic devices, for example lasers, simulated by a guided-wave approach with multiple sections. For basics in waveguide analysis, see for instance, [KK01]. Classical electrodynamics can be found in [Jac99], the finite element method and related topics are described in [VKC98].

We discuss waveguides (WGs) featuring a high index of refraction (also known as refractive index). The use of such high-index contrast WGs in novel guided-wave devices for telecom- and sensing applications allows for a very versatile tailoring of the flow of light, see Figure 4.1 for a visualization. An efficient design requires the direct numerical solution of Maxwell's equations [Cro10, p.649] in inhomogeneous media. In many important cases such devices can be successfully modeled as follows: (a) in the x-direction (direction of wave propagation), the material parameters are piecewise constant, (b) the material parameters and the optical fields do not depend on the y-coordinate, and (c) in the z-direction the material parameters are allowed to

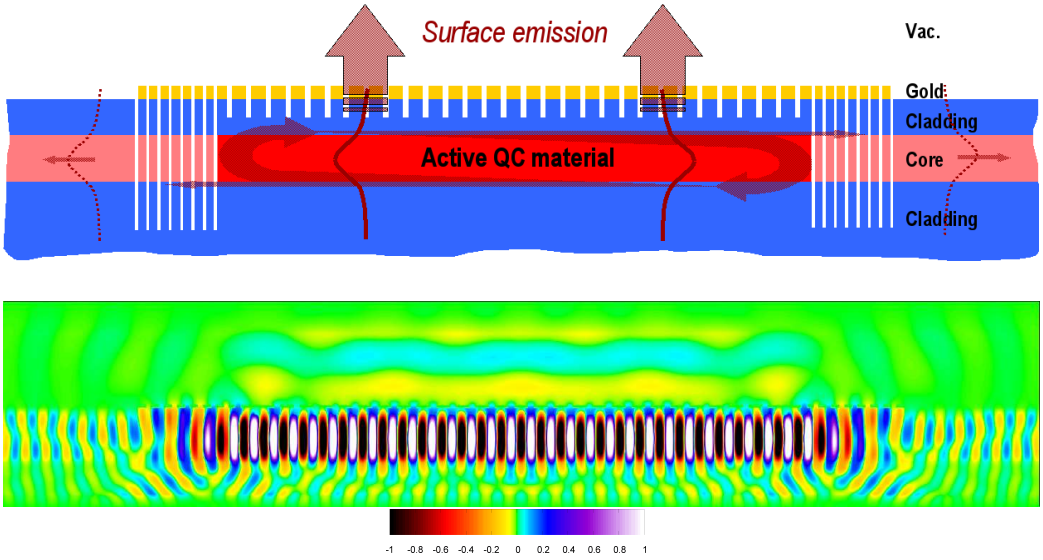

Figure 4.1: Design of a surface emitting quantum cascade laser (*top*), see also [FCS+94]; simulation of waveguides and their reflectivity (*bottom*) [FPB07].

vary arbitrarily. Usually, the z-dimension is of the order of up to several tens of wavelengths whereas the device extension into the x-direction is several hundreds of wavelengths.

A powerful numerical method for the solution of Maxwell's equations in such WG-based devices is the eigenmode expansion technique, which is often referred to as mode-matching (MM) technique [BB01, FC96, Sud93]. The electromagnetic field components in each section being homogeneous in the x-direction are represented here in terms of a set of local eigenmodes. MM requires a small computational effort compared to other numerical techniques like two-dimensional finite elements or finite difference time domain (FDTD), see for example [ZAB+99]) which can be regarded as "brute-force" methods from the viewpoint of device physics. However, MM can only be as stable and efficient as the algorithms used to determine the required set of local WG modes. Due to the open boundary conditions and materials with complex dielectric permittivities, these local eigenmodes typically have complex eigenvalues which makes their correct classification very difficult: numerical instabilities can arise from an improper truncation of the mode spectrum. In a recently developed variant of the MM technique – the variational mode-matching (VMM) [FPB07] – this stability problem is avoided by applying a Galerkin approach to the local wave equations and taking into account the whole spectrum of the discretized operators.

The VMM-approach

Within the 2D-assumption $\partial_y(\cdot) = 0$, Maxwell's equations for dielectric materials characterized by the dielectric permittivity $\varepsilon(x, z)$ take the form

$$\partial_x a \partial_x \phi + \partial_z a \partial_z \phi + k_0^2 b \phi = 0 \ , \tag{4.1}$$

where $\phi = E_y$, $a = 1$, $b = \varepsilon$ for TE- and $\phi = H_y$, $a = \frac{1}{\varepsilon}$, $b = 1$ for TM-polarization, respectively; $k_0 = \frac{2\pi}{\lambda_0}$ (vacuum wavelength λ_0).

In the z-direction, the simulation domain is $0 \leq z \leq L$. To permit an accurate description of radiation fields, an artificial absorber (that mimics an open domain) has to be installed near $z = 0$ and $z = L$. For this purpose, perfectly-matched layers (PMLs) are used by employing the complex variable stretching approach [TC98], that is, in the vicinity of the domain boundaries the coordinate z is extended into the complex plane: $z \mapsto \tilde{z} = z + \imath \int_0^z \mathrm{d}\tau\, \sigma(\tau)$, where σ is the PML parameter determining the absorption strength. At $z = 0$ and $z = L$ Dirichlet- or Neumann boundary conditions are set. However, they should not have a significant influence on the overall optical field since the physical boundary conditions must be given by the PMLs. In the x-direction, the structure is divided into n_l local WGs, which expand over $x_{l-1} \leq x \leq x_l = x_{l-1} + d_l$ with $1 \leq l \leq n_l$.

Under the necessary condition that ε does not depend on x, Equation (4.1) can be solved inside each local WG l with the separation ansatz

$$\phi^{(l)}(x, \tilde{z}) = \sum_{j=1}^{n_\varphi} \varphi_j(\tilde{z}) \sum_{\rho=1}^{n_\varphi} c_{j\rho}^{(l)} \left[\alpha_{\rho,+}^{(l)} \mathrm{e}^{\imath k_0 \nu_\rho^{(l)} (x - x_{l-1})} + \alpha_{\rho,-}^{(l)} \mathrm{e}^{-\imath k_0 \nu_\rho^{(l)} (x - x_l)} \right] \ , \tag{4.2}$$

where ρ labels the local waveguide modes. The transverse shape functions $\varphi_j(\tilde{z})$ (the same set is used for all local WGs) must satisfy the outer boundary conditions. Apart from this constraint, φ_j may be chosen rather freely allowing for adaptive refinement in z-regions where rapid variations of the field are expected. This ansatz reduces the 2D problem to a set of n_l 1D problems.

After inserting Equation (4.2) into Equation (4.1), Galerkin's method is applied to obtain a discretized version of Equation (4.1) for each local WG l. Finally, the coefficients $\alpha_{\rho,\pm}^{(l)}$ are "mode-matched" by imposing the physical boundary conditions at all the x_l-interfaces [FPB07].

The generalized complex symmetric EVP

For each local WG, the discretized version of Equation (4.1) is a generalized complex symmetric EVP of the form

$$A c_\rho = (\nu_\rho)^2 B c_\rho \ , \tag{4.3}$$

where we have suppressed the index l for simplicity. Here, the ν_ρ are the modal refractive indices and the $c_{j\rho}$ are the corresponding modal expansion coefficients occurring in Equation (4.2). A is a sum of a mass- and a stiffness-matrix, $A_{mj} = \int \mathrm{d}\tilde{z}\, \varphi_m(\tilde{z}) b(\tilde{z}) \varphi_j(\tilde{z}) - \frac{1}{k_0^2} \int \mathrm{d}\tilde{z}\, (\partial_{\tilde{z}} \varphi_m(\tilde{z})) a(\tilde{z}) (\partial_{\tilde{z}} \varphi_j(\tilde{z}))$, whereas B is a pure mass-matrix: $B_{mj} = \int \mathrm{d}\tilde{z}\, \varphi_m(\tilde{z}) a(\tilde{z}) \varphi_j(\tilde{z})$.

The generalized EVP in Equation (4.3) has the following properties: (a) A and B are complex symmetric: the complex coordinate $\tilde{z}$ originating from the PMLs (and the possibly complex material constants a and b) are responsible for the complex-valuedness; (b) B is indefinite (due to the open boundary conditions represented by the PMLs and a possibly negative material constant a); (c) the typical order of the matrices for 2D problems is $100 \ldots 1000$ (depending on the geometry and the required truncation order of the modal expansion – in 3D models the order can be much higher); (d) the full spectrum of eigenpairs is required; (e) the required accuracy is of the order 10^{-8} for the eigenpairs corresponding to the lowest order WG modes (approximately 10% of the mode spectrum); a somewhat lower accuracy (approximately 10^{-6}) is acceptable for the remainder of the spectrum; (f) depending on the WG geometry, some of the eigenvalues (especially those corresponding to the lowest order WG modes) may almost degenerate.

It is evident that an efficient eigenvalue solver which utilizes the symmetry of the EVP in Equation (4.3), as well as its special properties, is a very important building block for efficient 2D and 3D optical mode solvers.

In order to demonstrate a real testcase, a 1D waveguide problem structure was chosen, which is a $\mathrm{Si/SiO}_x$ twin waveguide operated in transversal magnetic (TM)-polarization at a wavelength $\lambda_0 = 1.55\,\mu\mathrm{m}$. The dielectric constants are $\varepsilon_{\mathrm{Si}} = 12.96$ and $\varepsilon_{\mathrm{SiO_x}} = 2.25$. The core thickness and -separation are $0.5\,\mu\mathrm{m}$ and $0.25\,\mu\mathrm{m}$, respectively. The z-extension of the model domain, terminated by electrically perfectly conducting walls, is $10\,\mu\mathrm{m}$. The PML-layer thickness is $1\,\mu\mathrm{m}$ with the PML-parameter $\sigma = 1$. As shape functions, localized linear hat functions and polynomial bubble functions with a degree up to 24 were used. The eigenpairs (λ_k, x_k) of the corresponding EVP of order $n = 1105$ lead to a weighted residual error $3.8 \cdot 10^{-14}$ which is a very satisfactory accuracy (for this test case, $\|A\|_2 = 928, \|B\|_2 = 2$), see [GSPF08] for more details, and Section 5.2 for the computational approach.

4.2 Phylogenetic Quality Assessment

This section is partly based on our paper "Phylogenetic Quality Assessment for Campus Grids" [SZvH+08].

From the time of Charles Darwin, biologists have had the dream to re-

construct the evolutionary history of organisms and express it in the form of a phylogenetic tree [NK00, p.3]. The knowledge of the evolutionary history of genes, proteins, genomes, and whole organisms, reflected by evolutionary or phylogenetic trees, has become an important factor in medical, biological, and bioinformatics research. The basic structure of a phylogenetic tree classifies organisms, with respect to their types of ribosomes, into the three domains Archaea, Bacteria, and Eucarya [CD04, p.77],[WKW90]. The following definition of the term "phylogenetic tree" was found in [MNW+04], an introduction to basic phylogenetic terms and concepts can be found in [Pag03].

Definition 4.2.1 (Phylogenetic tree)**.** "A phylogenetic tree is a leaf-labeled tree that models the evolution of a set of a taxa (species, genes, languages, placed at the leaves) from their most recent common ancestor (placed at the root)." [MNW+04]

See Figure 4.2 for an artwork taken from the Tree of Life web project site[1], where a schematic view of a phylogenetic tree is presented. Phylogenetic trees are hypotheses, not facts [WSCBF91, p.6]; hence, phylogenetic reconstruction is an essential task in current day research. Despite the speed of contemporary computers, many phylogenetic approaches are hampered by their runtime complexities and the exponential increase of possible trees. Unfortunately, more reliable statistical methods applying the maximum likelihood (ML) or Bayesian framework are also the computationally most demanding. Moreover, almost all phylogenetic methods have been shown to be NP-complete (see [GJ79] for a textbook about NP-completeness), including the Steiner tree problem [FG82], maximum parsimony [DJS86], compatibility trees [DS86], dissimilarity matrices [Day87], perfect trees [BFW92], and ancestral ML reconstruction [ABCH+04]; the computation of ML trees is even NP-hard [CT05].

After the reconstruction of a phylogenetic tree, a researcher usually has to test whether the obtained tree is well supported or whether suboptimal trees are significantly worse. Such quality assessments are usually based on simulations or resampling techniques like parametric or non-parametric bootstrapping [Efr79]. This statistical method is commonly used to place confidence intervals on subtrees of phylogenies. In the course of this method, resampling with replacement yields a series of bootstrap samples of the same size as the original data. These approaches require the evaluation of many simulated or resampled pseudo-samples to obtain support values. Unfortunately, this multiplies the computational needs of the above mentioned reconstruction methods.

[1] see the webpage of the Tree of Life web project, `http://tolweb.org/`

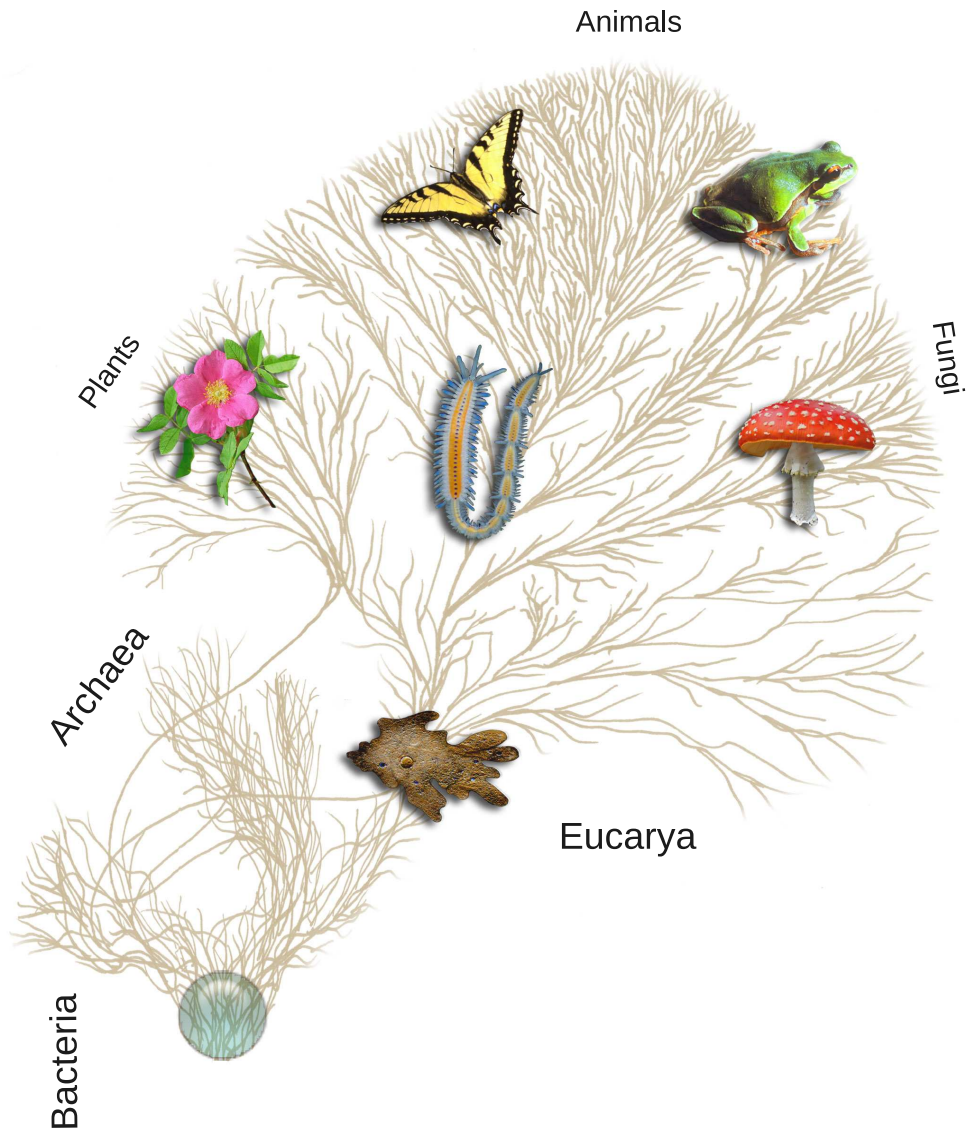

Figure 4.2: Artwork "Tree of Life", taken from the Tree of Life web project site, whose goal is to contain a page with pictures, text, and other information for every species and for each group of organisms, living or extinct; confer [MSM07] for an article describing this web project. The original diagram is supplemented by the classifications to the three domains Archaea, Bacteria, and Eucarya.

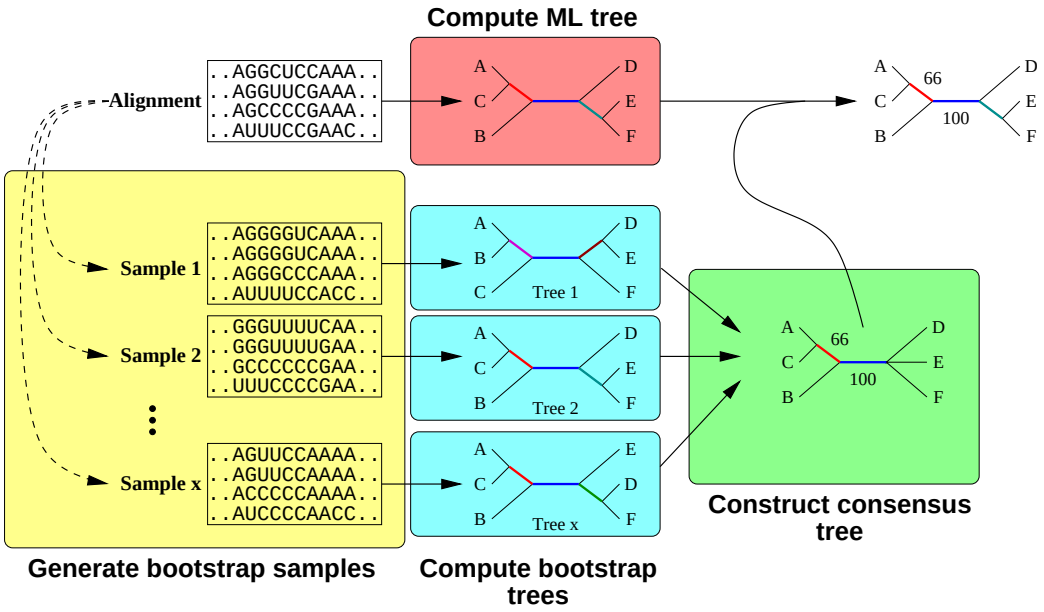

Figure 4.3: Parallel phylogenetic workflow, consisting of the steps sampling phase (*yellow*), tree phase (*red*) and (*cyan*), and consensus phase (*green*). This workflow will be implemented and evaluated applying a Condor-based campus grid.

Nonparametric bootstrap analysis [EHH96, Fel85] is one of the most popular quality assessment approaches, here it constitutes an easy scientific workflow consisting of three phases. We start from an initial sequence alignment, and in a first step (*sampling phase*), a large number of pseudo-samples is created by re-sampling with replacement from the original alignment. Then, for each pseudo-sample a tree is reconstructed (*reconstruction phase*). After all trees have been reconstructed from the pseudo-samples, this large set of trees is summarized in a consensus tree (*consensus phase*). The support values gained by bootstrap analyses are then usually transferred to the tree reconstructed from the whole data, in order to reflect the (un)certainty contained in the reconstructed branches. See Figure 4.3 for a visualization of this process.

Chapter 5

Sequential Case Studies

In this chapter, we discuss sequential (serial) numerical computations; such computations reside in a single process on a single CPU core. Although almost all modern computers feature a parallel hardware architecture, sequential computations remain very important. The following aspects play a major role. (∗) Parallel and distributed numerical programs always entail some overhead for communication between computing processes [Dem97, p.75]. Consequently, in situations where this overhead is relatively big, parallel implementations of algorithms may sometimes be unfeasible. In the extreme case, a parallel implementation on a parallel system may even perform worse than a sequential one. (∗) As a consequence of the communication overhead, the development of parallel algorithms and their implementation is usually significantly more complex than the development of their sequential counterparts. Due to parallel codes being more complex to develop than serial ones, a serial implementation of an algorithm may still precede its parallel version, in order to obtain a rapid prototype. (∗) As sequential systems usually consist of less complex hardware components, sequential computations fail less frequently than computations residing on parallel systems – this aspect is of capital importance on huge systems like clusters featuring thousands of CPU cores. In the worst case, one may have to restart the complete computation, even if just one CPU core fails. (∗) The performance of a serial implementation of an algorithm is easier to predict than the performance of its parallel version. Especially the performance of an implementation on different parallel hardware architectures is difficult to predict [ZKK04, CKPN00].

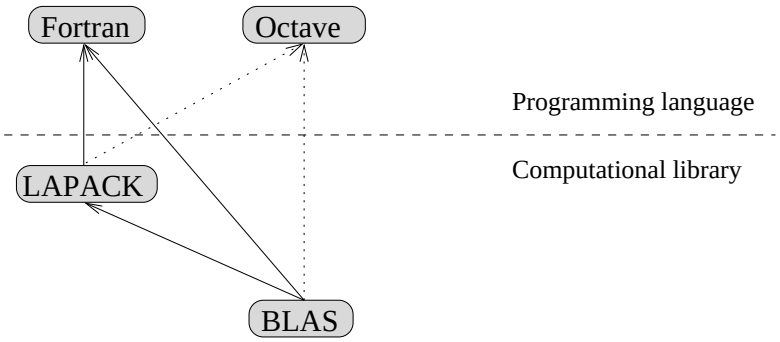

Figure 5.1: Our sequential codes have been mainly written in Fortran, some prototypical parts were first written in GNU Octave which internally uses BLAS and LAPACK. Subroutines from BLAS and LAPACK have been utilized as much as possible in Fortran programs, arrows indicate that routines from external libraries are called.

5.1 Sequential High Performance Programming

The three main issues in designing a piece of numerical software are ease of use, reliability, and speed [Dem97, p.6]. The choice of the programming language should be done carefully. Traditionally, Fortran is the dominating language for the sequential programming of numerical software on supercomputers [Wil88, PZA86]. In comparison with other general-purpose programming languages, Fortran has the most advanced mathematical operators and functions and is therefore optimal for numerical programming. One of the most important language features of Fortran is the support for vector- and matrix-operations out of the box [For03]. For example, C and C++ do a priori not support the handling of such structures.

One has to be careful in order to write a fast serial Fortran program, that is, to develop a good implementation of an appropriate algorithm. Due to the complexity of modern computing platforms, writing numerical code that achieves the best possible performance is extremely difficult [CFP08]. Software that especially aims at one specific computational platform, may perform worse on another. In order to encapsulate routines that may perform very differently on diverse platforms, existing packages for picking applicable subroutines and functions should be considered. For the field of numerical linear algebra, the BLAS is a very effective package to build numerical software. For example, the intrinsic Fortran `matmul` operation for the multiplication of two matrices is usually slower than calling the applicable routine

from the BLAS; see for instance [EGKU99][1]. We see the following reasons for this observation. (∗) The BLAS has been developed by experts in the area of numerical computation in order to make a set of basic mathematical operations as fast as possible. (∗) There is not only one routine for matrix multiplication in the BLAS, but different variants including dense, Hermitian, symmetric, and triangular matrices are available. By calling the particularly suitable routine, the user delivers the additional information, what type of multiplication should be performed. (∗) Optimized BLAS versions tailored for individual architectures are available and should be applied. However, for trying codes a first time, the built-in `matmul` is sufficient. As a general rule of thumb, considering structural properties of involved matrices (for example, symmetric, Hermitian, triangular, tridiagonal, ...) and picking corresponding subroutines is an important principle for fast codes.

5.2 Generalized Complex Symmetric EVP

This section is partly based on our paper "Tridiagonalizing Complex Symmetric Matrices in Waveguide Simulations" [GSPF08].

Generalized complex symmetric EVPs are a special variant of generalized complex non-Hermitian EVPs. In the following, we discuss methods for efficiently solving generalized complex symmetric (non-Hermitian) EVPs. Given matrices A, $B \in \mathbb{C}^{n\times n}$ with $A = A^\top$ (but $A \neq A^*$) and $B = B^\top$ (but $B \neq B^*$), the objective is to efficiently compute eigenvalues λ_k and eigenvectors x_k of the generalized complex symmetric EVP, such that $Ax = \lambda Bx$; see Section 2.2 for basics in numerical linear algebra that are relevant for this thesis.

5.2.1 Relevant Literature

A general introduction to numerical methods for large EVPs can be found, for instance, in the textbook "Numerical Methods for Large Eigenvalue Problems" by Y. Saad [Saa92]. B. Parlett describes the symmetric EVP in his classic book "The Symmetric Eigenvalue Problem" [Par98, Par80]. An overview of the different types of EVPs, as well as templates for their solutions, is given in the book "Templates for the Solution of Algebraic Eigenvalue

[1] The GNU Compiler Collection (GCC) v4.3 supports the command line parameter `-fexternal-blas` generating calls to BLAS routines for intrinsic matrix operations such as `matmul` rather than using the built-in algorithms; see list of changes, new features, and fixes for version 4.3, `http://gcc.gnu.org/gcc-4.3/changes.html`.

Problems: A Practical Guide" edited by Bai et al. [BDD+00]. Various further textbooks cover the topic, for example [Dem97, TB97, GL96, Hig96]. An early paper "$Ax = \lambda Bx$ and the Generalized Eigenproblem", published by G. Peters and J. Wilkinson, discusses basic ideas for generalized EVPs [PW70].

5.2.2 Related Work

Papers especially aiming at solving dense generalized complex symmetric EVPs are very rare, but sub- or related problems are discussed in the following papers. Some work has been done for solving standard EVPs using COTs, including [BOP98, BOR97, LQ97], or using a modified Jacobi method [Sea69]. Some papers discuss methods for generalized sparse complex symmetric EVPs, including the Jacobi method [LL92], subspace iteration [Leu95], or variants of the Jacobi-Davidson method [AC08, AH04]. In [LL92], a generalized complex symmetric eigensolver is discussed, based on the generalized Jacobi method. In this paper, two very small examples are given, but neither accuracy nor runtimes are evaluated. To the best of our knowledge, there are no further codes focussing on dense generalized complex symmetric EVPs.

The most common strategy so far is to ignore the algebraic properties and to apply the technology available for general non-Hermitian EVPs, as demonstrated in `zggev` (LAPACK). In the latter case, this means that firstly B is reduced to triangular form by applying a QR decomposition, then the problem is reduced to generalized Hessenberg form using unitary transformations. From the generalized Hessenberg form, eigenvalues and eigenvectors are computed with the QZ algorithm [MS73].

Concluding existing related work, we observe that methods and both serial and parallel codes especially tailored for generalized complex symmetric EVPs are very rare, as are evaluations of such codes. Consequently, we investigate solver approaches for generalized complex symmetric EVPs on state-of-the-art computer infrastructures in serial (this chapter) and parallel (Chapter 6).

5.2.3 Motivation

Although generalized complex symmetric EVPs do not occur as frequently in practice as real symmetric or complex Hermitian EVPs, there are some important applications where they arise [HJ85, p.201]. Concrete examples are the numerical solution of Maxwell's equations with complex material coefficients (accounting for losses) or certain absorbing boundary conditions used in the simulation of optoelectronic devices [AC08, GSPF08, FPB07,

AH04], the complex scaling method (also called complex-coordinate method, complex rotational method, or dilatation analyticity) in quantum mechanics [Moi98, BOR97, OM92, Rei82], and the application of the complex absorbing potential method [RM93].

The conventional approach for solving such EVPs serially, as implemented, for instance, in `zggev` (LAPACK), is to treat complex symmetric EVPs as general complex and therefore does not exploit the structural properties.

Our efforts are particularly motivated by the simulation of guided-wave multisection devices in optoelectronics, see Chapter 4. Techniques for numerically solving Maxwell's equations in this context lead to dense generalized complex symmetric EVPs, where reduced accuracy requirements provide an opportunity for trading accuracy for performance.

5.2.4 Methodology

The main challenge is to find ways for utilizing the structural symmetry in the absence of the mathematical properties of Hermitian matrices. Analogously to Hermitian EVPs, one possible approach for solving generalized complex symmetric EVPs $Ax = \lambda Bx$ starts with reducing it to standard form $My = \lambda y$. Complex symmetry allows for special techniques in this reduction step, confer a Cholesky-based factorization $B \mapsto LL^\top$ (see Section 5.3.1). Subsequently, a tridiagonalization process is performed on the standard EVP which results in a similar complex symmetric tridiagonal EVP $Tz = \lambda z$. After this tridiagonalization step, eigenvalues and eigenvectors of T are computed and the eigenvectors z are backtransformed to those of the original problem x.

Principal steps of our approach are visualized in Figure 5.2, the corresponding high-level methodology can be found in Algorithm 1. The procedure is divided into subproblems of computing a symmetric factorization of B, transforming from a generalized to a standard EVP, solving the standard EVP, and backtransforming eigenvectors in case they are desired. This procedure follows the structure of `dsygv` (LAPACK) for solving real symmetric EVPs.

5.2.5 Aspects of Context Adaptivity

The concept of context adaptivity has been introduced in Chapter 3, here we discuss it with respect to the serial generalized complex symmetric eigensolver. We discuss main goals, accompanying goals, context, methodology,

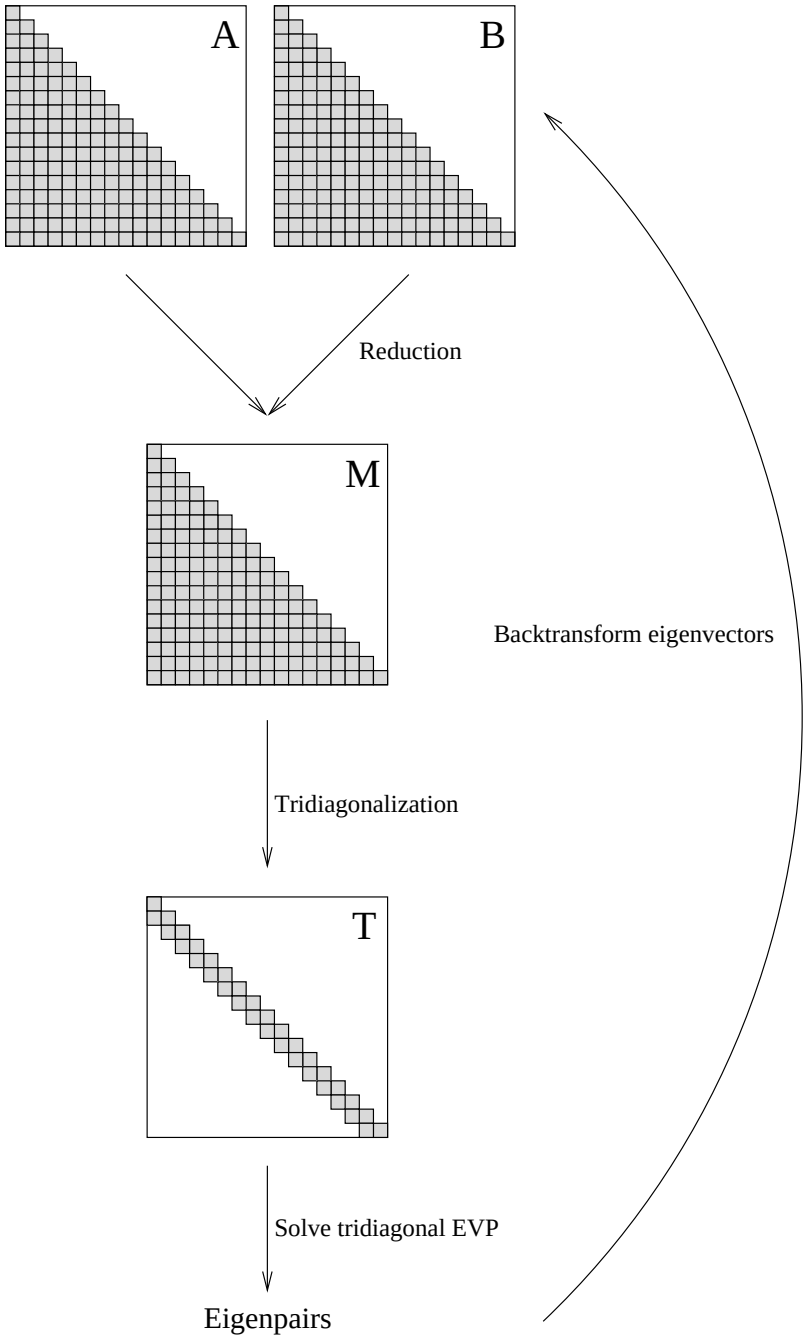

Figure 5.2: Steps to solve a generalized complex symmetric EVP. The generalized EVP (A,B) is transformed to a standard EVP M (*top*); M is reduced to tridiagonal form (*middle*); eigenpairs of the tridiagonal EVP are computed (*bottom*), eigenvectors are backtransformed. Individual boxes ■ indicate necessary entries: for A, B, and M only lower (or upper) triangle is needed; for T only diagonal and subdiagonal (= superdiagonal) are necessary.

Algorithm 1 Solve generalized complex symmetric EVP

Require: A, $B \in \mathbb{C}^{n \times n}$; $A = A^\top, B = B^\top$
1: Transform to standard EVP
2: Compute tridiagonal matrix T
3: Compute eigenvalues and eigenvectors of T
Ensure: $Tz = \lambda z$
4: **return** eigenvalues λ_k
5: Compute eigenvectors of standard EVP
Ensure: $My = \lambda y$
6: Compute eigenvectors of generalized EVP
Ensure: $Ax = \lambda Bx$
7: **return** eigenvectors x_k

and sketch the implemented computational kernels. In this section, *emphasized* words correspond to discussed terms in Chapter 3.

Main goals

The main goal motivating our two driver routines `zsygvn` (featuring a non-splitting tridiagonalization) and `zsygvs` (featuring a splitting tridiagonalization) are lower *runtimes* in comparison with the general solver `zggev` (LAPACK). Detailed evaluations of the complete solvers as well as of corresponding computational kernels will be done. A further goal is less *memory consumption*, as obviously only about half of the storage is required for symmetric matrices.

Accompanying goals

Accompanying goals *portability*, *reusability*, *robustness*, *usability*, and *architecture adaptivity* are partly followed by building our approach on top of existing BLAS and LAPACK routines.

Context

The two developed eigensolver variants respect context particularly by taking the *structure of input data*, namely symmetry, into account. The *computing platform* is any sequential computer system, there are no *temporal requirements* for our computations. Applied *precision* is double precision for real computations and double complex for complex computations. *Accuracy requirements* primarily depend on the application, a discussion according

reduced accuracy requirements of eigenpairs for an application in optoelectronics can be found in Section 4.1.

The new methods are adaptive to the context in the sense that they present a complementary approach with different potential runtime- and accuracy behavior to solve generalized complex symmetric EVPs. The tridiagonal solver implemented in `zstev` is adaptive in a sense that the maximum number of iterations has to be specified.

Methodology

Two new approaches consisting of *algorithmic variants* in individual computational kernels are presented to solve generalized complex symmetric EVPs. Different *implementation variants* are transparently realized through the utilization of the BLAS, which guarantees excellent runtime behavior of core routines on many platforms. *Trading* is demonstrated in a sense that potentially inaccurate methods are applied for the benefits of lower runtimes and less memory consumption.

Computational kernels

The investigated computational kernels are represented by the steps of solving the EVP, see Algorithm 1. Involved computational kernels are (∗) the indefinite factorization in `zpotrfi`, (∗) transformation to standard EVP in `zsygst`, (∗) splitting tridiagonalization in `zsytr1` and non-splitting tridiagonalization in `zsytr2`, (∗) `compev` to compute eigenvalues and `inverm` to compute eigenvectors, and (∗) existing approaches for backtransformation as parts of the standard solvers in `zsyevn` and `zsyevs` and as parts of the generalized solvers in `zsygvn` and `zsygvs`.

5.2.6 Hard- and Software Infrastructure

We apply the following hard- and software for evaluating our serial codes.

Hardware and computational libraries

The serial codes of this chapter were run on an SMP machine Sun Fire v40z with 4 dual-core Opteron 875 CPUs (2.2 GHz) and 24 Gbyte main memory. Only a single core was used and remaining cores were, to the extent possible, idle. See Figure 5.3 for an image depicting an identical machine, found on the webpage of the manufacturer Sun. Suse Linux Enterprise Server 10, the GNU Fortran (GCC) 4.1.2 compiler, LAPACK 3.1.1, Goto BLAS 1.20, and the AMD Core Math Library (ACML 4.0.1) were used. The choice for these two

Figure 5.3: An SMP machine Sun Fire v40z was used for all sequential computations in this chapter. It features 4 dual-core Opteron 875 CPUs (2.2 GHz) and 24 Gbyte main memory, running Suse Linux Enterprise Server 10. Image taken from the homepage of the manufacturer.

BLAS implementations was based on the following arguments: ACML has been developed by the manufacturer of the used Opteron CPU (AMD). Goto BLAS contains hand-tuned optimizations for various architectures, including Opteron. Our serial codes are mainly built on BLAS and LAPACK codes, GNU Octave has been used for rapid prototyping; see Figure 5.1 for the software building blocks of this chapter.

Optimized BLAS

An optimized BLAS is crucial for fast solver components implementing linear algebra functions. In order to demonstrate different runtime characteristics of linear algebra subroutines, we experimented with AMD Core Math Library (ACML) 4.0.1 and Goto BLAS 1.20. Experiments calling the general dense standard solver routine `zgeev` (LAPACK) illustrate very different runtimes for these two BLAS implementations. On our testsystem, we observe that computing eigenpairs with `zgeev` (LAPACK) linked against Goto BLAS is considerably faster than linked against ACML. For instance, for an EVP of order $n = 4000$, `zgeev` (LAPACK) linked against ACML needs about 8526 seconds, while `zgeev` (LAPACK) linked against Goto BLAS needs only about 1044 seconds. This is surprising, as the codes were running on an Opteron CPU which is manufactured by AMD. More details about these findings can be found in [GSPF08].

Self developed codes

Our codes are mainly written in Fortran 90 with a few parts following Fortran 77 standard. All evaluated parts have been implemented utilizing as much functionality of BLAS and LAPACK as possible. To the extent possible, sources of existing routines have been adopted and modified where necessary. Except for the splitting tridiagonalization (see Section 5.4) and tridiagonal solvers (see Section 5.5), both blocked and unblocked codes have been implemented. In general, determining the optimal blocking factor is not a trivial task, confer [Wha08] how to empirically tune it for increased performance. The used optimal blocking factor of LAPACK on the Sun was experimentally determined and set to 32.

The functionality of our routines corresponds to the functionality of comparable LAPACK routines. We implemented the following new sequential routines: `zsygvn` and `zsygvs` solve a generalized complex symmetric EVP featuring a non-splitting or splitting tridiagonalization, respectively. `zsyevn` and `zsyevs` are corresponding solvers for standard complex symmetric EVPs. A complex symmetric factorization based on a real symmetric Cholesky routine is called `zpotrfi`, the actual reduction of a generalized to a standard EVP is performed in `zsygst`. `zsytr1` is the name of the splitting tridiagonalization routine, and `zsytr2` is its non-splitting counterpart. `zgeevp` is a patched version of `zgeev` (LAPACK) to solve complex symmetric tridiagonal EVPs, skipping the reduction to Hessenberg form. `zstev` solves complex symmetric tridiagonal EVPs by calling appropriate routines `compev` and `inverm`[2].

The compiler switches for testing include `-g -Wall -march=opteron -O0 -fbounds-check -g -m64`, for evaluating we apply `-Wall -march=opteron -O3 -m64`[3].

5.2.7 Evaluation Strategy

All major routines are evaluated separately for problem sizes of order 100 to 1000 in steps of 100, and from 1500 to 4000 in steps of 500. Time measurement is done by calling `MPI_Wtime`, where pauses between evaluated operations aim at abolishing all possible cache effects of previous operations. `MPI_Wtime` was also used for serial timings to have a unique routine for both the serial and the parallel case. Pauses were done by calling Fortran intrinsic function `sleep`. Accuracy is measured in different ways, for example, by computing an eigenvalue error and a residual error.

[2] confer the codes by Cullum and Willoughby, `http://netlib.org/lanczos/`

[3] see GCC manual, `http://gcc.gnu.org/onlinedocs/`

We compare our serial routines with corresponding LAPACK routines. With respect to accuracies, we compare with routines that process computational problems of the same type; for example, we test complex symmetric factorization with a BK factorization routine, but we do not compare it with the accuracy of the Hermitian variant. With respect to runtimes, we compare with routines that operate on the same type (Hermitian, but not real symmetric) and that perform a similar operation; for example, we test complex symmetric factorization with a Hermitian factorization.

Test matrices

Here we describe the generation of test matrices, as they are important for accuracy. For dense complex symmetric EVPs, input matrices A are constructed by calling Fortran function `random_number` to generate pseudo-random numbers in the interval $[0 \dots 1]$, for the real and imaginary part separately. Subsequently, a complex symmetric matrix is achieved by computing $A + A^\top$. The 2-norm of such problems of the order $n = 100$ is about 142 and 5657 for $n = 4000$. Our matrices for complex symmetric tridiagonal EVPs feature considerably smaller norms in the range of about $2 \dots 3$. Only for the full solver (generalized EVP), do we generate pairs of matrices that we denominate as "SP" being discussed in Section 5.7.3.

Normalized runtimes

Most of the discussed algorithms feature $\mathcal{O}(n^3)$ runtime behavior, where n is the order of the problem. Therefore, to make the differences in the figures between the individual implementations clearly visible, the runtimes are uniformly normalized as $T(n)\frac{10^{10}}{n^3}$.

5.3 Transformation to Standard EVP

The transformation from generalized EVP $Ax = \lambda Bx$ to standard EVP $My = \lambda y$ includes the two steps (a) symmetric factorization of B and (b) actual reduction. In the following, we describe these two principal steps separately.

5.3.1 Symmetric Indefinite Factorization

In the course of generalized EVPs, we refer to the term factorization as the computation of a matrix product featuring simpler matrices to substitute B.

The applied factorization is based on the Cholesky factorization $B \mapsto LL^\top$ and implemented in `zpotrfi`.

Relevant literature

Basics about the Cholesky factorization can be found, for example, in [Bre06] and in various textbooks, including [Dem97, p.78], [TB97, Lecture 23], [Ü97b, p.235], [GL96, p.143], and [Hig96, p.204]. An error analysis of Cholesky factorizations for $n \times n$ symmetric positive definite matrices was done in [BDM89]. An efficient implementation in LAPACK was described in [CDO+96]. Different variants have been proposed, see [GL96, p.143–146].

Related work

The performance of `dpotrf` (LAPACK), performing a Cholesky factorization of a real symmetric positive definite matrix, was discussed in [EGKU99]. In [Ber05, Ber03], a complex symmetric factorization is investigated in the course of solving linear systems of equations $AX = B$. Unfortunately, evaluations only cover the full solver and do not tell details about the performance of the factorization. The paper is interesting, because it argues that a complex symmetric factorization does not necessarily rely on pivoting (which is commonly applied for such problems). The method for complex symmetric factorization is similar to the one we implemented.

J. Bunch and L. Kaufman discussed the Bunch-Kaufman (BK) factorization featuring a pivoting strategy in a paper aiming at solving symmetric indefinite linear systems [BK77]. A serial implementation of this algorithm applicable to complex symmetric matrices is available in `zsytrf` (LAPACK). The BK factorization is usually the method of choice for factorizing symmetric indefinite matrices [JP94].

Methodology

The usual way of factorizing indefinite B is done by a symmetry-preserving BK factorization. Although the Cholesky method is usually restricted to positive definite Hermitian matrices, it can also be applied to some complex symmetric matrices [Ber05, Ser80]. One such class consists of matrices $A = B + iC$, where B and C are both real, symmetric, and positive definite. In general, applying a Cholesky factorization to a non-positive definite matrix may impose ill-conditioned factors L. Nevertheless we investigate a Cholesky based factorization, as due to avoiding a pivoting strategy, a higher parallel performance than for BK factorization is assumed; confer, for instance, [Bis88] for a paper about performance problems with parallel pivoting.

A Cholesky factorization is applied to factorize a given symmetric positive definite matrix $B \mapsto LL^\top$ such that L is lower triangular (sometimes called Cholesky triangle), confer Definition 2.2.15; an analogous definition can be given to factorize $B \mapsto U^\top U$ such that U is upper triangular. In the complex Hermitian case ($B \mapsto LL^*$ or $B \mapsto U^*U$), the matrix B has to be positive definite in order to guarantee numerical stability.

Implementation

Our factorization is based on the real symmetric factorization in LAPACK routine `dpotrf`. In contrast to `dpotrf`, our matrices are complex symmetric indefinite, therefore the test for positive definiteness is skipped and some small modifications are necessary to handle complex matrices instead of real ones. The new routine is called `zpotrfi`, where the name indicates that it is based on the Cholesky factorization, but aiming at indefinite matrices.

`dpotrf` (LAPACK) includes blocked and unblocked codes, whereof we are going to describe the unblocked version. Algorithm 2 describes the basic Cholesky factorization, as it can be found, for example, in [HHL07, Luc04]. Apart from the skipped check for positive definiteness in Lines 4–6, the same algorithm is implemented in `zpotrfi`.

Algorithm 2 Cholesky factorization $B \mapsto LL^\top$

Require: $B \in \mathbb{C}^{n\times n}, B = B^\top$

1: $L \leftarrow$ lower triangle of B {Initialize}
2: **for** $k \leftarrow 1$ to n **do**
3: $\quad L(k,k) \leftarrow L(k,k) - L(k,1\colon k-1)L(k,1:k-1)^\top$
4: $\quad$ **if** $(L(k,k) \leq 0)$ **then**
5: $\quad\quad$ stop {Not positive definite}
6: $\quad$ **end if**
7: $\quad L(k,k) \leftarrow \sqrt{L(k,k)}$
8: $\quad$ **if** $(1 < \mathrm{k} < \mathrm{n})$ **then**
9: $\quad\quad L(k{+}1:n,k) \leftarrow L(k{+}1:n,k) - L(k{+}1:n,1:k{-}1)L(k,1:k{-}1)^\top$
10: $\quad$ **end if**
11: $\quad$ **if** $(\mathrm{k} < \mathrm{n})$ **then**
12: $\quad\quad L(k+1:n,k) \leftarrow \frac{1}{L(k,k)} L(k+1:n,k)$
13: $\quad$ **end if**
14: **end for**
15: **return** factor L

Accuracy

A factorization error of our routine `zpotrfi` is computed according to $\mathfrak{E}_F = \frac{\|B-LL^\top\|_2}{\|B\|_2}$. It is compared with the performance of the competitor routine for BK factorization `zsytrf` (LAPACK), whose accuracy is evaluated in terms of $\mathfrak{E}_{BK} = \frac{\|X-I\|_2}{\|B\|_2}$; here, X is the solution of the equation $LDL^\top X = B$, where X is computed by calling the linear equations solver `zsytrs` (LAPACK). The analogous evaluation of the BK routine calculating $B - LDL^\top$ has not been realized, as `zsytrf` does not explicitly return L and D.

Figure 5.4 demonstrates that measured accuracy of both routines is very similar for the applied random matrices. However, it has to be stated that slightly different metrics to evaluate accuracy have been applied, out of these two, a clearly more accurate routine cannot be given. Due to its pivoting strategy, LAPACK routine `zsytrf` may be preferred for ill-conditioned problems.

Runtimes

Figure 5.5 depicts evaluated routines according to their runtime behavior. We observe that `zpotrfi` is always faster than `zsytrf` (LAPACK), yet the difference is small. As expected, `zpotrf` (LAPACK) features practically the same runtime behavior as `zpotrfi`.

5.3.2 Reduction Operation

Subsequently to the symmetric factorization, the actual transformation from the generalized to standard EVP is performed.

Related work

The procedure of reducing a generalized EVP to a standard EVP is mostly covered in textbooks and papers dealing with the solution of generalized problems, see Section 5.2.1. An early paper especially aiming at this reduction process was published by R. Martin and J. Wilkinson [MW68].

Methodology

In the presence of efficient solvers for standard EVPs, it appears obvious to start by reducing from the generalized and solving the corresponding standard EVP. The first step is achieved by performing a symmetry preserving factorization of B (see Section 5.3.1). Subsequently, we compute $M = L^{-1}AL^{-\top}$, where M is the input matrix for the standard EVP, and

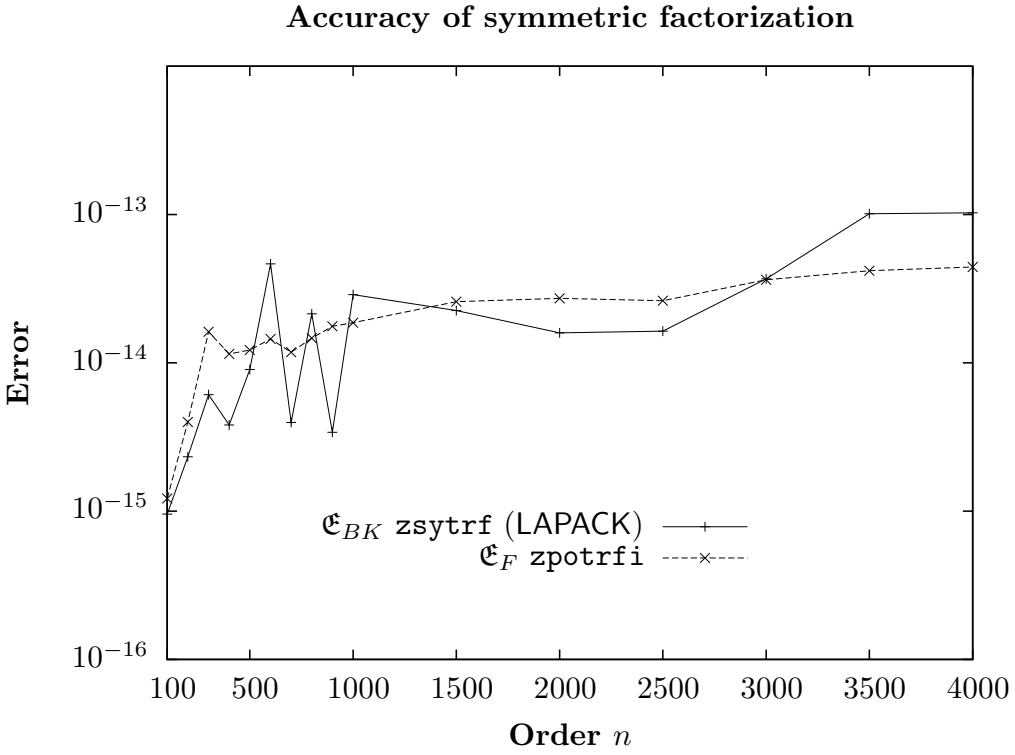

Figure 5.4: Accuracy of routine `zpotrfi` factorizing $B \mapsto LL^\top$, evaluated according to $\mathfrak{E}_F := \frac{\|B-LL^\top\|_2}{\|B\|_2}$; competitor BK routine `zsytrf` (LAPACK) is factorizing $B \mapsto LDL^\top$ and evaluated according to $\mathfrak{E}_{BK} := \frac{\|X-I\|_2}{\|B\|_2}$, where X has been computed as the solution of the equation $LL^\top X = B$ calling `ztrtrs` (LAPACK).

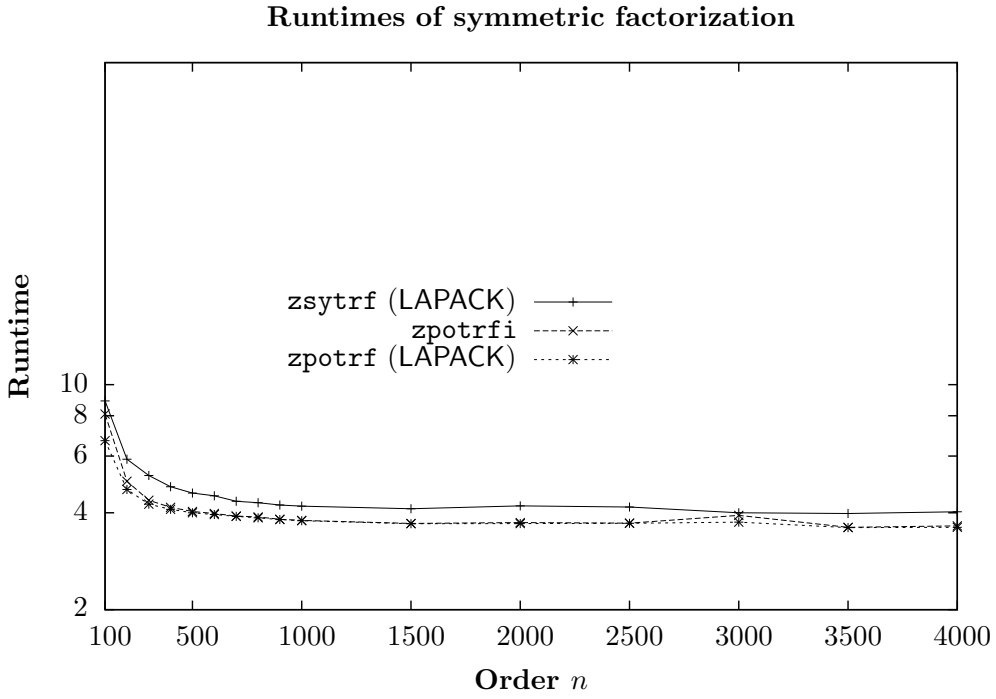

Figure 5.5: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of factorization routine `zpotrfi` compared to `zsytrf` (LAPACK) and `zpotrf` (LAPACK).

Reduction of EVP(A,B) to EVP(M).

$$\begin{aligned} Ax &= \lambda \underbrace{B}_{LL^\top} x \\ L^{-1}\cdot \parallel Ax &= \lambda L L^\top x \\ L^{-1} \underbrace{A}_{AL^{-\top}L^\top} x &= \lambda \underbrace{L^{-1}L}_{I} L^\top x \\ \underbrace{L^{-1}AL^{-\top}}_{M} \underbrace{L^\top x}_{y} &= \lambda \underbrace{L^\top x}_{y} \\ My &= \lambda y \end{aligned}$$

□

Figure 5.6: Transformation from generalized EVP (A,B) to standard EVP (M).

Computation of M.

$$\begin{aligned} L\cdot \parallel M &= L^{-1}AL^{-\top} \\ LM &= \underbrace{LL^{-1}}_{I}\underbrace{AL^{-\top}}_{H} \\ LM &= H \end{aligned}$$

□

Figure 5.7: Computation of M; the resulting problem $LM = H$, where L is triangular, can be solved once H has been computed.

Computation of H.

$$\begin{aligned} H &= AL^{-\top} \parallel \cdot L^{\top}, \text{transpose both sides} \\ LH^{\top} &= A^{\top} \parallel \text{Compute H by calling } \texttt{ztrtrs} \text{ (LAPACK)} \end{aligned}$$

□

Figure 5.8: Computation of H; the resulting equation $LH^{\top} = A^{\top}$ can be solved by calling `ztrtrs` (LAPACK), where $A^{\top} = A$; finally, H is computed by transposing $H^{\top}$

.

$y = L^\top x$. See Figure 5.6 for the full reduction to a standard EVP and back-transformation of eigenvectors, Figure 5.7 for the computation of M, and Figure 5.8 for its intermediate matrix H.

Implementation

The implementation of the reduction routine `zsygst` is a series of calls to LAPACK routines. Our implementation is algorithmically slightly different from the one in the LAPACK routine `dsygst`, which applies a transformation to the input matrix from both sides at the same time: our solution implemented in `zsygst` is a simplified preliminary implementation of this operation for the purpose of rapid prototyping. `zsygst` consecutively solves two linear systems of the type $LX = B$ with L from the factorization step in order to construct $M = L^{-1}AL^{-\top}$. In more detail, first `ztrtrs` (LAPACK) is used for solving $LX = A$ for X, yielding $X = L^{-1}A$. Then, X is transposed, and `ztrtrs` (LAPACK) is used again for solving $LM = X$ for M, so $M = L^{-1}X = L^{-1}AL^{-\top}$.

Accuracy

The transformation of a generalized complex symmetric EVP to standard form is evaluated in terms of runtimes only, as accuracy can hardly be separated from the factorization process. The accuracy of the full transformation process, comprehending factorization and actual reduction, is evaluated in Section 5.7.1.

Runtimes

The runtimes of the routine `zsygst` are depicted in Figure 5.9 and compared to its LAPACK counterpart `zhegst` for Hermitian problems. `zhegst` (LAPACK) is unsurprisingly faster, a behavior that we attribute to the more efficient reduction procedure.

Upon measuring runtimes of individual parts of `zsygst`, we observe a domination of the computational solution of $AX = B$ being called twice in the routine and implemented in `ztrtrs` (LAPACK). A transposition operation consumes comparatively little time, so it can be ignored in terms of computational effort.

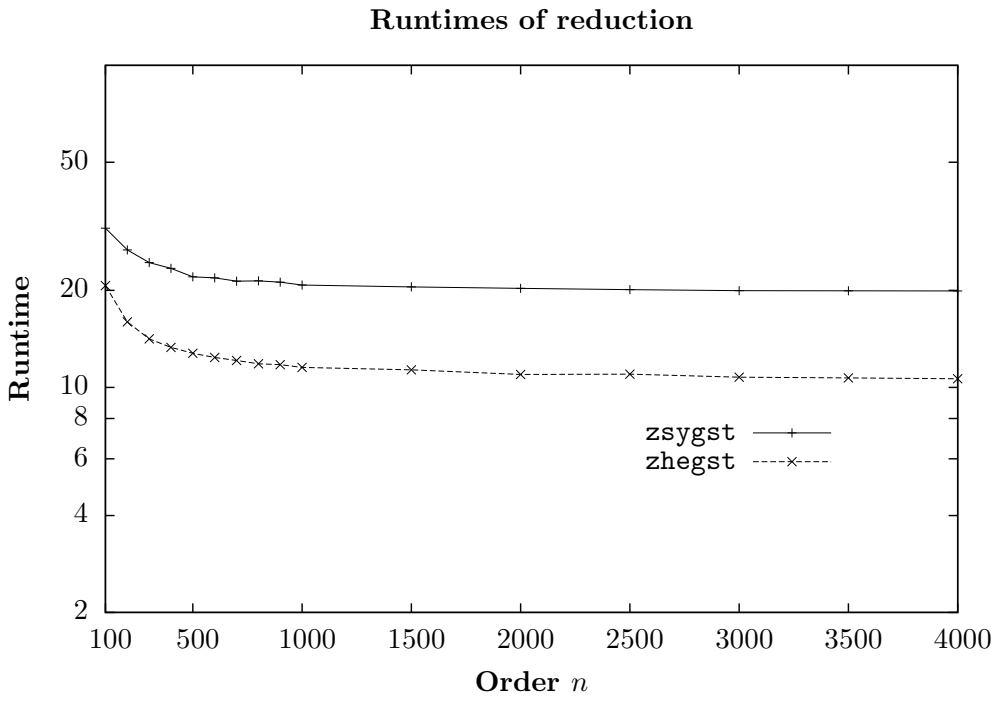

Figure 5.9: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of complex symmetric reduction routine `zsygst` and Hermitian pendant `zhegst` (LAPACK).

5.4 Tridiagonalization

Starting from a standard EVP $My = \lambda y$, we iteratively eliminate entries of M, resulting in a tridiagonal matrix T. T is similar to M, hence it has the same eigenvalues. In case that eigenvectors are desired, they have to be backtransformed to the original problem.

5.4.1 Related Work

A modified conventional Householder-based reduction method has been described in [OM92], but the paper does not report about achieved accuracies or measured runtimes. The tridiagonalization of a dense complex symmetric matrix by means of COTs has been investigated in [BOR97]. The latter paper lists measured runtimes and accuracy for computing eigenvalues and eigenvectors for relatively small complex symmetric EVPs of the order $n = 500$, 1000, and 2000. In terms of accuracy, comparison is difficult as different metrics were used. The authors compare runtimes of their tridiagonalization process to a Hermitian pendant `htridi` (EISPACK), observing speedups from 2.1 to 2.6. However, comparison with our routines is difficult as a meanwhile retired DEC AXP 3000-500 was utilized.

Splitting and non-splitting approaches

For tridiagonalizing a complex symmetric matrix M, two basic approaches can be distinguished: the splitting and the non-splitting method [GGP09]. The former approach, which has been discussed in [GSPF08, BOR97], is based on separating the tridiagonalization of the real part of M from the tridiagonalization of the imaginary part of M (which are both real symmetric matrices) as much as possible. The latter approach is an alternative which operates on the complex symmetric matrix as a whole, based on generalizations of complex Householder reflectors. The codes for the real symmetric (`dsytrd` (LAPACK)) and for the complex Hermitian (`zhetrd` (LAPACK)) case use orthogonal and unitary transformation matrices, respectively. In our method for the complex symmetric case, we need to use complex orthogonal transformation matrices with norms potentially larger than one. This leads one to expect a loss of numerical accuracy.

5.4.2 Splitting Tridiagonalization

Splitting tridiagonalization is one out of two discussed approaches to tridiagonalize a given input matrix M. Algorithm 3 depicts the high-level approach

of this procedure.

Methodology

The real part R and the imaginary part S of a complex symmetric matrix $M = R + iS$ are real symmetric matrices. In each iteration, the following steps are accomplished. Firstly, a column of R can be eliminated using a real orthogonal Householder transformation Q_R. After that, a smaller part of the corresponding column of S can be eliminated without causing any fill-in in R using another real orthogonal Householder transformation Q_I. Both of these operations are performed in real arithmetic, and both transformation matrices have unit norm. Eventually, a single nonzero element below the subdiagonal in S remains to be eliminated. This operation has to be performed in complex arithmetic, using a 2×2 COT, whose norm cannot be bounded a priori. In order to preserve symmetry, complex orthogonal similarity transformations (COTs) Q are needed which satisfy $Q^\top Q = I_n$. In general, $\|Q\|_2 \geq 1$ and thus the application of complex orthogonal matrices can increase numerical errors. When the column elimination is first performed in R and then in S, we call the procedure RI variant. Analogously, it is possible to eliminate first in S and then in R. We call this procedure IR variant.

Complex orthogonal transformations

The transformation matrix

$$G := \frac{1}{\sqrt{t^2 + s^2}} \begin{pmatrix} t & s \\ -s & t \end{pmatrix}, \quad t = t_1 + it_2 \in \mathbb{C},\ t_1, t_2, s \in \mathbb{R}\ , \tag{5.1}$$

defines a COT since $G^\top G = I_2$. Consequently, $GAG^\top$ is a similarity transformation of A. In the RI variant, a COT G_{RI} has to be determined such that

$$G_{RI} \begin{pmatrix} a + ib \\ ic \end{pmatrix} = \begin{pmatrix} d + ie \\ 0 \end{pmatrix},$$

where $a, b, c, d, e \in \mathbb{R}$ and $c \neq 0$. Choosing the parameters $t = s\left(\frac{b}{c} - i\frac{a}{c}\right)$, $s \neq 0$ arbitrary, the COT is given as

$$G_{RI} = \frac{1}{\sqrt{b^2 - a^2 + c^2 - i(2ab)}} \begin{pmatrix} b - ia & c \\ -c & b - ia \end{pmatrix}\ . \tag{5.2}$$

In the IR variant, a COT G_{IR} has to be determined such that

$$G_{IR} \begin{pmatrix} a + ib \\ c \end{pmatrix} = \begin{pmatrix} d + ie \\ 0 \end{pmatrix}.$$

Tridiagonalization of M.

$$
\begin{aligned}
Q \cdot \parallel My &= \lambda y && (5.4)\\
QM \underbrace{y}_{Q^\top Qy} &= \lambda Qy && (5.5)\\
\underbrace{QMQ^\top}_{T} \underbrace{Qy}_{z} &= \lambda \underbrace{Qy}_{z} && (5.6)\\
Tz &= \lambda z && (5.7)
\end{aligned}
$$

□

Figure 5.10: Proof: transform complex symmetric M to receive complex symmetric tridiagonal T.

With $t = s\left(\frac{a}{c} + i\frac{b}{c}\right)$, $s \neq 0$ arbitrary, the COT is given as

$$G_{IR} = \frac{1}{\sqrt{a^2 - b^2 + c^2 + i(2ab)}} \begin{pmatrix} a+ib & c \\ -c & a+ib \end{pmatrix} . \qquad (5.3)$$

Numerical aspects

In a splitting method, the complex orthogonal transformations (see Equation (5.1)) are the only non-unitary transformations, all other transformations used have unit norm. If $\|G\|_2 \gg 1$ the accuracy of the tridiagonalization process could be influenced negatively. During the tridiagonalization process, monitoring the norms of the COTs makes it possible to detect potentially large errors. Basic concepts have been suggested to avoid large norms, such as the recovery transformations proposed in [BOR97]. The underlying idea is to increase numerical stability by permuting columns via a Jacobi rotation.

Elimination process

Out of the two variants RI and IR splitting tridiagonalization, we focus on the RI variant. An advanced approach dynamically applying either of the two methods is discussed later in this section.

Given a complex symmetric input matrix M, a sequence of $k = 1 \ldots n-2$ symmetry preserving similarity transformations $Q^{(k)}$ are applied to M. The result is a complex symmetric tridiagonal matrix T, see Figure 5.10 for proof of this basic approach. In the following we illustrate the calculation of Q. Q is the matrix of all similarity transformations applied on M in order to

		$H_R^{(1)}$	$H_R^{(1)}$	$H_R^{(1)}$
			$H_R^{(2)}$	$H_R^{(2)}$
$H_R^{(1)}$				$H_R^{(3)}$
$H_R^{(1)}$	$H_R^{(2)}$			
$H_R^{(1)}$	$H_R^{(2)}$	$H_R^{(3)}$		

		$G^{(1)}$	$H_S^{(1)}$	$H_S^{(1)}$
			$G^{(2)}$	$H_S^{(2)}$
$G^{(1)}$				$G^{(3)}$
$H_S^{(1)}$	$G^{(2)}$			
$H_S^{(1)}$	$H_S^{(2)}$	$G^{(3)}$		

Table 5.1: Annihilation process for a matrix of order $n = 5$. Each entry in the matrix corresponds to the similarity transformation and the iteration, in which the according element is annihilated; real part (*left*), imaginary part (*right*). G denotes a COT, H denotes a Householder reflection.

calculate the tridiagonal matrix T. H_R and H_I are real Householder reflectors, applied for annihilating real and imaginary fractions of M, respectively. G is a complex orthogonal transformation (COT). Since $H_R = (H_R)^\top$ and $H_I = (H_I)^\top$, this yields Term (5.8), where M is tridiagonalized to T.

$$\underbrace{\underbrace{\underbrace{G^{(n-2)}H_R^{(n-2)}}_{Q^{(n-2)}}\cdots\underbrace{G^{(1)}H_I^{(1)}H_R^{(1)}}_{Q^{(1)}}}_{Q} M \underbrace{\underbrace{H_R^{(1)}H_I^{(1)}(G^{(1)})^\top}_{(Q^{(1)})^\top}\cdots\underbrace{H_R^{(n-2)}(G^{(n-2)})^\top}_{(Q^{(n-2)})^\top}}_{Q^\top}}_{T}, \quad (5.8)$$

Please notice that in the last step of the tridiagonalization process, there is no Householder reflection needed on the imaginary part of S. In Table 5.1, we show this tridiagonalization process for an example of order $n = 5$; each entry contains the similarity transformation and the step, in which this annihilation is accomplished.

In the case that eigenvectors of M are desired, a backtransformation step follows tridiagonalization. In order to facilitate backtransformation, the tridiagonalization phase has to support it, for example, by explicitly building a transformation matrix Q. Currently, tridiagonalization routine `zsytr1` is capable of building Q, but this is not done in the most efficient way. A more efficient way is implemented in the corresponding routine for real matrices, called `dsytrd` (LAPACK). It iteratively computes and stores scalar factors τ_k and elementary reflectors v_k in order to be utilized for backtransformation afterwards. Currently `zsytr2` does not support building Q, neither does it support any other way of backtransforming eigenvectors.

Algorithm 3 Splitting tridiagonalization (RI variant)

Require: $M \in \mathbb{C}^{n \times n}, M = M^\top$
1: $T \leftarrow M$ {Initialize}
2: **for** $k \leftarrow 1$ to $n-2$ **do**
3: $T_R \leftarrow real(T)$, $T_I \leftarrow imag(T)$ {Split}
4: $T_R \leftarrow Q_R^{(k)} \cdot T_R \cdot Q_R^{(k)}$ {Annihilate $T_R(k+2:n,k)$, $Q_R = (Q_R)^\top$}
5: $T_I \leftarrow Q_R^{(k)} \cdot T_I \cdot Q_R^{(k)}$ {Corresponding update}
6: **if** $k < n-2$ **then**
7: $T_I \leftarrow Q_I^{(k)} \cdot T_I \cdot Q_I^{(k)}$ {Annihilate $T_I(k+3:n,k)$, $Q_I = (Q_I)^\top$}
8: $T_R \leftarrow Q_I^{(k)} \cdot T_R \cdot Q_I^{(k)}$ {Corresponding update}
9: **end if**
10: $T \leftarrow T_R + iT_I$ {Merge}
11: $T \leftarrow Q_G^{(k)} \cdot T \cdot (Q_G^{(k)})^T$ {Annihilate $T_I(k+2,k)$, $Q_G \neq (Q_G)^\top$}
12: **end for**
13: **return** tridiagonal matrix T

RI / IR variants

The order of processing R and S can be determined independently in each iteration of the tridiagonalization process. For both variants, the norm of each COT can be precomputed. Based on this information, the COT with the smaller norm can be selected and the corresponding variant carried out. Obviously, this heuristic choice is only a local minimization and there is no guarantee that it minimizes the accumulated norm of all COTs in the tridiagonalization process.

In order to quantify the potential benefit of dynamically choosing RI or IR variant in each iteration, a prototypical code in GNU Octave has been evaluated, see Algorithm 4. This code features maturity in terms of accuracy, but it is not optimized for speed. Therefore, only accuracy is discussed here.

A complex symmetric matrix M of order $n = 1500$ is constructed by calling GNU Octave function `rand` to generate pseudo-random numbers in the interval $[0 \ldots 1]$, for the real and imaginary part separately. Subsequently, a complex symmetric matrix is achieved by computing $M + M^\top$. Afterwards, the matrix is tridiagonalized to T by (a) solely RI transformations, (b) solely IR transformations, and (c) an adaptive RI / IR algorithm. The latter variant precomputes the 2-norm of transformations performed in each step and chooses accordingly. The maximum relative eigenvalue error is computed according to $\mathfrak{E}_{RIIR} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where λ_k is an eigenvalue of M computed by GNU Octave eigensolver `eig`, and $\tilde{\lambda}_k$ is an eigenvalue of the resulting

tridiagonal matrix T computed by `eig`.

In a concrete experiment, method (a) exhibits $\mathfrak{E}_{RIIR}$ of about $1.78 \cdot 10^{-11}$, an average COT 2-norm of about 4, and all individual COT 2-norms summing up to about 5984; method (b) results in $\mathfrak{E}_{RIIR}$ of about $1.91 \cdot 10^{-11}$, and method (c) gives $\mathfrak{E}_{RIIR}$ about $2.46 \cdot 10^{-11}$, while the sum of norms is again 5894. The product of corresponding individual norms is $\gg 10^{308}$ and therefore not representable in GNU Octave. Further tests with different random matrices reveal that none of these three methods is best. Moreover, we observe that computing the 2-norm of each transformation is not a reliable method to find the best sequence of transformations.

Algorithm 4 Splitting RI / IR variants

Require: $M \in \mathbb{C}^{n \times n}, M = M^\top$

1: $T \leftarrow M$ {Initialize}
2: **for** $k \leftarrow 1$ to $n-2$ **do**
3: **if** $\|\mathrm{COT}_{RI}\|_2 \leq \|\mathrm{COT}_{IR}\|_2$ **then**
4: Apply RI
5: **else**
6: Apply IR
7: **end if**
8: **end for**
9: **return** tridiagonal matrix T

Implementation

The splitting tridiagonalization in `zsytr1` has been done from scratch, as no single LAPACK routine is similar. RI and IR variants are very similar, therefore we describe only the RI variant. Householder transformations are done by calling corresponding LAPACK routines `dlarfg` to compute a real elementary reflector, and `dlarfx` to apply it. In the course of computing the COT, we cannot employ a corresponding LAPACK routine, hence we have to call BLAS routines directly. s can be chosen arbitrarily, the dependent parameter t is computed according to $s(\frac{b}{c} - i\frac{a}{c})$. There is no known effect in terms of numerical stability for choosing s, here, s is set to 1. Finally, the COT is applied by means of BLAS column- and row operations.

5.4.3 Non-Splitting Tridiagonalization

The second approach discussed is the non-splitting tridiagonalization implemented in our routine `zsytr2`.

Methodology

In [Par64, Bud63], tridiagonalization of real unsymmetric matrices was proposed. The basic building block for the non-splitting complex symmetric tridiagonalization method is a generalization of Householder reflectors (see for example [GL96]). As illustrated in [GGP09, OM92], a complex symmetric reflector V for complex symmetric matrices can be defined formally analogously to the Householder reflectors for real symmetric matrices, yielding

$$V = I_n - \frac{2}{x^T x} x x^\top \ ,$$

where the vector $x \in \mathbb{C}^n$ eliminates all but the first entry of a given vector $z \in \mathbb{C}^n$ ($Vz = \hat{z}e_1$, $\hat{z} = \pm\sqrt{(z^\top z)} \in \mathbb{C}$) and is given as

$$\begin{aligned} x_1 &= z_1 - \hat{z} \ , \\ x_k &= z_k, \qquad k = 2, \ldots, n \ . \end{aligned}$$

In numerical routines x is often rescaled so that $x_1 = 1$. Then

$$\begin{aligned} x_k &= \frac{z_k}{z_1 - \hat{z}}, & k \in \{2, \ldots, n\} \ , \\ \frac{2}{x^\top x} &= \frac{\hat{z} - z_1}{\hat{z}} = 1 - \frac{z_1}{\hat{z}} \ . \end{aligned}$$

To avoid numerical errors due to cancellation and to keep x_1 and the $x_{2\ldots n}$ of the same order of magnitude, the sign for $\hat{z}$ is chosen such that $|z_1 - \hat{z}|$ is maximized. Note that $\sqrt{x^\top x}$ for $x \in \mathbb{C}^n$ is not a norm (consider, for example, $(1\ i)(1\ i)^\top = 0$). Thus, symmetry-preserving complex symmetric reflectors are not unitary. This causes numerical instabilities.

A sequential tridiagonalization is realized in the LAPACK routines `dsytrd` for real symmetric EVPs and in `zhetrd` for Hermitian EVPs. A special variant for complex symmetric matrices is missing in LAPACK. We give a brief methodological description of the real symmetric variant `dsytrd` (LAPACK). The principal goal is the reduction of a real dense matrix M to symmetric tridiagonal form T. Either upper or lower type can be chosen and strictly the according part is referenced only. One important feature of the implementation is the realization as blocked code operating on blocks of k rows or columns. In each step, a unitary similarity transformation $Q^\top M Q$ is performed. For that purpose, an elementary reflector $H = I - \tau v v^\top$ has to be computed. The individual updates are given by rank $2k$ operations of the form $M - VW^\top - WV^\top$, operating on k rows or columns. For reductions smaller than the specified blocksize, the last or only block is reduced by an unblocked code.

Implementation

The non-splitting tridiagonalization is implemented in routine `zsytr2`, which is based on the real symmetric tridiagonalization routine `dsytrd` (LAPACK). As described earlier, a fundamental difference between the real symmetric and the complex symmetric tridiagonalization is the generation of the reflector. In the real symmetric case, the Householder reflector is generated in `zlarfg` (LAPACK), while in our complex symmetric case it is realized in our newly developed routine `zlarfgn`.

Evaluation

We are evaluating the splitting tridiagonalization routine `zsytr1` and the non-splitting tridiagonalization routine `zsytr2` in terms of accuracy and runtimes.

Accuracy

Both routines are evaluated in terms of the maximum relative eigenvalue error computed as $\mathfrak{E}_{T1} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where λ_k is an eigenvalue of M computed by `zgeev` (LAPACK), and $\tilde{\lambda}_k$ is an eigenvalue of T computed by `zgeevp`. Furthermore, the quality of the tridiagonalization process is evaluated by computing $\mathfrak{E}_{T2} := \frac{\|T - Q^\top M Q\|_2}{\|M\|_2}$. This evaluation is only done for `zsytr1`, as computing Q is currently not supported in `zsytr2`.

The maximum relative eigenvalue error depicted in Figure 5.11 reveals that splitting tridiagonalization routine `zsytr1` is more accurate than its splitting counterpart `zsytr2`. This observation is significant and holds for all tested orders. The accuracy of the tridiagonalization process in `zsytr1` is relatively stable for all evaluated orders and in the range of 10^{-13} to 10^{-12}.

Runtimes

Figure 5.12 depicts runtimes of `zsytr1` (RI variant) and `zsytr2`. The higher accuracy of `zsytr1` comes for the price of lower runtimes, `zsytr2` is clearly faster. For example, for order $n = 2000$, `zsytr1` (time for constructing Q not included) takes about 212 seconds, while `zsytr2` takes only about 22 seconds, leading to a speedup of 9.7. `zhetrd` (LAPACK) has a similar runtime behavior as `zsytr2`, but is somewhat faster.

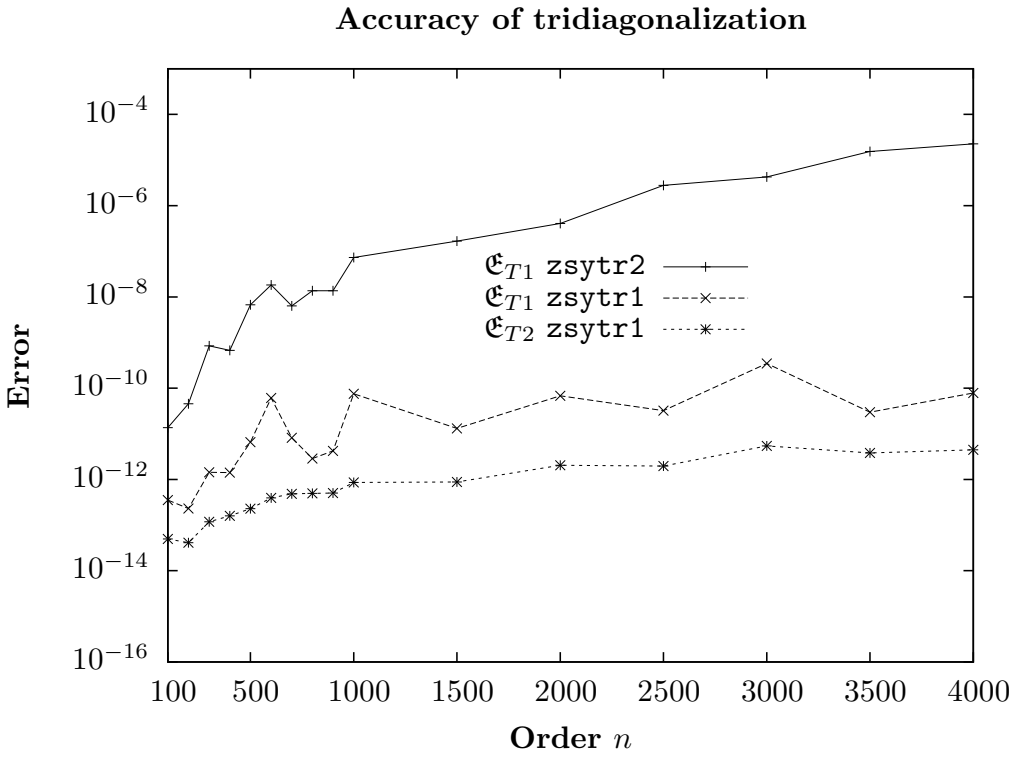

Figure 5.11: Accuracy of the complex symmetric tridiagonalization; $\mathfrak{E}_{T1} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$ denotes the maximum relative eigenvalue error and $\mathfrak{E}_{T2} := \frac{\|T - Q^\top M Q\|_2}{\|M\|_2}$ denominates the involved tridiagonalization error.

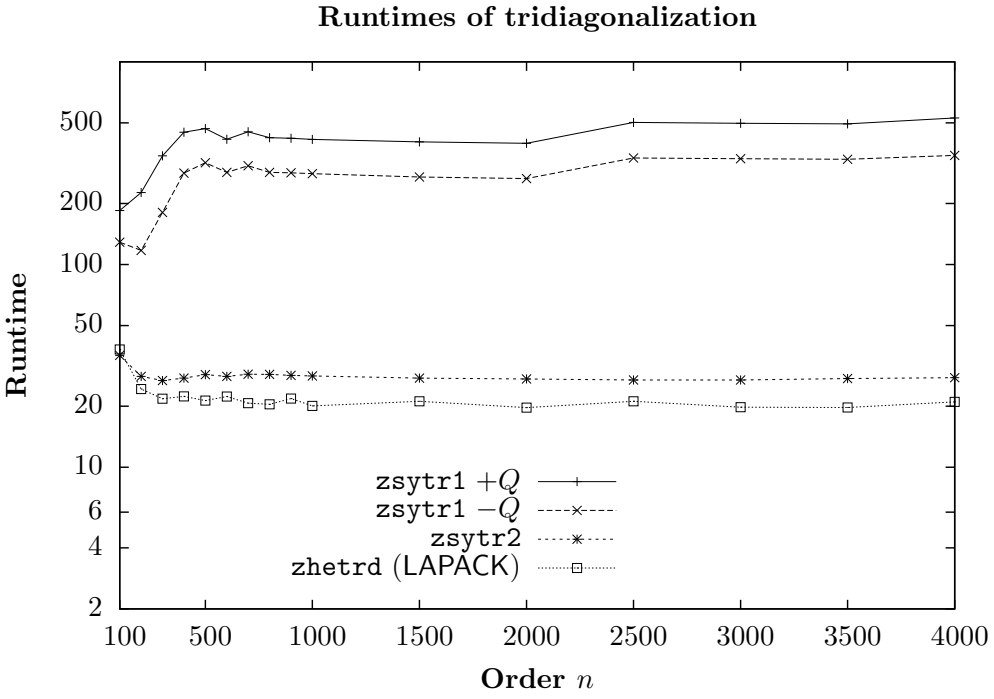

Figure 5.12: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of the complex symmetric tridiagonalization, where $+Q$ denotes the computation of a transformation matrix, and $-Q$ denotes that a transformation matrix is not constructed.

5.5 Solution of the Tridiagonal EVP

After tridiagonalizing, we solve the complex symmetric tridiagonal EVP $Tz = \lambda z$. If only eigenvalues are required, this step constitutes the final computational step. In the case that eigenvectors are additionally desired, see Section 5.6 for the backtransformation step.

5.5.1 Related Work

In analogy with dense symmetric EVPs, most tridiagonal solvers are not tailored for complex symmetric EVPs. See, for instance, the dissertation by I. Dhillon [Dhi97] aiming at real symmetric and Hermitian tridiagonal EVPs. In [OM92], the resulting complex symmetric tridiagonal EVP is solved using a modified QR algorithm [Wat08]. In a book by J. Cullum and R. Willoughby [CW02], eigenvalues of complex symmetric matrices are computed by applying a QL procedure [CW96, GD89], eigenvectors are computed utilizing inverse iteration [Ips97, PW79].

5.5.2 Methodology

Algorithm 5 depicts a high-level view of the computation of eigenvalues and eigenvectors to solve the tridiagonal EVP. In principle, dense solvers can also be used for tridiagonal EVPs, however, their runtimes may obviously be unnecessarily large. Here, the spectrum of the complex symmetric tridiagonal EVP is computed by applying a QL procedure, eigenvectors are computed utilizing inverse iteration.

Algorithm 5 Computing eigenpairs of the tridiagonal EVP

Require: Tridiagonal matrix T, $T \in \mathbb{C}^{n \times n}; T = T^\top$
1: Compute eigenvalues of T by applying a QL procedure
2: **for** $k \leftarrow 1$ to n **do**
3: Compute eigenvector k of T by applying inverse iteration
4: **end for**
Ensure: $Tz = \lambda z$
5: **return** eigenpairs of T

Implementation

Fortran routines `compev` and `inverm` were taken from the book [CW02] for implementing a QL procedure and inverse iteration, respectively; the same

approach was chosen in [BOR97]. These codes necessitated changes to allow proper compilation on modern compilers, removals of unused code lines (for example, print-statements), and the substitution of the hardcoded machine precision by a call to the corresponding LAPACK routine `dlamch`. On calling `inverm`, one has to specify the maximum number of iterations. Experiments revealed that the quality of the computed eigenvectors is not very sensitive to the specified value, 3 should mostly be a good choice.

A tridiagonal solver based on `EPSSolve` (SLEPc), actually designed for sparse EVPs, may be another option, but we have not evaluated it yet.

Accuracy

The quality of the tridiagonal solver is evaluated by computing the maximum relative eigenvalue error, based on a tridiagonal input matrix. $\mathfrak{E}_{TS} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where λ_k is an eigenvalue computed by `zgeevp`, and $\tilde{\lambda_k}$ is an eigenvalue computed by the evaluated routine `compev` or `zgeev` (LAPACK). `compev` and `zgeev` (LAPACK) do not necessarily return computed eigenvalues in the same order. In order to compare the matching pair of eigenvalues of the two sets of eigenvalues, the following algorithm is applied. In a loop over all eigenvalues, for each eigenvalue in set one, we assign the nearest unassigned eigenvalue of set two. This simple approach may not always deliver the optimal results, but proved satisfactory in practice.

A maximum residual error to evaluate eigenpairs is computed according to $\mathfrak{R}_{TS} := \max_k \frac{\|(T - \tilde{\lambda}_k I_n)\tilde{x}_k\|_2}{\|T\|_2}$. Here, $(\tilde{\lambda}_k, \tilde{x}_k)$ denominates an eigenpair computed by the evaluated routine `zstev` or `zgeev` (LAPACK). Figure 5.13 illustrates accuracies associated with the solution of complex symmetric tridiagonal EVPs.

The accuracy of the general solver `zgeev` (LAPACK) is very high and stable for different orders, both in terms of its residual error and its eigenvalue error. Our routine `zstev` is clearly less accurate, in terms of both applied metrics. Accuracy decreases with higher order.

Runtimes

The runtimes of our tridiagonal solver routine `zstev` is compared with the general dense solver routine `zgeev` (LAPACK), see Figure 5.14 for the results.

For all orders, `zstev` is clearly faster than `zgeev` (LAPACK).

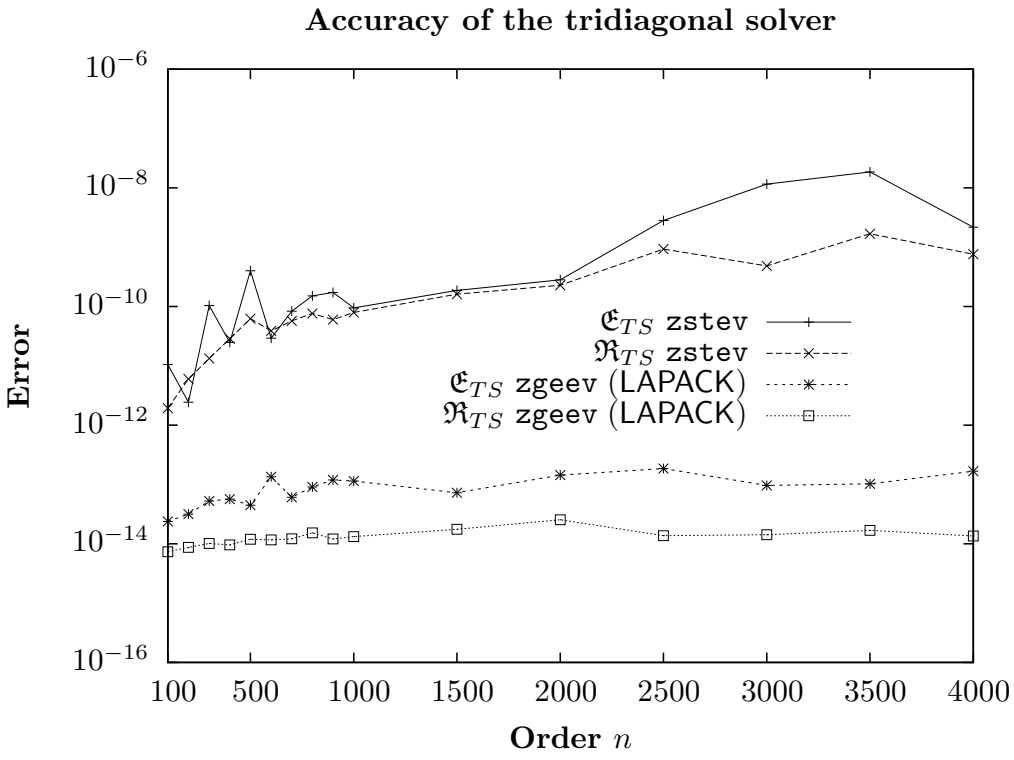

Figure 5.13: Accuracy of the complex symmetric tridiagonal solver `zstev` compared to the general dense standard solver `zgeev` (LAPACK); $\mathfrak{E}_{TS} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$ denotes the maximum relative eigenvalue error, $\mathfrak{R}_{TS} := \max_k \frac{\|(T - \tilde{\lambda}_k I_n)\tilde{z}_k\|_2}{\|T\|_2}$ denotes the maximum residual error.

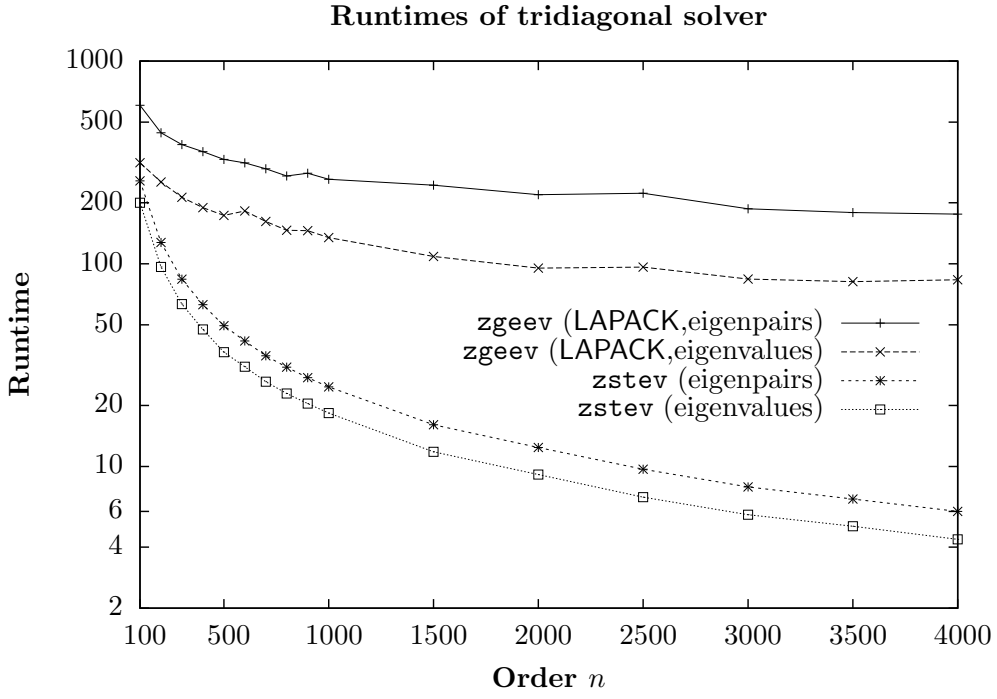

Figure 5.14: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of the complex symmetric tridiagonal solver `zstev` compared to the dense general solver `zgeev` (LAPACK).

5.6 Backtransformation

Backtransformation incorporates the "transforming back" (a) of eigenvectors from the tridiagonal to the standard EVP and (b) of eigenvectors from the standard to the generalized EVP.

5.6.1 Related Work

We have not found any papers especially dedicated to the backtransformation process for computing eigenvectors. Backtransformation is thematically very closely connected to the computation of eigenvectors, and it is also determined by it.

5.6.2 Methodology

The backtransformation of eigenvectors from tridiagonal to standard EVP is realized by the general matrix-matrix multiplication $Y = Q^\top Z$, where Y contains the eigenvectors of the standard EVP, Q is the transformation matrix, and Z contains the eigenvectors of the tridiagonal EVP. This is one possible method, see Section 5.4.2 for a further discussion of this backtransformation process.

The backtransformation of eigenvectors from standard to generalized EVP corresponds to the solution of the equation $L^\top X = Y$, where L was computed when factorizing $B \mapsto LL^\top$; X corresponds to the eigenvectors of the generalized EVP and Y corresponds to the eigenvectors of the standard EVP. L is triangular and Y does not feature any special structure.

Implementation

As no special structure of $Q^\top$ or Z can be observed, the general matrix-matrix multiplication for complex matrices `zgemm` (BLAS) is used to backtransform eigenvectors from tridiagonal to standard EVP.

The backtransformation of eigenvectors from standard to generalized EVP is implemented by a call to the solver routine `ztrtrs` (LAPACK) solving equations of the type $A^\top X = B$; A is triangular, X constitutes the solution matrix.

Accuracy

There is no expedient way to evaluate the accuracy of backtransformation, therefore we can not give any results here.

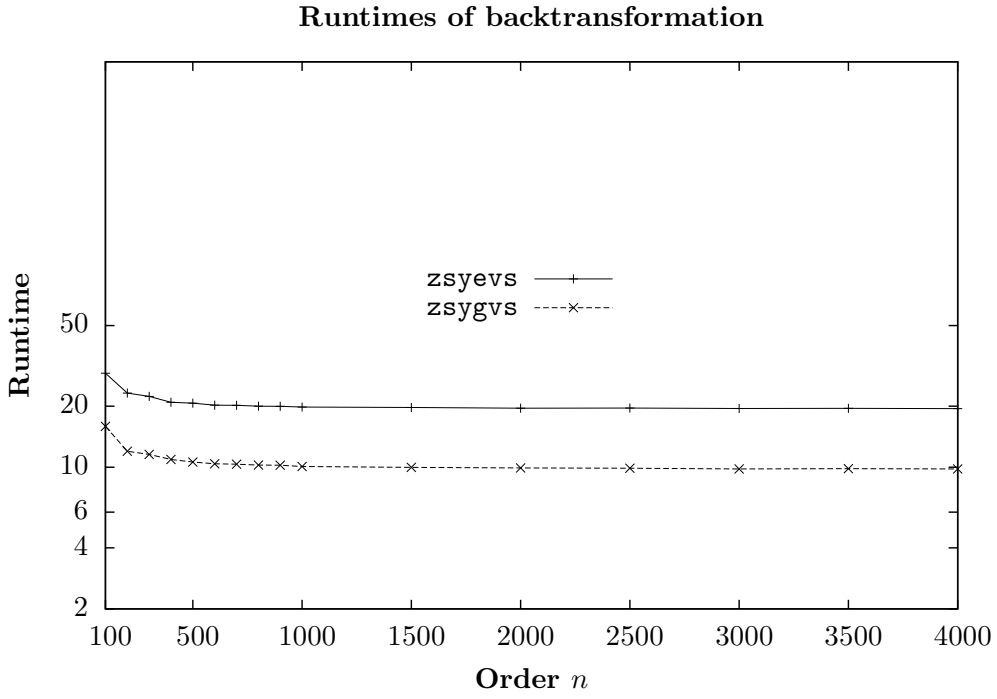

Figure 5.15: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of the backtransformation in the standard solver `zsyevs` (calling `zgemm` (BLAS)) and the generalized solver `zsygvs` (calling `ztrtrs` (LAPACK)).

Runtimes

Runtimes of backtransformation have been evaluated, see Figure 5.15 for the results.

Backtransformation in `zsyevs` consumes considerably more time than the backtransformation in `zsygvs`. We note that the backtransformation in `zsyevs` has some potential for improvements. The observed differences are unsurprising, as different computational methods were applied.

5.7 Combined Evaluations

This section combines some of the analyzed components of previous sections to evaluate more practical aspects of implemented codes. Combined evaluations include the transformation from generalized to standard EVP, standard EVP solvers, and generalized EVP solvers.

5.7.1 Transformation to Standard EVP

This section discusses the transformation from the generalized to the standard EVP. It combines the symmetric indefinite factorization in `zpotrfi` and the actual reduction in `zsygst`.

Accuracy

A maximum relative eigenvalue error is computed according to $\mathfrak{E}_{FR} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where eigenvalues of the computed standard EVP are compared to the eigenvalues of the generalized EVP computed by `zggev` (LAPACK). Figure 5.16 depicts this accuracy. The measured maximum relative eigenvalue error is substantial for small orders n and growing fast with higher n, therefore we identify this step as one of two major sources (the other one being tridiagonalization) of error for the full generalized solver. The magnitude of this error is higher than expected, one reason for this observation may be determined by high condition numbers in the factorization step. See the generalized solver in Section 5.7.3 for a further discussion.

Runtimes

Normalized runtimes are compared with the Hermitian pendant and depicted in Figure 5.17. We observe that due to the slower reduction, the whole process is slower than for Hermitian EVPs. Once the reduction operation has been optimized, similar runtimes as in the LAPACK procedure for Hermitian problems are expected.

5.7.2 Standard EVP

We evaluate our two standard EVP solvers `zsyevn` (featuring a non-splitting tridiagonalization) and `zsyevs` (featuring a splitting tridiagonalization). This process comprises the steps complex symmetric tridiagonalization in `zsytr1` or `zsytr2` and the solution of the tridiagonal EVP in `zstev`.

Accuracy

A maximum relative eigenvalue error is determined according to $\mathfrak{E}_S := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where the eigenvalue of the evaluated standard solver is compared with `zgeev` (LAPACK). A maximum residual error is computed according to $\mathfrak{R}_S := \max_k \frac{\|(M - \tilde{\lambda}_k I_n)\tilde{y}_k\|_2}{\|A\|_2}$ and compared with the residuals of the eigenpairs of `zgeev` (LAPACK). See Figure 5.18 for measured accuracies. We

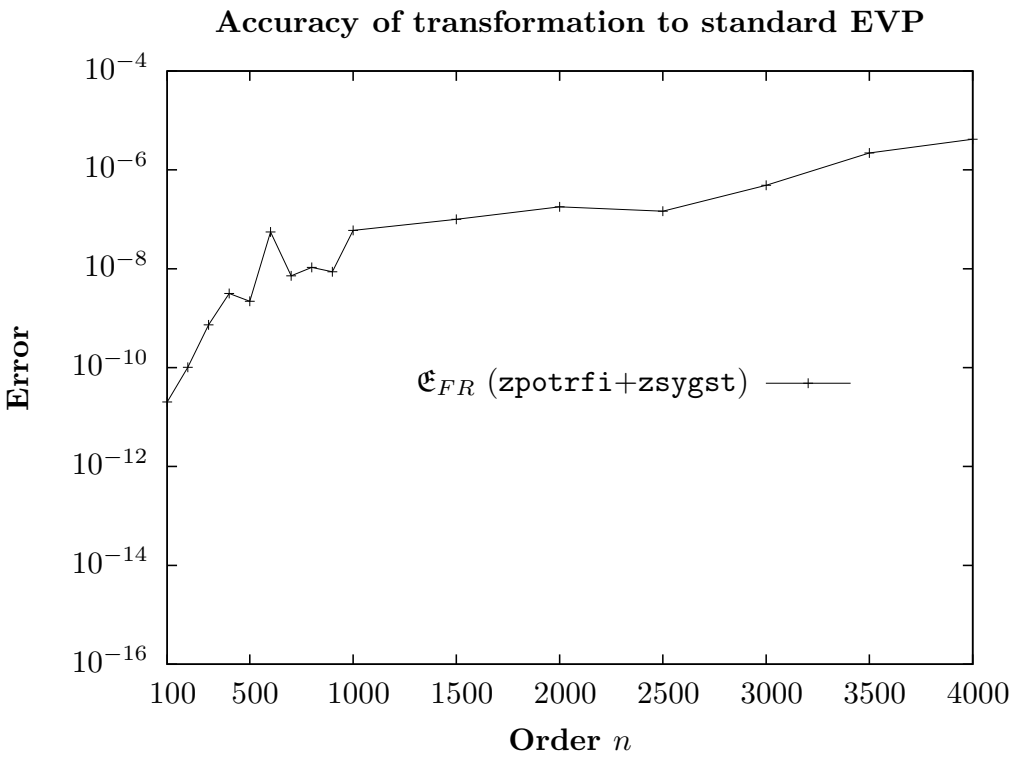

Figure 5.16: Accuracy of full reduction; a maximum relative eigenvalue error is computed as $\mathfrak{E}_{FR} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, where eigenvalues of the reduced problem (corresponding to the sequence of calls to `zpotrfi` and `zsygst`) are compared to the eigenvalues of the original problem computed by `zggev` (LAPACK); $\tilde{\lambda}_k$ and λ_k are the eigenvalues of the computed standard EVP and of the generalized EVP, respectively.

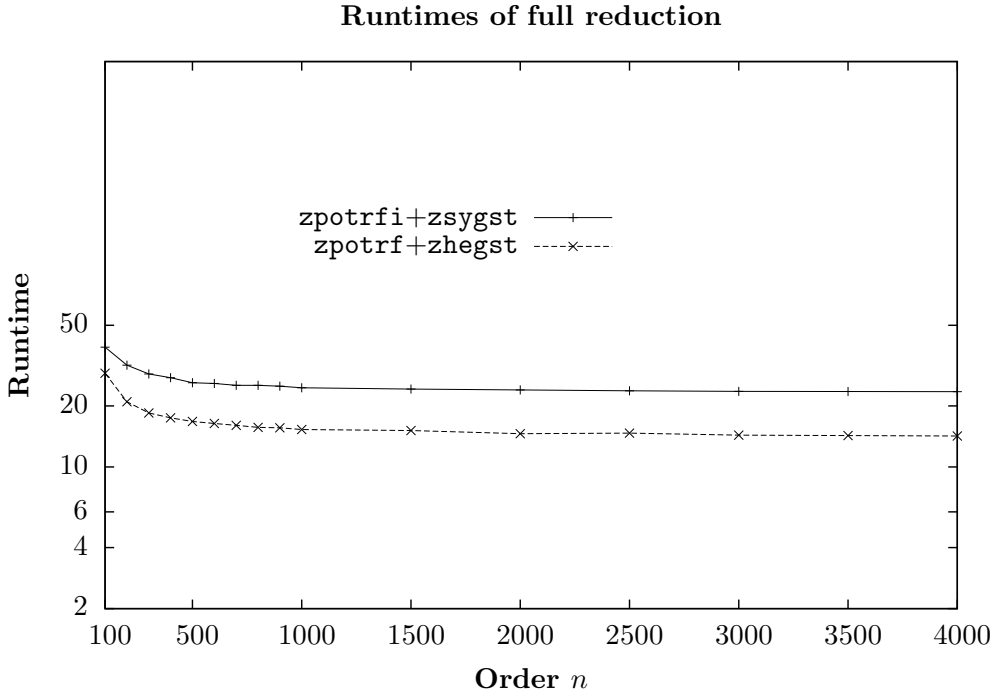

Figure 5.17: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of the transformation from generalized to standard EVP calling `zpotrfi` followed by a call to `zsygst`, in comparison with the procedure for Hermitian problems.

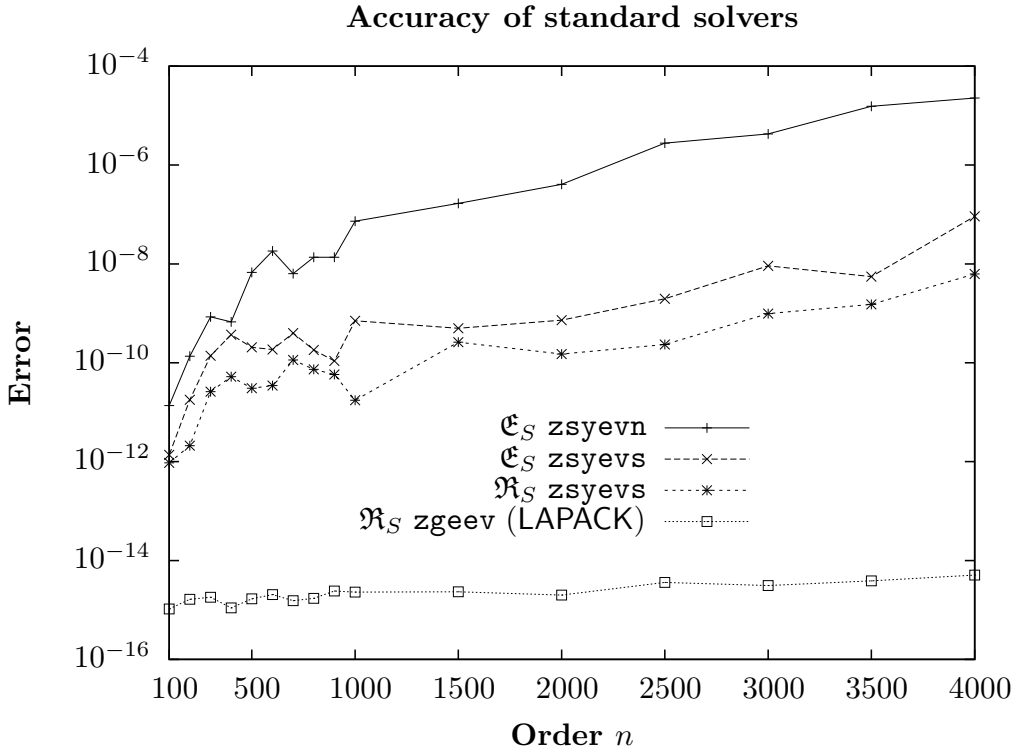

Figure 5.18: Accuracy of standard solvers, evaluated by a maximum relative eigenvalue error $\mathfrak{E}_S := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$ and a maximum residual error $\mathfrak{R}_S := \max_k \frac{\|(M - \tilde{\lambda}_k I_n)\tilde{y}_k\|_2}{\|M\|_2}$; λ_k denotes an eigenvalue of `zgeev` (LAPACK), $\tilde{\lambda}_k$ and $\tilde{y}_k$ denote eigenvalue and eigenvector of the evaluated routine, respectively.

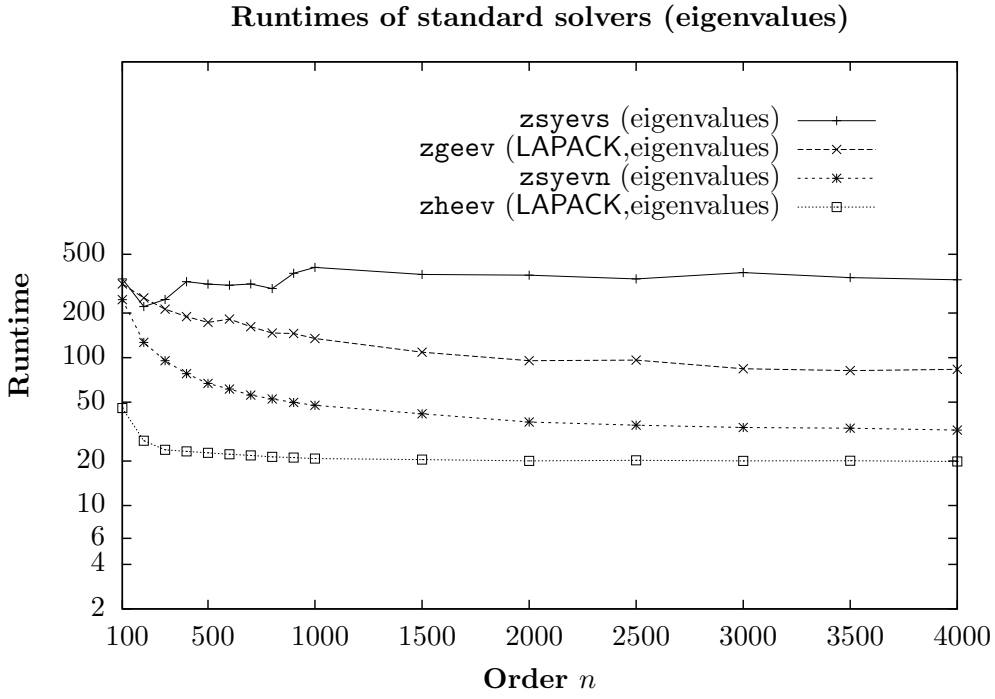

Figure 5.19: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of different variants of standard solvers for computing eigenvalues.

observe accuracies of `zsyevs` starting about three orders of magnitude lower than `zgeev` (LAPACK). While accuracy of the LAPACK routine remains constant with higher orders n, we observe decreasing accuracies for `zsyevn` and `zsyevs`. `zsyevn` features an eigenvalue error about one order of magnitude higher than `zsyevs`. The two curves representing eigenvalue errors of `zsyevn` and `zsyevs` feature similar characteristics as those for tridiagonalization in Figure 5.11. Consequently, the tridiagonalization process can be identified as a major source for the standard solvers. These results are similar to earlier evaluations in [GGP09], where partly different metrics were applied.

Runtimes

Runtimes of `zsyevn` and `zsyevs` are depicted in Figure 5.19 (eigenvalues) and Figure 5.20 (eigenpairs). Accordingly, `zsyevs` is slower than its competitor routine `zgeev` (LAPACK). This observation affects both the computation of eigenvalues and the computation of eigenvectors. Furthermore, the computation of eigenpairs in `zgeev` (LAPACK) is faster then the computation of eigenvalues only in `zsyevs`. This behavior can most probably be attributed

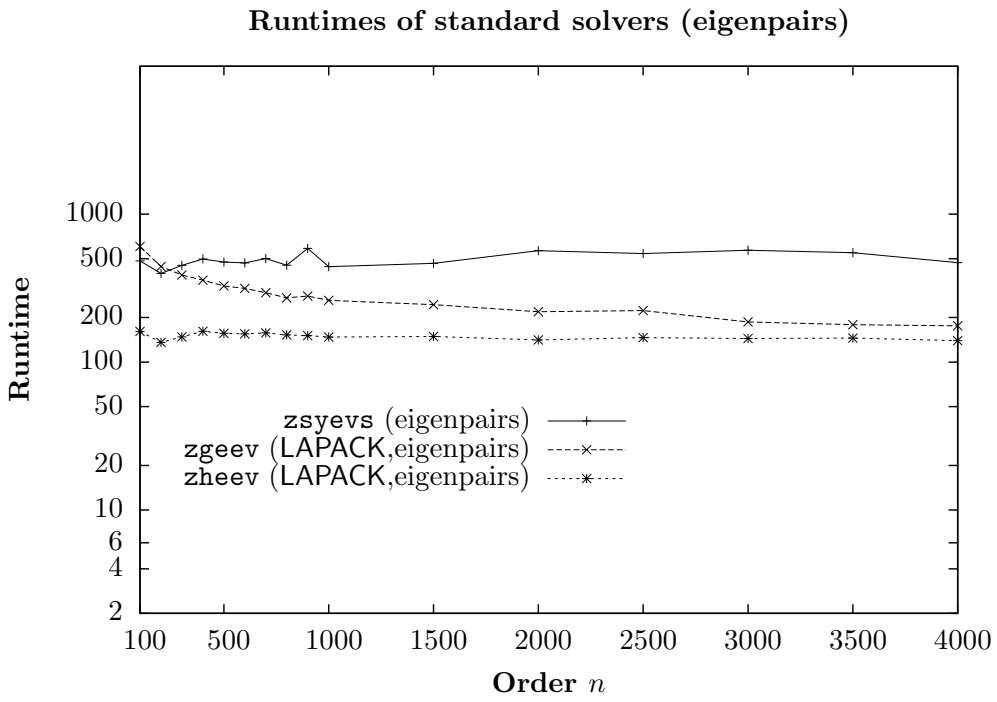

Figure 5.20: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of different variants of standard solvers for computing eigenpairs.

to the access patterns of the approaches. While `zgeev` (LAPACK) operates mostly on blocks of matrices and therefore takes advantage of level-3 BLAS operations, `zsyevs` operates on single rows and single columns and therefore abstains from calling the fast level-3 BLAS operations. "Upgrading" from level-2 to level-3 operations is not trivial and incorporates major redesign of the whole tridiagonalization process.

`zsyevn` computes eigenvalues considerably faster than `zsyevs` and `zgeev` (LAPACK), but slower than its Hermitian pendant `zheev` (LAPACK). It can be anticipated that a blocked version of the splitting tridiagonalization process would considerably increase the runtime performance of `zsyevs`. Once backtransformation is implemented in `zsyevn`, similar runtimes as for `zheev` (LAPACK) are expected.

5.7.3 Generalized EVP

We evaluate our generalized complex symmetric EVP solvers `zsygvn` (featuring a non-splitting tridiagonalization) and `zsygvs` (featuring a splitting tridiagonalization).

In previous accuracy evaluations of this work, we have been applying purely randomly generated matrices (here called RND, see Section 5.2.7). For reasons explained later, we additionally evaluate generalized solvers with pairs of "special" matrices called SP.

Special matrices

Accuracy tests with large random pairs of matrices cause the following difficulty: while accuracy of the solver decreases, it becomes more difficult to assign the right pairs of eigenvalues of the tested routine and the testing routine to each other. For huge problems, it sometimes happens that "wrong" eigenvalues are compared. This is the reason, why two lines in Figure 5.21 are incomplete. In order to avoid this pitfall, we constructed pairs of matrices SP, where the eigenvalues are known in advance, thus it is not needed to compare the desired eigenvalues with those computed by a reference routine.

For the testmatrices of type SP, matrix B is created randomly as for type RND, but matrix A is constructed such that the generalized EVP (A, B) has a given spectrum: we start with two random complex symmetric matrices, B and Z. Using `zgeev` (LAPACK) we compute the matrix $X = (x_1, x_2, \ldots, x_n)$ of right eigenvectors of Z. After scaling each eigenvector, $x_k \mapsto x_k(x_k^\top x_k)^{-0.5}$, X is a complex orthogonal matrix, hence $XX^\top = X^\top X = I$. Then we construct a diagonal matrix Λ with prescribed eigenvalues $\hat{\lambda}_k = k + k(-1)^{k+1}i$. We deliberately choose the eigenvalues with a big distance between two neigh-

bors. Then, B is factorized $B \mapsto LL^\top$. Condition numbers of randomly generated triangular matrices tend to exponentially grow, while those of random dense matrices just grow linearly [VT98]. This is not the case for randomly generated full complex symmetric matrices. Therefore, we start with generating a full random complex symmetric matrix B. Finally, the complex symmetric matrix $A := LX\Lambda X^\top L^\top$ is constructed. The generalized EVP (A, B) then has the prescribed $\hat{\lambda}_k$ as eigenvalues.

Accuracy

Denoting the eigenvalues computed by `zggev` (LAPACK) with λ_k and the eigenvalues computed by `zsygvn` or `zsygvs` with $\tilde{\lambda}_k$, the maximum relative eigenvalue error has been computed according to $\mathfrak{E} := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, respectively. Figure 5.21 depicts the results. Accordingly, computed eigenvalues for matrix type RND feature a lower accuracy than eigenvalues for matrix type SP. `zsygvs` achieves a better accuracy in eigenvalues than `zsygvn`. For orders n larger than a few hundreds, accuracy of `zsygvn` and of `zsygvs` for problems of type RND is insufficient: it happens that for increasing eigenvalue errors, it becomes harder to find pairs of matching eigenvalues for comparison, therefore the corresponding two lines for `zsygvn` (type RND) and for `zsygvs` (type RND) are incomplete. Problems of type SP appear better conditioned than problems of type RND (which was the initial intention of generating matrices of type SP for the previously "best" routine `zggev` (LAPACK) and "worst" routine `zsygvn`), and the evaluation for $n = 100$ to 4000 is unproblematic.

A residual error to evaluate eigenpairs is computed according to $\mathfrak{R}_G := \max_k \frac{\|(A-\tilde{\lambda}_k B)\tilde{x}_k\|_2}{\|A\|_2\|B\|_2}$. Corresponding evaluations show a different picture. In Figure 5.22, we can see that `zggev` (LAPACK) features very high accuracy remaining roughly stable over all evaluated orders. For small problems, `zsygvs` starts with a reduced accuracy of about three orders of magnitude. The gap in accuracy between the two routines increases to about 5 orders of magnitude, becoming roughly stable on huge orders. With a maximum residual error of below 10^{-10} for order $n = 4000$, this residual error is at an acceptable level. `zsygvn` is currently unable to backtransform eigenvectors, therefore residuals can not be evaluated.

Runtimes

Runtimes of `zsygvn` and `zsygvs` are depicted along with their competitor routine `zggev` (LAPACK) in Figure 5.23. The performances of `zsygvn` and `zsygvs` are very encouraging. The fastest variant for computing eigenvalues is `zsygvn`, then comes `zsygvs`, followed by `zggev` (LAPACK); the fastest vari-

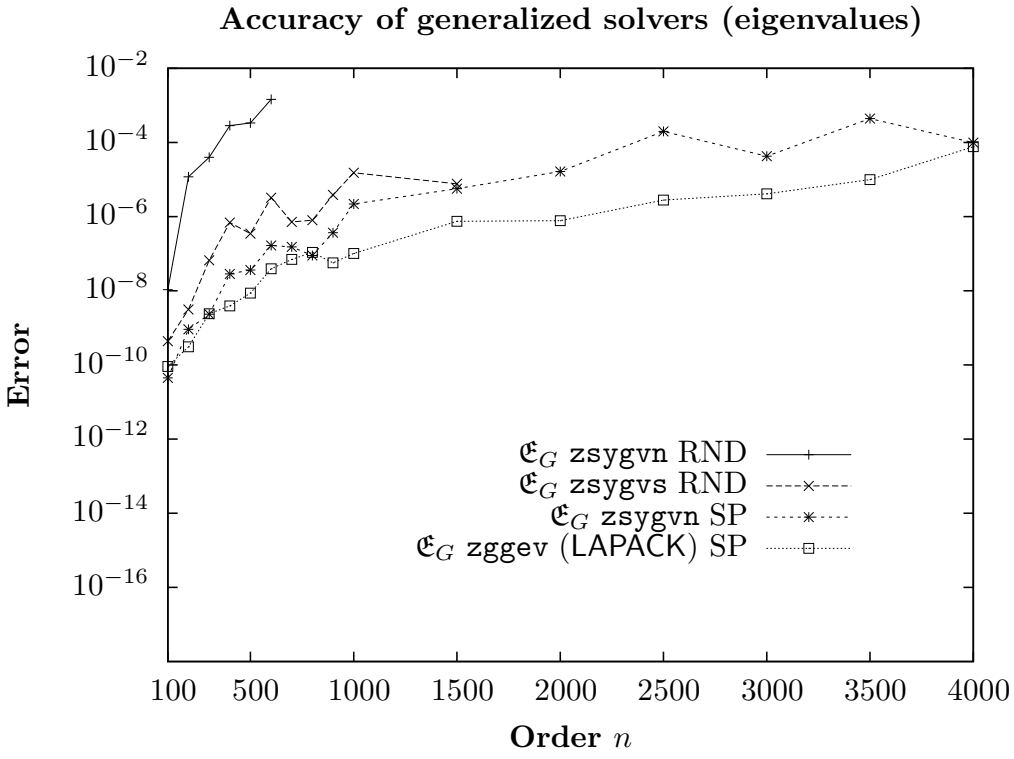

Figure 5.21: Eigenvalue error of the generalized solver `zsygvs` (performing a splitting tridiagonalization) and `zsygvn` (performing a non-splitting tridiagonalization) evaluated by a maximum relative eigenvalue error $\mathfrak{E}_G := \max_k \frac{|\tilde{\lambda}_k - \lambda_k|}{|\lambda_k|}$, compared with `zggev` (LAPACK). RND denominates a pair of randomly generated matrices, SP denominates a pair of special matrices.

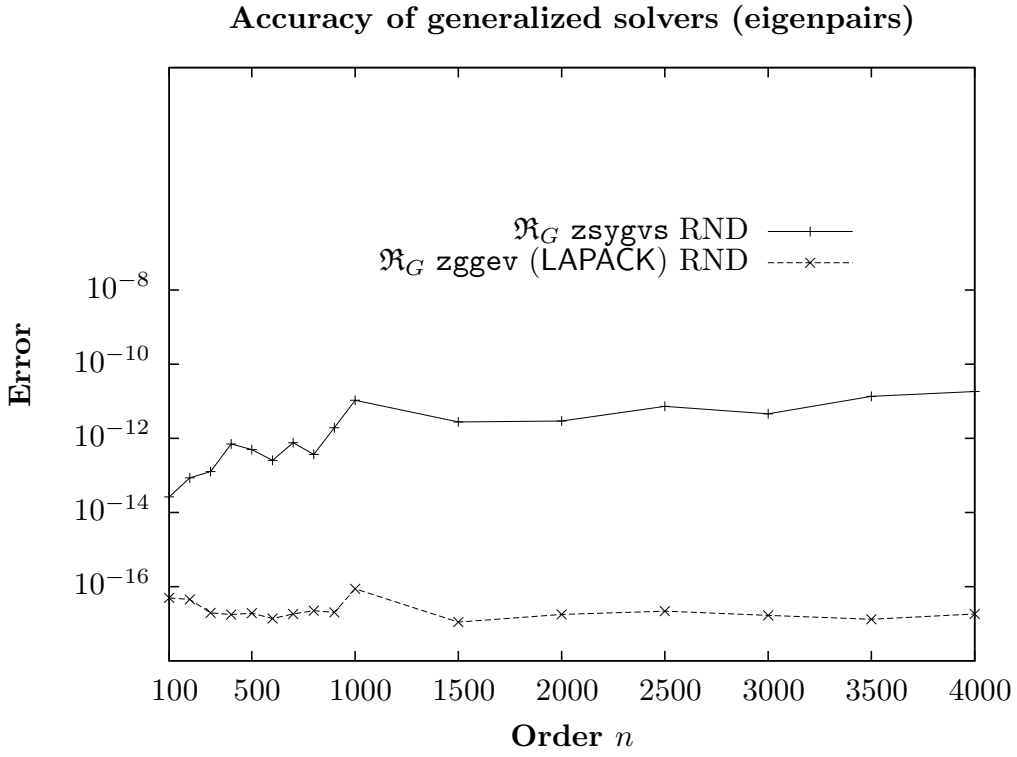

Figure 5.22: Residual error of the generalized solver `zsygvs` (performing a splitting tridiagonalization) evaluated by a maximum residual error $\mathfrak{R}_G := \max_k \frac{\|(A-\tilde{\lambda}_k B)\tilde{x}_k\|_2}{\|A\|_2\|B\|_2}$ in comparison with `zggev` (LAPACK). RND denominates a pair of randomly generated matrices.

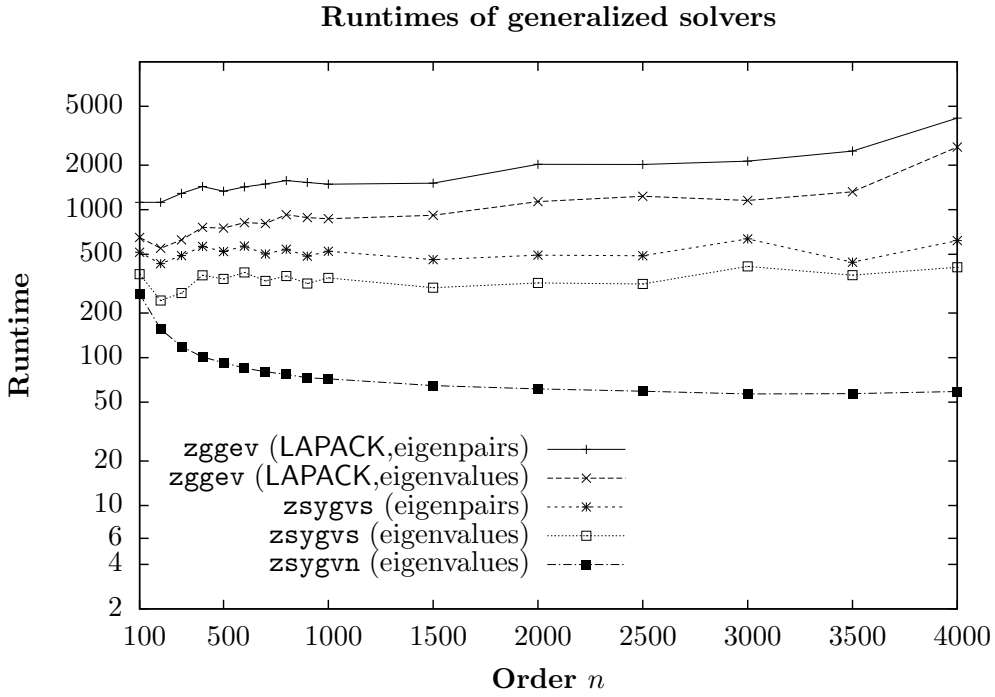

Figure 5.23: Normalized (log scale) runtimes $T(n)\frac{10^{10}}{n^3}$ of new generalized solvers `zsygvs` (performing a splitting tridiagonalization) and `zsygvn` (performing a non-splitting tridiagonalization) compared to the dense general solver `zggev` (LAPACK).

ant for computing eigenpairs is `zsygvs`, `zggev` (LAPACK) is slower; `zsygvs` computes eigenpairs faster than `zggev` (LAPACK) computes eigenvalues only.

Moreover, we evaluate speedups of `zsyevn` and `zsyevs` versus `zggev` (LAPACK). A speedup S is hereby computed as $S = \frac{T_1}{T_2}$, where T_1 denotes the time consumed for the solution of the problem applying routine `zggev` (LAPACK), and T_2 denotes the time consumed for the solution of the same problem applying the evaluated routine `zsygvn` or `zsygvs`. Figure 5.24 shows this speedup curve. For computing eigenvalues with `zsygvn`, we observe an increasing speedup with increasing problem sizes, reaching a value of 43 for larger problem sizes. This increase of the speedup S with the matrix order is due to the fact that `zsygvn` has a lower asymptotic complexity than `zggev` (LAPACK), operating with a tridiagonal matrix in the final phase instead of a Hessenberg matrix since symmetry is preserved. `zsygvn` is currently restricted to computing eigenvalues, therefore we can not evaluate

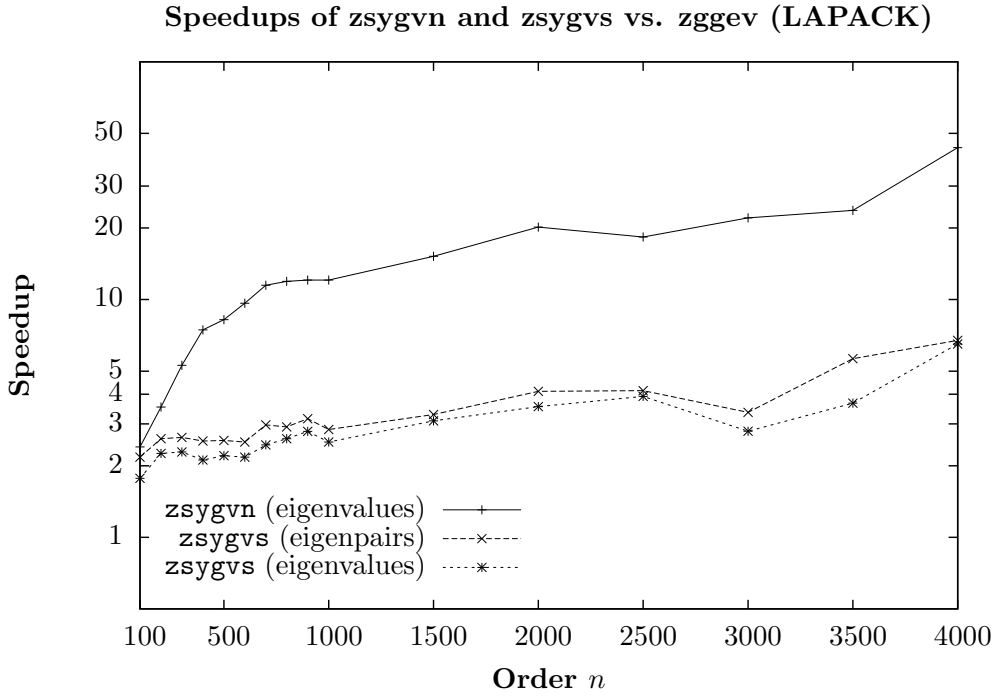

Figure 5.24: Speedups of `zsygvn` and `zsygvs` over `zggev` (LAPACK); speedup $S = \frac{T_1}{T_2}$, where T_1 denotes the time consumed for the solution of the problem applying routine `zggev` (LAPACK), and T_2 denotes the time consumed for the solution of the same problem applying the evaluated routine `zsygvn` or `zsygvs`.

speedups computing eigenpairs. For computing eigenvalues and eigenpairs with `zsygvs`, we observe speedups from about 2 to about 6, respectively, where speedups for eigenpairs are slightly better than for eigenvalues only.

Shares of runtimes

Figure 5.25 depicts shares of runtimes of `zsygvn`, computing eigenvalues only. For order $n = 1000$, we observe that factorization consumes the least time, followed by the tridiagonal solver, reduction, and tridiagonalization being the slowest part. For higher orders, tridiagonalization remains the dominant part, but factorization increases its fraction on overall-runtime. The corresponding percentages for parts factorization / reduction / tridiagonalization / tridiagonal solver are as follows; for order $n = 1000$, we observe rounded percentages of 5/29/40/26, for $n = 2000$ 6/34/45/15, for $n = 3000$ 7/35/48/10, and for $n = 4000$ 7/36/50/8.

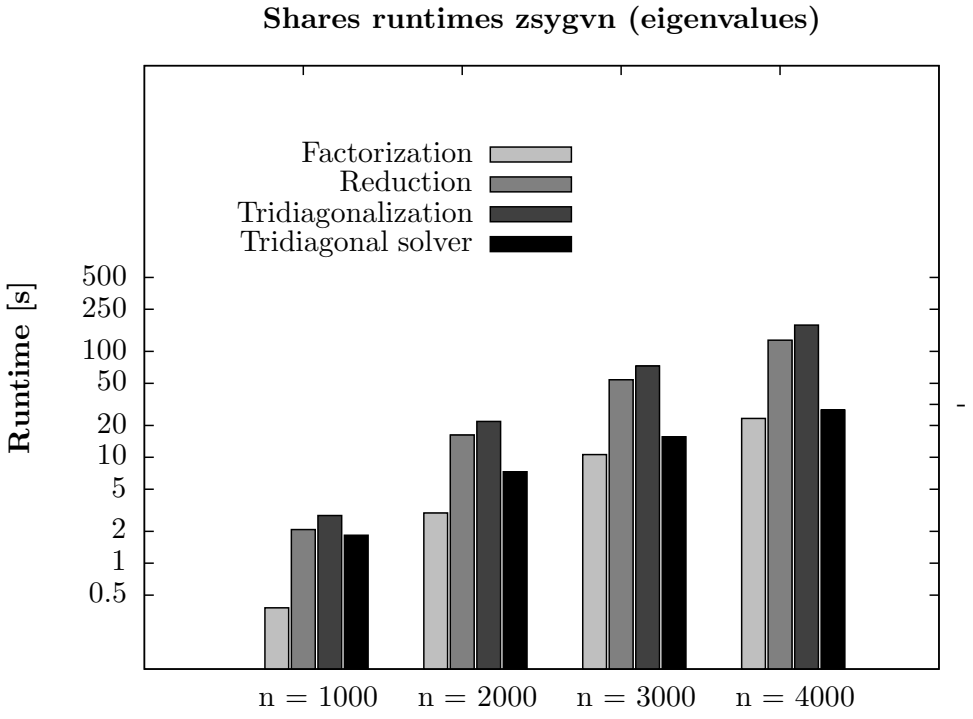

Figure 5.25: Shares (log scale) of runtimes for cardinal steps of routine `zsygvn`, computing eigenvalues of a generalized complex symmetric EVP.

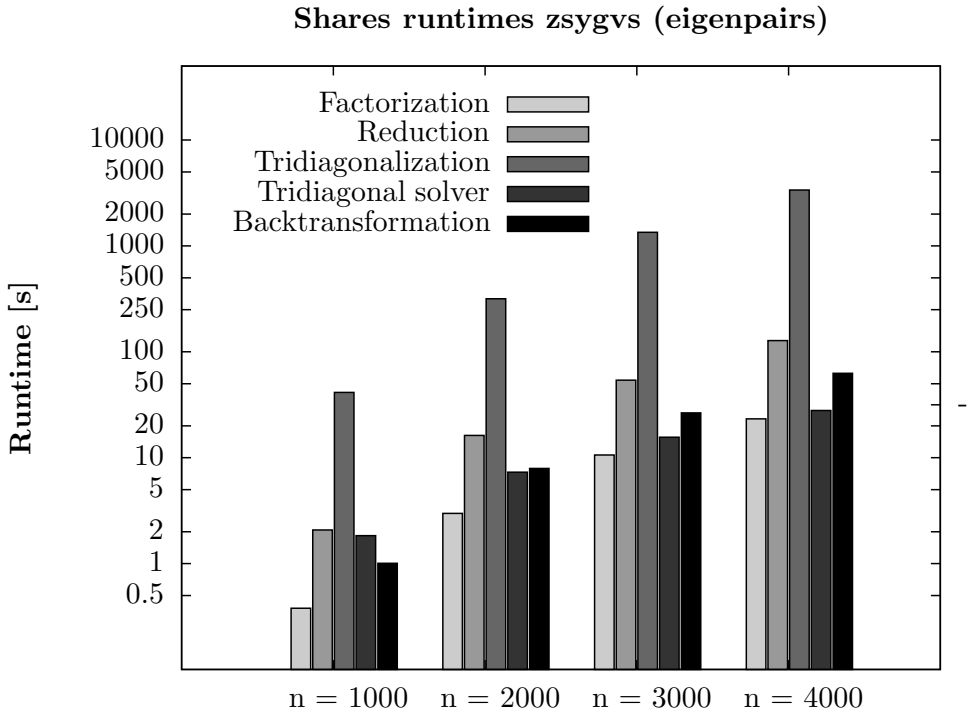

Figure 5.26: Shares (log scale) of runtimes for cardinal steps of routine `zsygvs`, computing eigenpairs of a generalized complex symmetric EVP.

Figure 5.26 depicts shares of runtimes of `zsygvs`, computing eigenpairs. For $n = 1000$, we observe that factorization consumes the smallest part of overall runtime, followed by backtransformation, tridiagonal solver, reduction, and factorization consuming the biggest share. For higher orders, the factorization consumes a considerably bigger share, while tridiagonalization remains the dominating part for all tested orders. The corresponding percentages for parts factorization / reduction / tridiagonalization / tridiagonal solver / backtransformation are as follows; for order $n = 1000$, we observe rounded percentages of 1/4/89/4/2, for $n = 2000$ 1/5/90/2/2, for $n = 3000$ 1/4/93/1/2, and for $n = 4000$ 1/4/93/1/2.

A comparison between the shares of `zsygvn` and `zsygvs` reveals that for all evaluated orders tridiagonalization consumes a notably larger fraction in `zsygvs`. This evidences that an optimization of the tridiagonalization process would substantially increase the overall performance of `zsygvs`.

Chapter 6

Parallel Case Studies

Due to the advent of various parallel architectures in scientific computing (confer Chapter 2), implementations utilizing parallel infrastructures have become very important. As a consequence of the involvement of multiple computing elements in order to reach a common goal, parallel computations are always more complex than serial ones. In scientific applications, parallel computing usually involves the cardinal steps (*a*) computation, (*b*) communication, and (*c*) synchronization [Bis04]. (*b*) incorporates the sending and receiving of data items to processes, which use the data for subsequent computations (confer Definitions 6.1.1 and 6.1.2).

A superstep contains either a number of computation steps or a number of communication steps, followed by a global barrier synchronization [Bis04, p.3]. See Figure 6.1 for an illustration of the schedule model of an abstract algorithm featuring five supersteps. The current chapter features fine-grained solver components aiming at solving complex symmetric generalized EVPs and a coarse-grained case study realized by a grid workflow in phylogenetics.

6.1 Parallel High Performance Programming

As sequential high performance programming is a difficult task, the more complex parallel high performance programming is even harder. In the case of parallel environments, narrowing the gap between the theoretical peak performance and the practical sustained performance is a highly complex task, see for example [SSB+08, GDS+06].

ScaLAPACK is a collection of high-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM or MPI [BCC+97, p.3]. It is a very effective

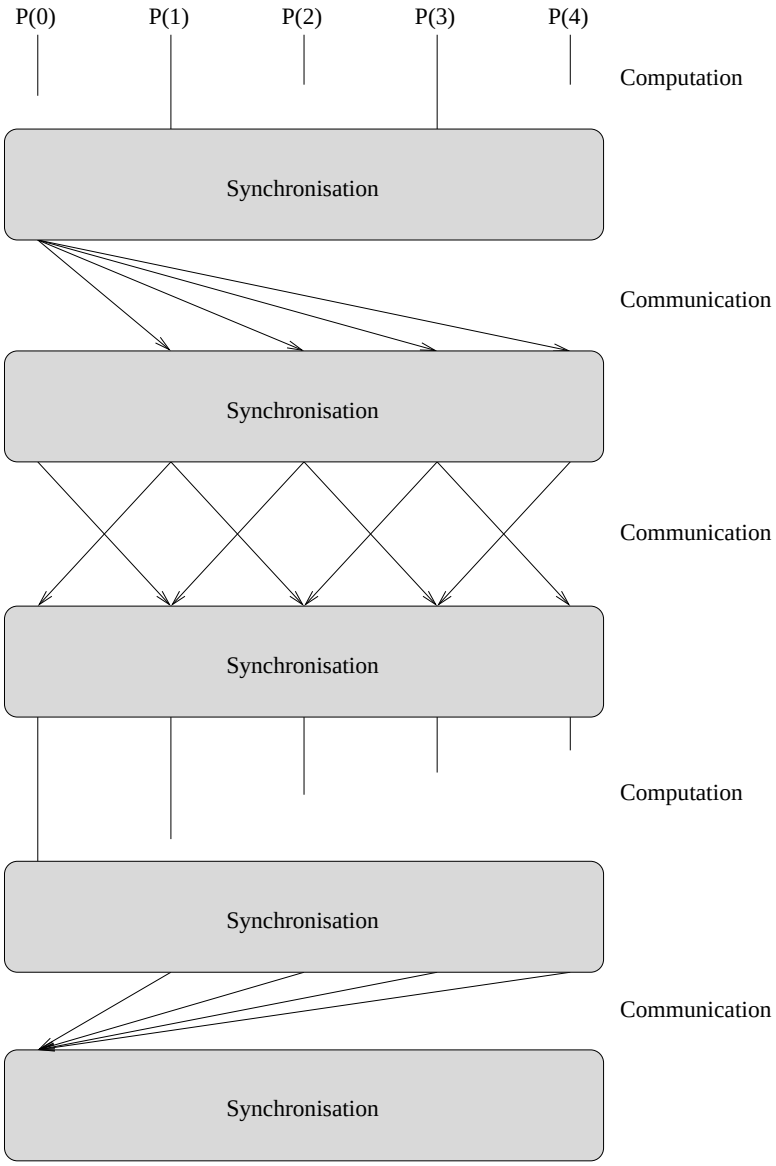

Figure 6.1: Schedule model of an abstract algorithm involving five processes P(0) ... P(4), featuring computation, communication, and synchronization in a sequence of five supersteps; a vertical line denotes local computation, an arrow denotes communication between processes. Each superstep is terminated by a global synchronization; taken from [Bis04, p.4].

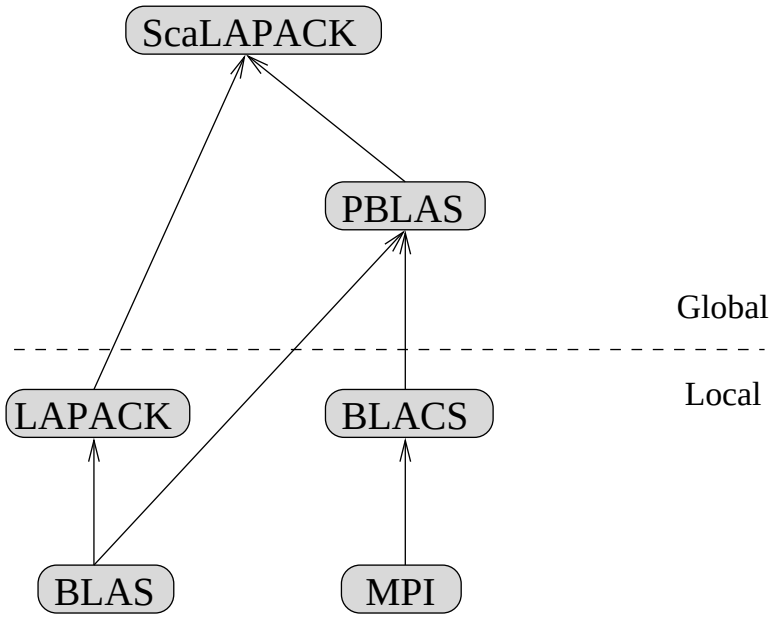

Figure 6.2: Scalable LAPACK (ScaLAPACK) is based on Basic Linear Algebra Subprograms (BLAS), Linear Algebra Package (LAPACK), Message Passing Interface (MPI), Basic Linear Algebra Communication Subprograms (BLACS), and Parallel BLAS (PBLAS); illustration taken from [BCC+97, p.5].

and successful linear algebra package for parallel systems. The usual way of deploying ScaLAPACK is by compiling from freely available source files; moreover, precompiled binaries are available for a selection of computer systems, including Cray T3E, Intel Paragon, IBM SP-2, SGI Power Challenge Array, SGI Origin 2000, AIX46K, DEC Alpha, HP 9000, Intel/Linux, SGI64, SUN4, and SUN4SOL2[1]. Figure 6.2 lists ScaLAPACK's basic building blocks.

As communication on parallel systems is usually an expensive operation, the key aspects for efficient data-intensive parallelization of software are data locality (see, for example, [WL91]) and the data distribution among computing processes. The following definitions are crucial in parallel high performance programming, they follow the ScaLAPACK Users' Guide [BCC+97, p.213-216].

Definition 6.1.1 (Process)**.** Basic unit or thread of execution that minimally includes a stack, registers, and memory. Multiple processes may share a physical processor. The term processor refers to the actual hardware.

[1]This list of supported computer systems is taken from `http://netlib.org/scalapack/archives/`, names of individual systems of vendors may slightly vary.

Definition 6.1.2 (Data distribution)**.** Method by which the entries of a global matrix are allocated among the processes, also commonly referred to as decomposition or data layout.

Definition 6.1.3 (Process grid)**.** The way we logically view a parallel machine as a one- or two-dimensional rectangular grid of processes. For two-dimensional process grids, the variable P_r is used to indicate the number of rows in the process grid (first dimension of the two-dimensional process grid). The variable P_c is used to indicate the number of columns in the process grid (the second dimension of the two-dimensional process grid). The collection of processes need not physically be connected in the two-dimensional process grid.

Definition 6.1.4 (Block size of the distribution)**.** The number of contiguous rows or columns of a global matrix to be distributed consecutively to each of the processes in the process grid. The block size is quantified by the notation $MB \times NB$, where MB is the row block size and NB is the column block size. The distribution block size can be square, $MB = NB$, or rectangular, $MB \neq NB$. Block size is also referred to as the partitioning unit or blocking factor.

Array Distributions

We are dealing with two-dimensional arrays, their different ways of mapping pieces of data to processes are described here. Data distribution schemes of section 6.1 are taken from the ScaLAPACK Users' Guide. Since numerically intensive algorithms in numerical linear algebra usually operate on big amounts of distributed data, efficient algorithms depend on particularly suitable data distributions. In the following, processes are enumerated from 0 to $P - 1$, while the columns k of the matrix are enumerated from 1 to N.

One-dimensional block column distribution

This distribution (see Figure 6.3, (*left*)) assigns a block of contiguous columns to successive processes, each process receives precisely one block of columns of the original matrix. tc represents the maximum number of columns stored per process, it is calculated according to $tc = \lceil \frac{N}{P} \rceil$. Column k is owned by process $\lfloor \frac{k}{tc} \rfloor$. For example, let a matrix consist of 25 columns, and let there be four processes. $tc = \lceil \frac{25}{4} \rceil = 7$. The matrix will be distributed as seven columns on process 0, seven columns on process 1, seven columns on process 2, and finally four columns on process 3.

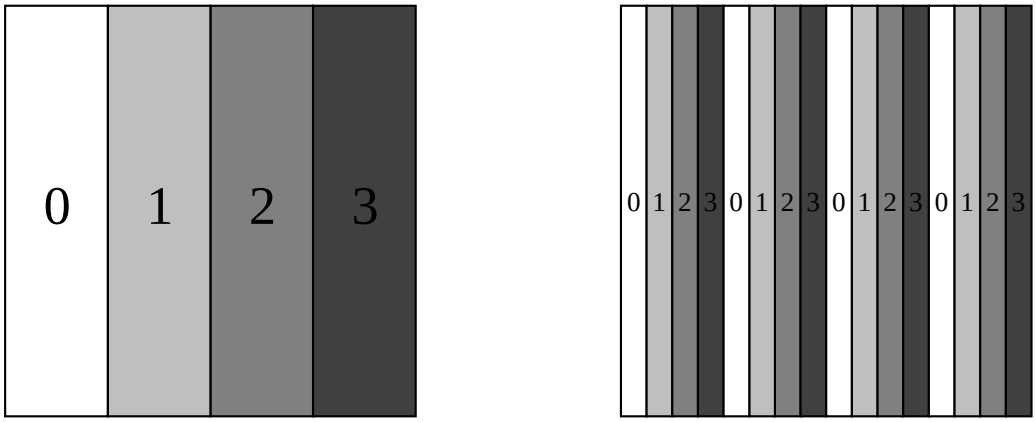

Figure 6.3: The one-dimensional block (*left*) and cyclic (*right*) column data distributions operating in four processes, corresponding to [BCC+97, p.59]. Levels of gray represent processes 0 to 3.

One-dimensional cyclic column distribution

The one-dimensional cyclic column distribution assigns single columns to the available processes, see Figure 6.3. Column k is assigned to process $(k-1)$ mod P.

One-dimensional block-cyclic column distribution

Here, the distribution of columns is done in two steps. First, the columns are divided into groups of block size NB. Afterwards, the groups are distributed in a cyclic manner to the processes. Hence, column k is owned by process $\lfloor \frac{k-1}{NB} \rfloor$ mod P. See Figure 6.4 for an illustration of the one-dimensional block data distribution.

Two-dimensional block-cyclic distribution

This layout involves the arrangement of processes in rectangular arrays. P processes are arranged in a $P_r \times P_c$ array, indexed (p_r, p_c), where $0 \leq p_r < P_r$ and $0 \leq p_c < P_c$. Columns and rows are divided in individual sizes, therefore two types of blocking factors can be defined, called MB (row block size) and NB (column block size). An example of a two-dimensional block-cyclic distribution is depicted in Figure 6.4 (*right*), where $N = 16$, $P = 4$, $P_r = 2$, $P_c = 2$, $MB = 2$, and $NB = 2$. This distribution is usually used for parallel dense linear algebra, including ScaLAPACK [LJ93, DvdGW92].

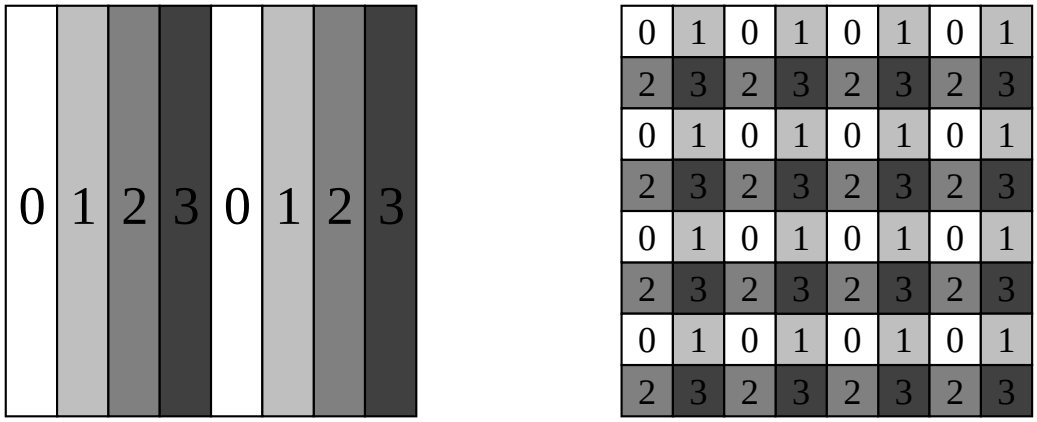

Figure 6.4: The one-dimensional block-cyclic column (*left*)- and the two-dimensional block-cyclic (*right*) distributions operating in four processes, corresponding to [BCC+97, p.60]. Levels of gray represent processes 0 to 3.

6.2 Toward a Parallel Complex Symm. Eigensolver

This section is partly based on our paper "Toward a Parallel Solver for Generalized Complex Symmetric Eigenvalue Problems" [SPSG10].

For the sequential case, we discussed two solver variants to solve generalized complex symmetric EVPs: the splitting and the non-splitting approach (see Section 5.2). Due to the better sequential runtimes, we focus now on the non-splitting solver.

Parallelization of cardinal steps of the non-splitting solver for complex symmetric generalized EVPs is investigated. While for the serial case, all steps have been discussed, we confine here to the first three steps parallel factorization, parallel reduction, and parallel tridiagonalization. For mathematical methodology and accuracy of sequential evaluations, we refer to Section 5.2.

6.2.1 Related Work

We note that high quality parallel eigensolver software is very rare. For real symmetric and for complex Hermitian (but not for complex symmetric) generalized EVPs, parallel implementations are available in ScaLAPACK, there seem to be no plans especially for complex symmetric EVPs [DDP+07].

SLEPc [HRV05] features `EPSSolve` to solve generalized EVPs, but it focuses on sparse problems and projection methods. The challenge addressed here is to develop and analyze computational kernels that are building blocks for a complete parallel complex symmetric generalized eigensolver capable of exploiting the special algebraic properties.

6.2.2 Motivation

In addition to the motivations specified for the serial solver, a parallel solver is highly motivating in order to utilize parallel computer architectures for solving EVPs that are too big for serial systems. The parallel eigensolver is especially encouraging, as most of the current computer architectures are parallel.

6.2.3 Parallel Approach

Our parallel solver for generalized complex symmetric EVPs is a ScaLAPACK-based MPI-style parallelization of the sequential codes summarized in Section 5.2 for both shared and distributed memory architectures. Our starting point is based on the parallel real symmetric solver variant `pdsygvx` (ScaLAPACK). `pdsygvx` firstly calls `pdpotrf` to factorize B, then applies `pdsyngst` to transform the generalized to a standard EVP, followed by `pdsyevx` to solve it.

In common with other ScaLAPACK driver routines, the method is parallelized by a data parallel approach utilizing a block-cyclic distribution. The hierarchy of calling subroutines, distribution- and communication schemes of our implementations are the same as in `pdsygvx` (ScaLAPACK).

The developed parallel driver routine for solving a generalized complex symmetric EVP is called `pzsygvn`. Besides auxiliary routines, it firstly calls (a) `pzpotrfi` for a parallel complex symmetric indefinite factorization, followed by (b) `pzsygst` for a parallel transformation from generalized to standard EVP, and (c) `pzsyevn` for solving the corresponding standard EVP. `pzsyevn` first applies a parallel non-splitting tridiagonalization and reduces the matrix of the standard EVP to tridiagonal form. The rest of this routine is work in progress: we do not yet have a parallel solver for computing eigenpairs of the resulting complex symmetric tridiagonal EVP. Our implementations of Steps (a)-(c) are new developments.

6.2.4 Aspects of Context Adaptivity

The concept of context adaptivity has been introduced in Chapter 3, here we discuss it with respect to the parallel generalized complex symmetric eigensolver.

Main goals

The main goal motivating our driver routine `pzsygvn` is that there is no routine to solve generalized complex symmetric EVPs in ScaLAPACK; consequently, the main goals are reduced *runtimes* running in parallel. Runtimes cannot be compared to direct competitor routines, as there is no parallel solver for general generalized EVPs; however, we can compare runtimes with parts of the Hermitian solver routine `pzheev` (ScaLAPACK), as partly similar methodology is applied.

Accompanying goals

Accompanying goals *portability*, *reusability*, *robustness*, *usability*, and *architecture adaptivity* are partly followed by building our approach on top of existing PBLAS and ScaLAPACK routines. In the following, *parallel scalability* is evaluated in detail.

Context

In analogy with the sequential approach, the parallel counterpart takes *structure of input data*, namely symmetry, into account. The *computing platform* is any parallel system capable of running the applied software (see Section 6.2.5), we do not have any *temporal requirements*. Applied *precision* is double complex. *Accuracy requirements* are mainly determined by the application, therefore we have no further requirements to the sequential case.

The new methods are adaptive to the context in the sense that they present a complementary approach for parallel architectures.

Methodology

A new approach to solve generalized complex symmetric EVPs in parallel is discussed. Different *implementation variants* are transparently realized through the utilization of the PBLAS, which guarantees excellent runtime behavior of core routines on many platforms. For the parallel case, there is

no *trading* in terms of a faster but less accurate approach, because we have no solver for such EVPs in ScaLAPACK.

Computational kernels

The investigated computational kernels are represented by parts of the steps of solving the EVP (see Figure 5.2). Therefore, involved computational kernels are the newly developed computational routines (∗) `pzpotrfi` (parallel symmetric indefinite factorization), (∗) `pzsygst` (parallel reduction operation), and (∗) `pzsytr2` (parallel complex symmetric tridiagonalization).

6.2.5 Hard- and Software Infrastructure

Parallel codes of this section were run on the supercomputer HPCx, which is now in its final stage an IBM eServer 575 cluster offering a total of 2560 CPU cores (16 on each node) on 1280 IBM Power5 processors (1.5 GHz)[2], see Figure 6.5 for a photo. HPCx started in 2002 as a six-year project provided in three phases, the cluster is now in its final phase (it received prolongation for further 14 months) [ABG+05]. The highest ranking in the TOP500 list was #9 in 2002[3]. From November 2002 to November 2008, it has constantly been listed in the TOP500. It now has a theoretical peak performance of 15.36 Teraflop/s and a sustained performance of 12.94 Teraflop/s (see TOP500 list November 2008).

Codes are written in Fortran, only the routine `pzsymv` (parallel complex symmetric matrix-vector multiplication, used here in complex symmetric tridiagonalization) is written in C, as it is based on the PBLAS which is written in C. Evaluations have been carried out on the cluster HPCx with compiler IBM xlf 10.1.0.10 and IBM ESSL 4.3.0.0 as BLAS, LAPACK 3.0, and ScaLAPACK 1.7. ScaLAPACK's communication is done with BLACS 1.1, utilizing IBM POE 4.3.2.5. The used optimal blocking factor for ScaLAPACK was experimentally determined and set to $MB = NB = 96$. Jobs are submitted as LoadLeveler scripts, with the following essential settings: `job_type=parallel`, `node_useage=not_shared`, `stack_limit=200MB`; `poe` was used to actually execute the executable.

Accuracies of parallel routines are not evaluated, but tests have been undertaken to prove correctness of the results; all matrices are dynamically generated. All major routines are subsequently evaluated according runtime behavior.

[2] An overview of its architecture can be found on the HPCx User's Guide, `http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/`.

[3] see corresponding list in the TOP500, `http://www.top500.org/site/history/2217`

Figure 6.5: Cluster HPCx, housed at Daresbury Laboratory (England) and offering 2560 CPU cores, was used for all parallel scalability studies.

6.2.6 Evaluation Strategy

All major routines are evaluated separately for problem sizes 4096, 6144, and 8192. Time measurement is done by calling `MPI_Wtime`, where `MPI_Barrier` guarantees that time is measured until the last process has finished. Running nodes are not shared with other users to avoid potentially inaccurate runtimes. Scalability is discussed in terms of the relative speedup, where the baseline is the execution of the parallel program on a single CPU core.

Sometimes, evaluations reveal a superlinear speedup, that is, the observed speedup is better than linear (occasionally called ideal) speedup. See, for example, [MFS94] for a discussion about superlinear speedups.

6.3 Shares of Runtimes

Here we discuss the runtimes on a single CPU core of HPCx of routines for symmetric factorization (`pzpotrfi`), reduction operation (`pzsygst`), and tridiagonalization (`pzsytr2`). These routines are discussed regarding their parallel scalability in the following sections.

Before evaluating parallel scalability, it is important to measure the shares of runtimes of individual routines, in order to determine dominating parts. We observe that the factorization of B (`pzpotrfi`) consumes relatively little time, transformation from generalized to standard EVP (`pzsygst`) consumes a considerably bigger share, and tridiagonalization (`pzsytr2`) consumes most of the time. The shares are as follows. For order $n = 4096$, factorization consumes 5%, reduction 40%, and tridiagonalization 55% of the time. For $n = 6144$, these percentages are 5/36/59; and for $n = 8192$ we measure 5% for factorization, 35% for reduction, and 60% for tridiagonalization. Figure 6.6 depicts absolute runtimes of the implemented routines of `pzsygvn`. Accordingly, the tridiagonalization step is the dominating part for all analyzed orders on a single CPU core of HPCx.

6.4 Parallel Transformation to Standard EVP

The transformation from the generalized to standard EVP involves (a) the symmetric factorization of B and (b) the actual reduction to standard EVP.

6.4.1 Parallel Symmetric Factorization

A parallel symmetric factorization routine `pzpotrfi` has been developed that aims especially at complex symmetric indefinite matrices. Analogously to its

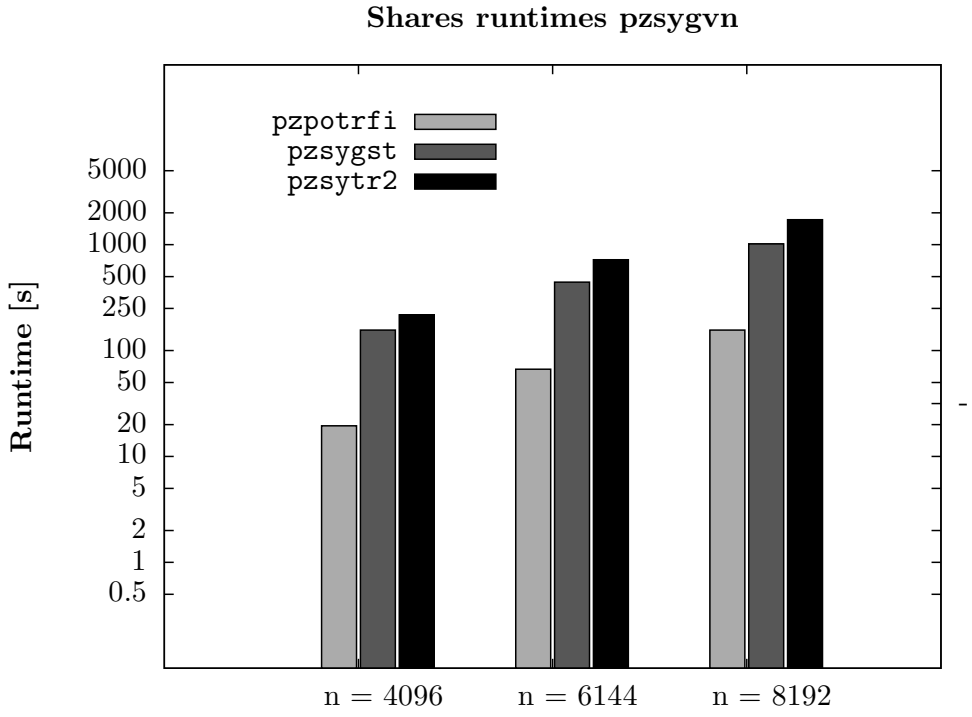

Figure 6.6: Absolute runtimes in seconds for factorization in `pzpotrfi`, reduction in `pzsygst`, and tridiagonalization in `pzsytr2` for orders $n = 4196$, 6144, and 8192 on a single CPU core on HPCx.

sequential counterpart `zpotrfi`, it factorizes $B \mapsto LL^\top$.

Related work

ScaLAPACK features parallel Cholesky factorizations for real symmetric positive definite problems in routine `pdpotrf` and Hermitian positive definite problems in `pzpotrf`, respectively. A parallel factorization of indefinite problems is currently not available in ScaLAPACK. However, a parallel BK factorization variant was developed and evaluated on a Fujitsu AP3000 distributed memory machine on 16 nodes [SL01, Str00]. Unfortunately, these papers do not discuss parallel speedups on a variable number of cores. In [JP94], reasonable speedups are observed on a parallel implementation of a BK factorization running one to four processors on a four-processor CRAY Y-MP.

Implementation

`pzpotrfi` is based on the real symmetric Cholesky factorization routine `pdpotrf` (ScaLAPACK) factorizing positive-definite real matrices. Its structure is the same as in our implementation, but some modifications, including data types and different subroutine calls, were necessary. Furthermore, the check for positive-definiteness is removed, as it is obviously conflicting with indefinite matrices.

Scalability study

Parallel symmetric factorization on HPCx has been evaluated on 2 to 1024 nodes, see Figure 6.7 for relative speedups on orders $n = 4096$, 6144, and 8192. Considering the linear speedup as ideal scalability, we observe an excellent speedup for up to 16 nodes. The reason for this behavior is the number of cores on one node, as CPU 16 cores on the same node do not need to communicate over the network. Speedups on higher number of cores involve communication and thus scalability decreases; for order $n = 8192$, we observe a speedup of about 128. Generally, bigger orders result in bigger matrices and better scalability. `pzpotrfi` scales well until about 512 cores, for 1024 cores the consumed overall runtime remains roughly stable.

6.4.2 Parallel Reduction Operation

Given the symmetric factorization of $B \mapsto LL^\top$, the second step for transforming the generalized to the standard EVP is the actual reduction.

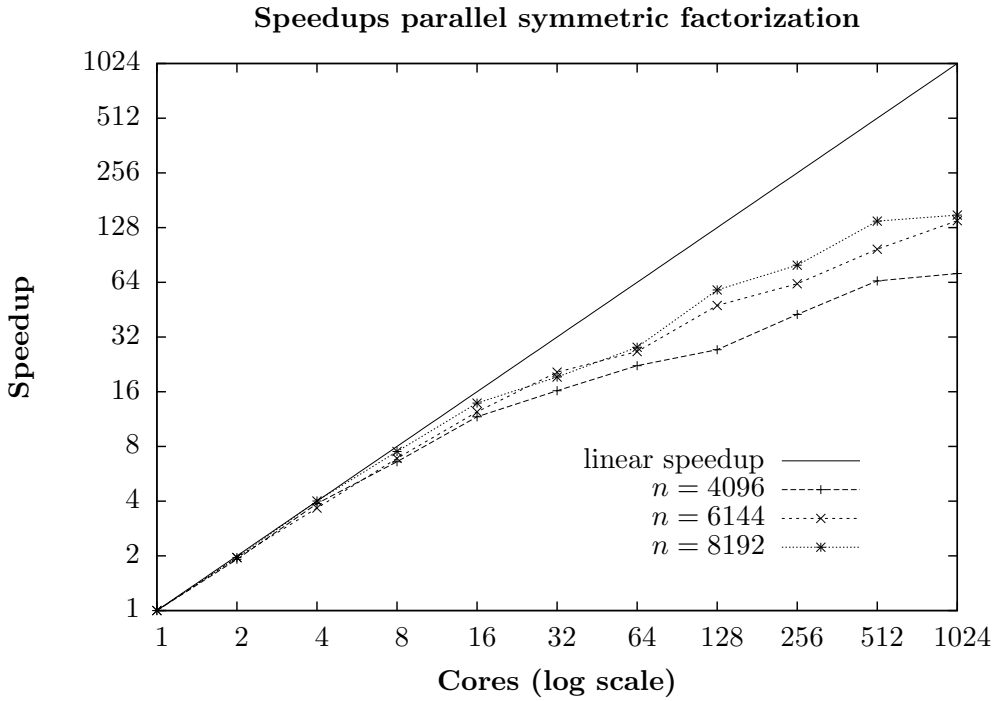

Figure 6.7: Relative (log scale) speedups of parallel complex symmetric indefinite factorization routine `pzpotrfi` factorizing $B \mapsto LL^\top$ on HPCx.

Related work

Papers focussing on the parallel reduction from generalized to standard EVPs are rare, because this subproblem is usually discussed in the course of a complete generalized eigensolver.

Implementation

The implementation of `pzsygst` is a series of calls to ScaLAPACK routines. Our implementation is algorithmically slightly different from the one in the ScaLAPACK routine `pdsygst`, which applies a transformation to the input matrix from both sides at the same time: our solution implemented in `pzsygst` is a simplified preliminary implementation of this operation for the purpose of rapid prototyping (confer the serial case in Section 5.3.2). `pzsygst` consecutively solves two linear systems of the type $LX = B$ with L from the factorization step in order to construct $M = L^{-1}AL^{-\top}$. In more detail, first `pztrtrs` (LAPACK) is used for solving $LX = A$ for X, yielding $X = L^{-1}A$. Then, X is transposed, and `pztrtrs` (ScaLAPACK) is used again for solving $LM = X$ for M, so $M = L^{-1}X = L^{-1}AL^{-\top}$.

Scalability study

Figure 6.8 depicts parallel runtimes for problem sizes $n = 4096$, 6144, and 8192 on the cluster HPCx running on 2 to 1024 cores. The vast majority of the runtime is spent in `pztrtrs` (ScaLAPACK), calls to `pztranu` (ScaLAPACK) and `pzlacpy` (ScaLAPACK) consume comparatively very little time. Hence, scalability study of the reduction process is very similar to the scalability of calling the solver of the triangular system `pztrtrs` (ScaLAPACK) twice in a row. We observe excellent scaling behavior for all tested matrix sizes, 2 to 1024 cores. A parallel reduction operation from both sides, as implemented in `pdsygst` (ScaLAPACK) might improve runtimes, however, the parallel scalability of our approach is already satisfying.

6.5 Parallel Tridiagonalization

The approach discussed here is based on the non-splitting tridiagonalization and on adapting the tridiagonalization process for real symmetric matrices implemented in the ScaLAPACK routine `pdsytrd`.

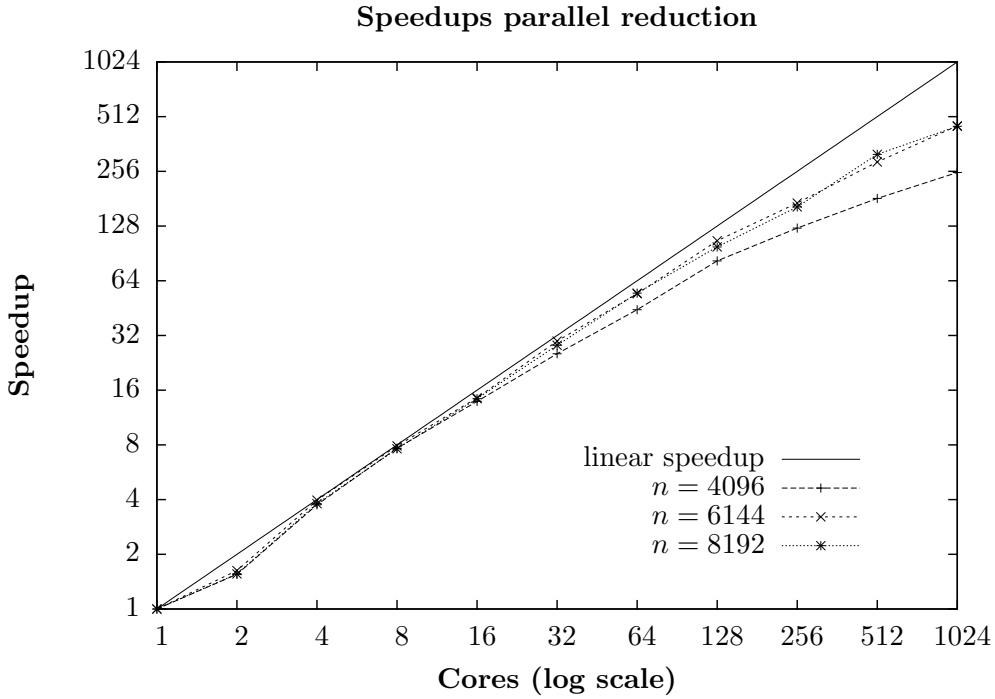

Figure 6.8: Relative (log scale) speedups of `pzsygst` transforming the generalized EVP $Ax = \lambda Bx$ to a standard EVP $My = \lambda y$ on HPCx.

6.5.1 Related Work

We do not know of any papers dealing with dense parallel tridiagonalization of complex symmetric matrices. Papers dealing with the parallel tridiagonalization process usually only discuss real symmetric or complex Hermitian cases (see, for example, [BLS94, CUSD88]), while papers dealing with complex symmetric tridiagonalization usually only deal with the serial problem (see Chapter 5).

6.5.2 Implementation

The implementation of `pzsytr2` (non-splitting tridiagonalization) is based on the ScaLAPACK codes for real symmetric matrices with driver routine `pdsytrd`. Besides modifications in various auxiliary routines, most significant changes were undertaken for the computation of the complex symmetric reflector in `pzlarfgn`, confer Section 5.4.3 for the serial case implemented in `zlarfgn`.

6.5.3 Scalability Study

Figure 6.9 depicts relative speedup values for scaling experiments. As the parallel tridiagonalization is the dominating step, we evaluate not only `pzsytr2`, but also its Hermitian pendant `pzhetrd` (ScaLAPACK). Scaling on 2 cores results in a very good performance for both routines. We observe a somewhat surprising superlinear speedup on 4 and 8 cores within a single node that we believe is caused by cache effects. Superlinear speedups on HPCx have also been reported, for example, in a thesis by E. Davidson on studying the performance of a lattice Boltzmann code [Dav08, p.33]. Experiments on 16, 32, and 64 CPU cores show satisfactory speedups; on 128, 256 and 512 CPU cores, the speedup curve degrades. For 512 CPU cores, parallelization is not feasible for matrix size 4096; there are somewhat better runtimes for matrix sizes 6144 and 8192, compared to 256 cores. 1024 CPU cores can not be utilized efficiently anymore. Furthermore, our experiments illustrate that our driver routine `pzsytr2` has a similar scaling behavior as the corresponding routine `pzhetrd` (ScaLAPACK), as can be expected from the parallelization strategy used (confer Section 6.5.2). For 1024 CPU cores and matrix sizes 4096 and 6144, `pzhetrd` (ScaLAPACK) is unexpectedly slower than for corresponding problems computed by `pzsytr2`. This deviation is hard to explain, as elapsed runtimes are very short (about $\frac{1}{100}$ second for each CPU core) and hence potentially more inaccurate than for less CPU cores.

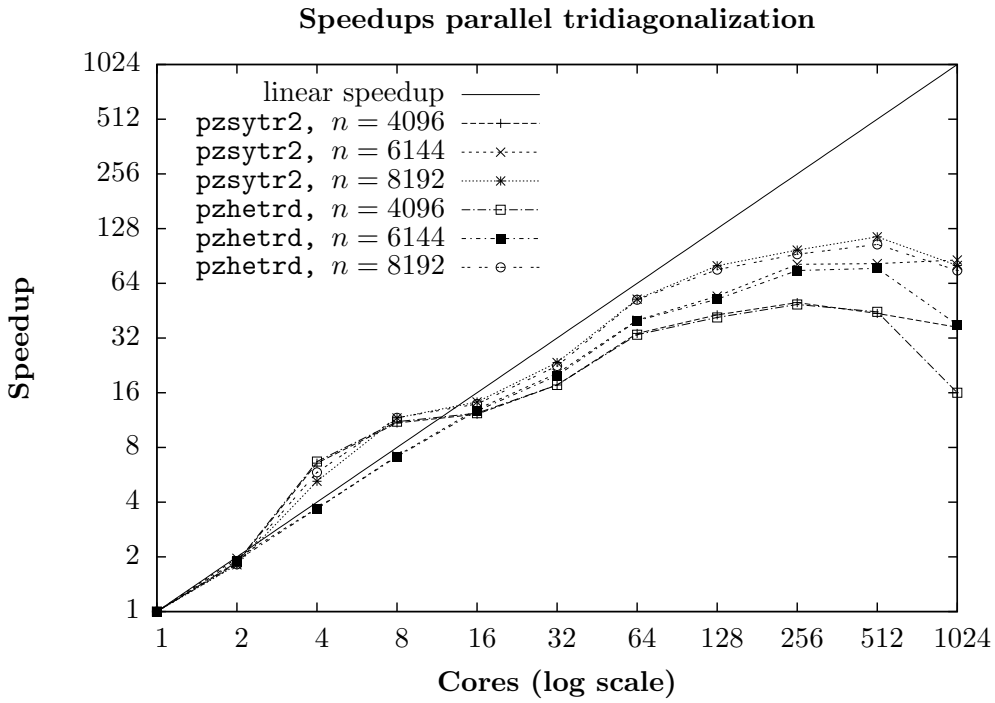

Figure 6.9: Relative (log scale) speedups of parallel complex symmetric tridiagonalization routine `pzsytr2` tridiagonalizing a standard EVP $My = \lambda y$ to a tridiagonal EVP $Tz = \lambda z$, in comparison with Hermitian tridiagonalization in `pzhetrd` (ScaLAPACK) on HPCx.

6.6 Phylogenetic Quality Assessment

This section is partly based on our paper "Phylogenetic Quality Assessment for Campus Grids" [SZvH+08].

Bootstrap analysis is a common but very time-consuming task to assess the quality of reconstructed phylogenetic trees, see Section 4.2 for basics about phylogenetic quality assessment. Here we suggest, implement, and apply a workflow based approach to distribute the bootstrap analysis on computational grid infrastructures. We use the Condor [TTL05, LLM88] grid middleware, known to be capable of controlling computational grids of various sizes. On March 30th 2010, 2266 pools comprising 308832 hosts were reported to run Condor worldwide[4]; see [TTL06] for details on how the popularity of Condor is measured. The performance of the Condor-based distributed workflow is benchmarked in comparison to the sequential analysis. We present the proof-of-concept that scientific workflow approaches on a Condor-based campus grid environment are a promising way to reduce the waiting time of embarrassingly parallel tasks in phylogenetic and bioinformatics research.

6.6.1 Related Work

Computational phylogenetics, as well as various other areas of bioinformatics, aim at achieving new biological insights by supplementary utilizing grid technology and grid resources. Bioinformatics uses databases, data processing methods, and software to get results through mass computation [Jin05]. Due to the computational demand of applications in bioinformatics, many grid-based initiatives have been launched, for instance, the OpenMolGRID [SMR+05], GLAD [TWN04], and Squid [CGdMD05]. [SMLK06] describes the concept of grid computing, [JCC+04] discusses grid as a bioinformatic tool.

Covering phylogenetics, a few past grid efforts can be observed including Grid AxML [SLMW02] (based on Cactus Tools [ABD+01]), Phylojava [SNP+03] (as part of the European DataGrid project [GJRB02]), and a Biodiversity GRID [JWG+04]. A. Stamatakis discusses some additional grid efforts in his dissertation in 2004, entitled "Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees based on the Maximum Likelihood Method".

Our approach can be classified as HTC (see Section 2.3.1), and Condor is its main exponent. The main benefit of the presented approach based on

[4] see the Condor world map, `http://www.cs.wisc.edu/condor/map/` for current statistics

Condor is the easiness of utilization of nodes featuring arbitrary computational power, that is, it eases the integration of individual computers into a powerful and mature grid (see, for example, [FMM00]). To the best of our knowledge, Condor has not yet been utilized to distribute parallel workflows for computing phylogenies.

6.6.2 Aspects of Context Adaptivity

The concept of context adaptivity has been introduced in Chapter 3, here we discuss it with respect to phylogenetic quality assessment.

Main goals

The major goal attained is to reduce the *elapsed time* of a workflow utilizing a sequential computer system to conduct phylogenetic analyses by utilizing a parallel system. This goal is achieved by a parallel approach utilizing a campus grid infrastructure. *Accuracy* of computation is not influenced by the parallel approach.

Accompanying goals

Our implementation uses the application software `seqboot` from the PHYLIP package, IQPNNI, and TREE-PUZZLE[5]; all of them are available as C source codes and can be easily compiled on modern computer systems. The middleware Condor is available as source code as well as precompiled binaries for many popular platforms. Our script is written in Perl, which is installed on many systems. Consequently, accompanying goals *portability*, *reusability*, *robustness*, *usability*, and *architecture adaptivity* are fulfilled to a high degree. *Parallel scalability* is considered by measuring efficiency of parallel testruns.

Context

The approach discussed adapts to different contexts by dynamically adapting to any parallel configuration of host computers – assuming that Condor is installed on the individual machines. Typically adequate parallel configurations are idle PCs in student labs or desktop computers during the night, including older machines featuring single- or multi-core CPUs. The utilized *computing platform* is a collection of such resources in a pool. We do not have *temporal-* or *accuracy requirements*.

[5]Details about applied software can be found in Section 6.6.5.

Methodology

Applied methodology is a *parallelization* realized by the middleware Condor that implements high throughput computing for most of the "standard" PCs used on the desktop or as servers. The demands on individual computers is mainly determined by the requirements of the core part of the application software which is IQPNNI in our case. Memory requirements are typically very low, depending on the chosen model options[6]

Computational kernels

The computational kernels under consideration are represented by individual instances of the "tree phase" (see Figure 4.3), implemented by the program IQPNNI. The computational core routine in IQPNNI is the ML evaluation of trees, more precisely, Brent's method to determine the optimal branch lengths consumes the largest share [MVHS05, Section 2.2]. The investigation is therefore focused on concurrent instances of this kernel on a distributed system, the computational kernel itself is not modified.

6.6.3 Introduction

This section gives an overview of the problem setting and the applied middleware Condor.

Problem setting

Since the evaluations of different pseudo-samples are highly independent, phylogenetic quality assessment is a promising target for grid applications to reduce the running time for analysis based on its embarrassingly parallel nature. Here we will exemplify the use of computational grid technologies for a straight-forward bootstrap analysis of ML-based phylogenies. The aim is to provide an easy setup to run such an analysis on campus grids using Condor (and possibly other systems) as middleware. The scientific challenge arises in the exploitation of a large number of computational resources at the same time to compute phylogenies in parallel.

[6] see IQPNNI manual, `http://www.cibiv.at/software/iqpnni/iqpnni-manual.html`

Condor

Condor is a specialized workload management system for compute-intensive jobs[7]. The goal of the Condor project is to develop, implement, deploy, and evaluate mechanisms and policies that support HTC on large collections of distributively owned computing resources[8]. One basic idea of Condor is to ease the exploitation of idle CPU cycles of computers and to concentrate these resources in a pool. In such a pool, every machine can serve a variety of roles, machines can serve more than one role simultaneously. One important role is the "central manager", which is a unique machine in a pool acting as negotiator between resources and resource requests. A "submit machine" is any machine in the pool configured to allow Condor jobs to be submitted. An "execute machine" is any machine in the pool being configured to allow the execution of jobs. For each execute machine, the central manager collects information including, for example, the type and current utilization of the CPU, available RAM, and operating system. On each job submission, Condor performs matchmaking between the requirements of a submitted job and the specifications of a potential execute machine.

Condor can find machines to execute programs, but it does not schedule programs based on dependencies, like it is necessary for workflows. DAGMan (Directed Acyclic Graph Manager) is a meta-scheduler for Condor that can be used to model the dependencies of input, output, and execution of one or more programs. When modeling a workflow, nodes represent programs, and edges represent dependencies. Upon execution of the workflow in the Condor environment, DAGMan submits jobs to Condor and takes care that all dependencies are fulfilled. As DAGMan submits jobs to Condor, ordering required for the directed acyclic graph is enforced. Hereby, DAGMan is in charge of scheduling, recovering, and reporting.

6.6.4 Parallel Approach

The parallel approach is a parallel scientific workflow based on the Condor middleware utilizing DAGMan to control the workflow. Most time of the whole phylogenetic process is spent in the reconstruction phase, therefore we focus on this part regarding an efficient parallelization strategy. In principle, we can choose to deploy the sequential IQPNNI or its parallel MPI-based counterpart pIQPNNI. As we want to harness idle resources of all available hosts in the Condor pool, serial instances of IQPNNI are automatically launched on

[7] see "What is Condor?" on the Condor webpage, `http://www.cs.wisc.edu/condor/description.html`

[8] see the webpage of the Condor project, `http://www.cs.wisc.edu/condor/`

available cores.

The parallel counterpart pIQPNNI was not applied within our campus grid, as a parallel program within a parallel environment produces unneeded overhead. However, pIQPNNI should be deployed whenever a local parallel system, especially a multi-core system which is not running Condor, is available (confer [SO08] for a paper about efficient computation of the phylogenetic likelihood function on multi-core architectures). In the latter case, the imposed overhead of Condor is expected to be larger then the overhead imposed by the utilization of an MPI environment.

6.6.5 Implementation

In a first step, a campus grid infrastructure based on the Condor middleware has been implemented at the Faculty of Computer Science of the University of Vienna. It is planned to expand it by host computers of the Faculty of Life Sciences, see Figure 6.10. To setup non-local grid infrastructures can be a highly non-trivial task, see for example [SG09, CBK+04].

The parallel scientific workflow consists of the three steps sampling phase, reconstruction phase, and consensus phase (confer Section 4.2). In the sampling phase, many pseudo-samples are generated with seqboot (from PHYLIP 3.67[9] [Fel89]. In the construction phase, these ML tress are reconstructed using IQPNNI 3.2[10] [MVSvH06, MVHS05, VvH04]. IQPNNI is a program to infer maximum-likelihood phylogenetic trees from DNA or protein data with a large number of sequences. Finally, they are assembled into a relative majority consensus tree, as implemented in TREE-PUZZLE 5.2[11] [Sch03, SSVvH02]. This bootstrap analysis workflow with ML-based phylogeny reconstruction has been ported to a grid infrastructure, that is, it has been gridified. For further examples of gridifications in bioinformatics, see for example [SMLK06].

seqboot, IQPNNI, and TREE-PUZZLE are executed with standard parameters, except for TREE-PUZZLE the `tree search procedure` is set to "quartett puzzling" and the applied `parameter estimation` is set to "neighbor-joining tree". See the respective manuals for further possible options.

The workflow is implemented as a single DAGMan description file, containing the following information: (∗) the list of our three individual jobs and (∗) dependencies constituting a directed acyclic graph applied by declaring a static `parent` / `child` structure (see Figure 6.11 (*left*)). The sampling phase and the consensus phase are implemented as single jobs in separate Condor

[9] see the webpage of PHYLIP, `http://evolution.genetics.washington.edu/phylip.html`

[10] see the webpage of IQPNNI, `http://www.cibiv.at/software/iqpnni/`

[11] see the webpage of TREE-PUZZLE, `http://www.tree-puzzle.de/`

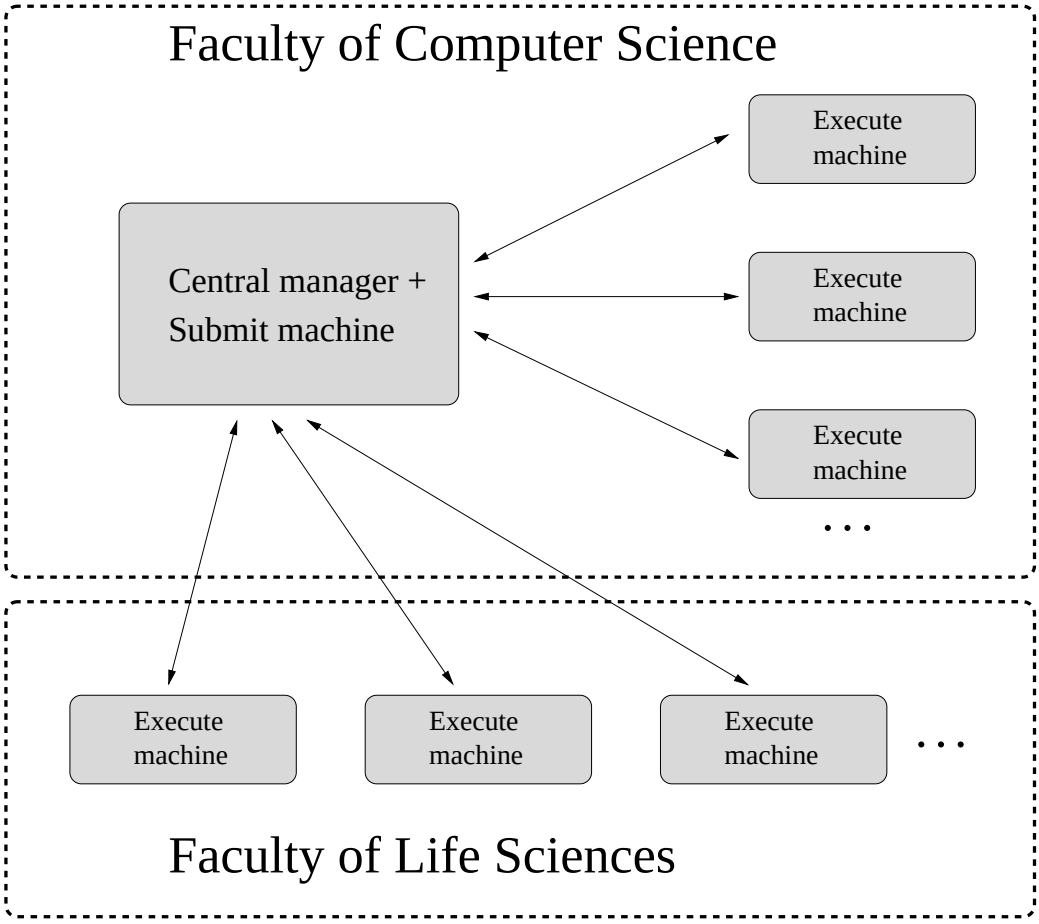

Figure 6.10: Experimental setup of the Condor-based campus grid at the University of Vienna, where a Sun v40z is used to administer jobs, while execute machines constitute a pool of available computers.

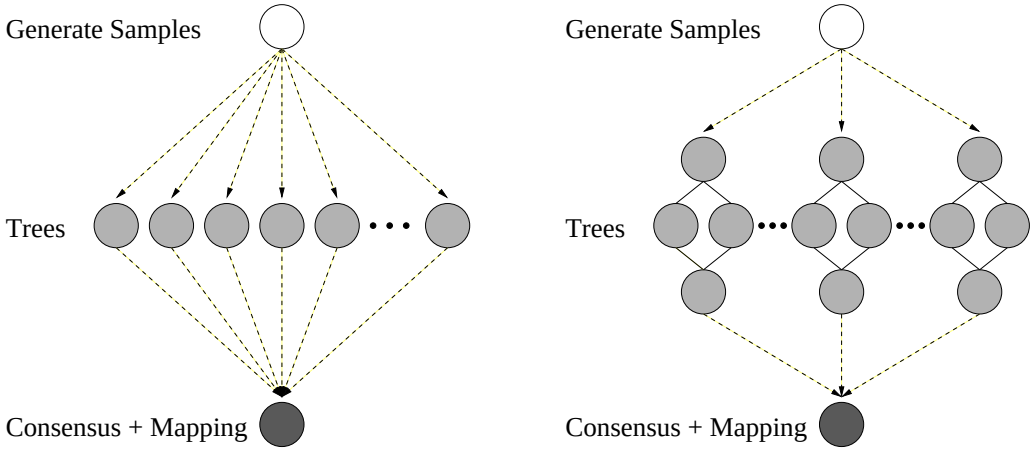

Figure 6.11: Parallelization strategies for the phylogenetic workflow. The (*left*) workflow represents a serial approach, where parallelization may be accomplished by parallel execution of individual tree instances; the (*right*) workflow represents a parallel approach within the tree phase, where parallelization is applied during the construction of single trees.

description files. In the reconstruction phase, a configurable number of parallel jobs are implemented with a single Condor description file created by our Perl script. Once the description file has been created, the workflow can be executed by submitting the corresponding DAGMan file. Subsequently, DAGMan takes control over the workflow and enforces all needed dependencies of individual steps; there is no user interaction necessary during the execution of the whole workflow.

Testbed

The experimental setup comprises machines in two different locations. Location (a) features the Condor central manager, which is a Sun v40z, running Suse Enterprise Server 10, see Section 5.2.6 for more details. This machine exhibits 4 dual core Opteron CPUs, whose eight cores are treated by Condor as eight separate execute machines. In location (b), four desktop PCs connected by a 100 Mbit link have been added to the Condor pool. They contain Intel Core 2 Duo CPUs resulting in eight additional cores for our Condor pool. In total, this experimental campus grid offers 16 CPU cores. Each core can be accessed by Condor to execute one program instance at a time.

Evaluation strategy

We assess the computing node performance [Dik07] which is the performance of a single node, by comparing the sequential runtime of the bootstrap workflow with the Condor-distributed workflow. To measure the runtime overhead of the distributed workflow, the following setup has been performed on the eight cores of the Condor central manager machine. We executed eight sequential workflows simultaneously to reduce possible biases like cache effects or shared memory effects in favor of the sequential version running on an idle eight-core computer. In the parallel case, we just submit our workflow to Condor. As runtimes for the distributed (T_{parallel}) and the average sequential case ($T_{\text{sequential}}$) we used the mean wall clock time of all respective runs. Then we computed the parallel Speedup $S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}$ and the efficiency $E[\%] = 100 \cdot \frac{S}{p}$, where p is the number of cores available, here eight.

6.6.6 Results

We applied the implemented distributed workflow using the grid above to two different datasets. We used a small dataset of Elongation factor (EF) sequences from the TREE-PUZZLE software which allowed for running extensive sequential and distributed benchmarks in reasonable time. Furthermore we used a real-world dataset of ribosomal RNA (rRNA) sequences spanning all domains of life. We executed the analyses with 1000 bootstrap samples assuming rate heterogeneity with four Γ-rate categories.

The benchmark tests with the small set showed a promising efficiency E of 94% (that is, speedup $S = 7.52$) compared to the sequential run. Note that the runtime of the different pseudo-samples varies due to the stochastic nature of the resampling procedure. Thus, at the end of the workflow the number of cores actually computing decreases until the last reconstruction has finished leaving more and more cores idle. Most of the 6% overhead missing in the efficiency can be attributed to the idle cores at the end of the ML tree computation.

The reconstructed rRNA phylogeny with bootstrap support values is depicted in Figure 6.12. The sequences range over all domains of life, the Archean sequences serve as outgroup to root the tree, the Bacterial sequences are depicted in black, and the three different types of eucaryotic sequences are colored. It is nicely shown where the three different genome sequences found in eucaryotes have originated. The nuclear sequences (*blue*) can be found at the root of the Bacterial tree. The position and bootstrap values support that the origin of the chloroplasts (*green*) and the mitochondria (*red*) was

indeed an endosymbiosis event, namely of a cyanobacterium-like ancestor for the chloroplasts and an α-bacterium-like ancestor.

The tree also shows that with the current dataset we cannot resolve all bacterial relationships with confidence. The bootstrap analysis of the rRNA dataset took less than 8.5 hours compared to an estimated running time of more than 5.5 days on a normal single-core PC. This gives clear evidence that Condor-based workflows can easily and substantially reduce the running time of large scale phylogenetic bootstrap analysis without introducing a large overhead for the user.

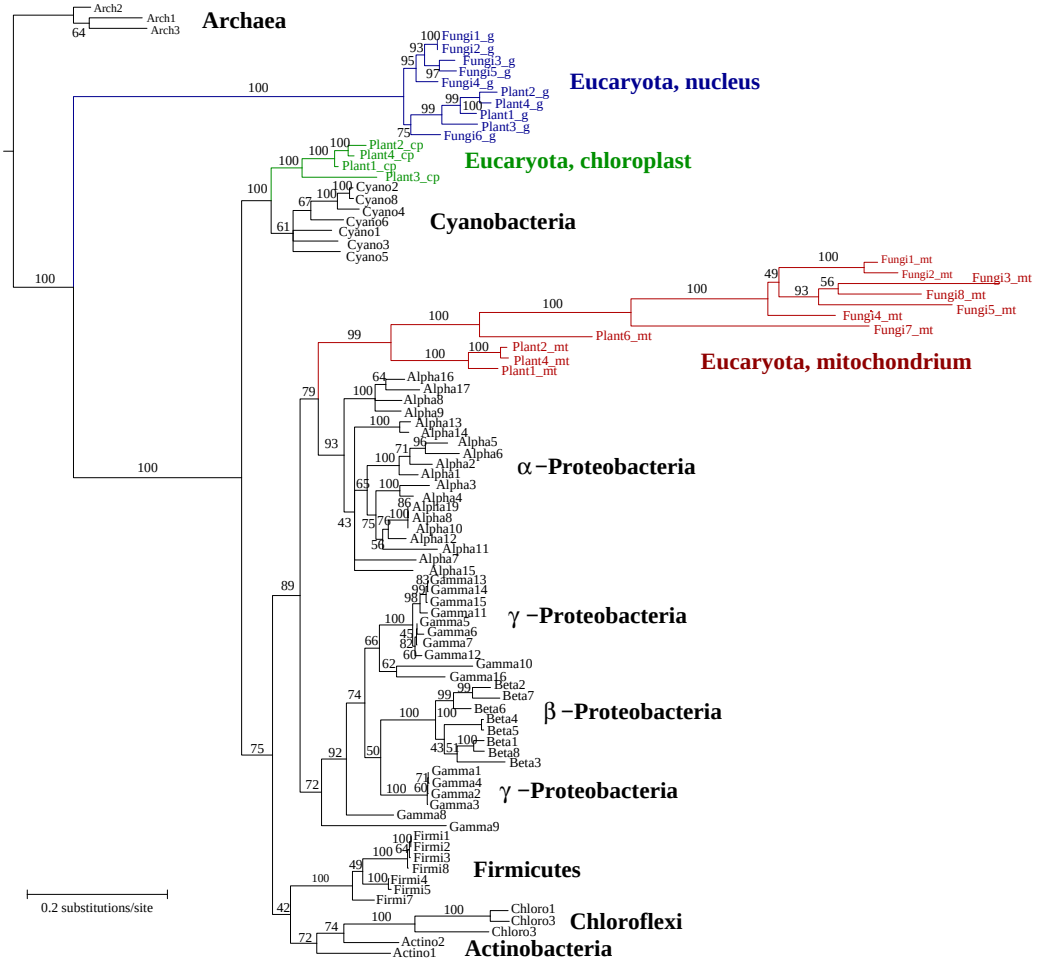

Figure 6.12: Phylogenetic rRNA tree, as computed by our parallel grid workflow implementation utilizing Condor and DAGMan; the numbers at the branches denote bootstrap support values, the sequences range over all domains of life, the Archean sequences serve as root, the three different types of eucaryotic sequences are well presenting the two endosymbiosis events of Bacteria as mitochondria and plastids into eucaryotic cells.

Chapter 7

Conclusions

This chapter comprises conclusions and future work of this dissertation for the respective computational problems.

7.1 Sequential Generalized Complex Symmetric EVP

Motivated by application problems arising in optoelectronics, serial components aiming at solving generalized complex symmetric EVPs have been developed and evaluated in terms of accuracy and runtimes. The state-of-the-art approach for solving such EVPs is to call a general solver (for example, `zggev` (LAPACK)) and abstain from utilizing the special properties of complex symmetric matrices.

7.1.1 Achievements

Driver routine `zsygvs` provides a complete generalized complex symmetric solver, consisting of the parts factorization, reduction, tridiagonalization, and computation of eigenpairs (including backtransformation). Driver routine `zsygvn` provides a generalized complex symmetric solver for computing eigenvalues (eigenvectors not supported yet) featuring a similar approach, but with a different tridiagonalization step. All codes are implemented in Fortran and (with the exception of the tridiagonal solver) based on BLAS and LAPACK function calls.

7.1.2 Results

`zsygvs` is faster than its competitor `zggev` (LAPACK), however, this comes for the price of potentially lower accuracy. `zsygvn` computes eigenvalues even faster than `zsygvs`, and therefore also faster than `zggev` (LAPACK), but less accurate than `zsygvs` and `zggev` (LAPACK). Consequently, our approaches trade potentially smaller accuracy by exploiting structure for better runtimes.

7.1.3 Future Work

The following ideas may further improve runtime performance of `zsygvs`. Current splitting tridiagonalization routine `zsytr1` is operating on single rows and columns. A blocked variant could dramatically increase runtime performance. Backtransformation of eigenvectors from tridiagonal problem to standard problem and from standard problem to the generalized problem are currently done in an inefficient way. Faster solutions are demonstrated in `dsyev` (LAPACK) for the standard EVP and `dsygv` (LAPACK) for the generalized EVP).

In `zsygvs`, the complex orthogonal transformation (COT) contributes greatly to the error of the tridiagonalization process. An improved COT could dramatically increase overall accuracy.

Backtransformation of eigenvectors in `zsygvn` from the tridiagonal to the standard EVP is currently unsupported, precluding the computation of eigenvectors of the generalized problem; see `dsygv` (LAPACK) for a starting point how to realize this.

The following approach may improve accuracy achieved with `zsygvn`. Computation of the complex elementary reflector in `zlarfgn` may be improved by a rescaling method, as in the LAPACK routine for real symmetric problems `dlarfg`.

7.2 Parallel Generalized Complex Symmetric EVP

Motivated by the huge runtime costs of solving big EVPs, a parallel approach especially aimed at generalized complex symmetric EVPs has been discussed. The obvious approach to treat complex symmetric EVPs as general ones is currently unavailable in ScaLAPACK. A parallel approach tailored for complex symmetric EVPs has been developed, where so far the cardinal steps factorization, transformation to standard EVP, and non-splitting tridiagonalization have been implemented and evaluated.

7.2.1 Achievements

Driver routine `pzsygvn` provides the framework of a generalized complex symmetric eigensolver, where the parts factorization in `pzpotrfi`, transformation to standard EVP in `pzsygst`, and non-splitting tridiagonalization in `pzsytr2` have already been implemented.

7.2.2 Results

As `pzsygvn` currently lacks some parts, only its computational routines can be evaluated separately. In terms of runtimes, we observe encouraging scalability for parallel executions on HPCx for evaluations on up to 1024 cores. However, these results heavily depend on the properties of the system (especially network performance) and the order of the problem.

7.2.3 Future Work

The ultimate goal is the implementation of a complete parallel generalized complex symmetric eigensolver. In the course of this ambition, the cardinal steps tridiagonal solver, and backtransformation have to be undertaken. While for backtransformation, a similar parallel code to the serial case should be implemented, the situation for the tridiagonal solver is more difficult. As the current serial tridiagonal solver is not based on LAPACK codes, there is no corresponding variant in ScaLAPACK.

7.3 Phylogenetic Quality Assessm. f. Campus Grids

Due to vast computational demands of computing large phylogenies, the bootstrap analysis workflow of ML-based phylogenies has been ported to a Condor campus grid. Condor was chosen, as it is a very major grid middleware easening the dynamic integration of both dedicated and otherwise idle workstations.

7.3.1 Achievements

The parallel computation of phylogenies (here featuring IQPNNI as a central component) on a Condor campus grid is a novelty. Furthermore, we prototypically assess the performance overhead in order to justify the approach.

7.3.2 Results

Due to very limited communication during the processing of the workflow and a potentially big number of available CPU cycles, this approach is very encouraging. Our evaluations reveal an efficiency of 94% on the used Condor campus grid infrastructure.

7.3.3 Future Work

In our testbed, Condor jobs were running on each individual core of a multi-core system. A complemental variant is to use the parallel version of IQPNNI to run once on each multi-core machine, and to use the serial version only on single-core machines. This might lead to a reduced parallel communication overhead.

List of Figures

List of Tables

Zusammenfassung

Einführung

Computational Kernels sind der kritische Teil rechenintensiver Software, wofür der größte Rechenaufwand anfällt; daher müssen deren Design und Implementierung sorgfältig vorgenommen werden. Zwei wissenschaftliche Anwendungsprobleme aus der Optoelektronik und aus der Phylogenetik, sowie dazugehörige Computational Kernels motivieren diese Arbeit. Im ersten Anwendungsproblem werden Komponenten zur Berechnung komplex-symmetrischer Eigenwertprobleme diskutiert, welche in der Simulation von Wellenleitern in der Optoelektronik auftreten. LAPACK und ScaLAPACK beinhalten sehr leistungsfähige Referenzimplementierungen für bestimmte Problemstellungen der linearen Algebra. In Bezug auf Eigenwertprobleme werden ausschließlich reell-symmetrische und komplex-hermitesche Varianten angeboten, daher sind effiziente Codes für komplex-symmetrische (nicht-hermitesche) Eigenwertprobleme sehr wünschenswert. Das zweite Anwendungsproblem behandelt einen parallelen, wissenschaftlichen Workflow zur Rekonstruktion von Phylogenien, welcher entworfen, umgesetzt und evaluiert wird. Die Rekonstruktion von phylogenetischen Bäumen ist ein NP-hartes Problem, welches äußerst viel Rechenkapazität benötigt, wodurch ein paralleler Ansatz erforderlich ist.

Die grundlegende Idee dieser Arbeit ist die Untersuchung der Wechselbeziehung zwischen dem Kontext der behandelten Kernels und deren Effizienz. Ein Kontext eines Computational Kernels beinhaltet Modellaspekte (z.B. Struktur der Eingabedaten), Softwareaspekte (z.B. rechenintensive Bibliotheken), Hardwareaspekte (z.B. verfügbarer Hauptspeicher und unterstützte darstellbare Genauigkeit), sowie weitere Anforderungen bzw. Einschränkungen. Einschränkungen sind hinsichtlich Laufzeit, Speicherverbrauch, gelieferte Genauigkeit usw., möglich.

Methodik

Das Konzept der Kontextadaptivität wird für ausgewählte Anwendungsprobleme in Computational Science gezeigt. Die vorgestellte Methode ist ein Meta-Algorithmus, der Aspekte des Kontexts verwendet, um optimale Leistung hinsichtlich der angewandten Metrik zu erzielen. Es ist wichtig, den Kontext einzubeziehen, weil Anforderungen gegeneinander ausgetauscht werden könnten, resultierend in einer höheren Leistung. Zum Beispiel kann im Falle einer niedrigen benötigten Genauigkeit ein schnellerer Algorithmus einer bewährten, aber langsameren, Methode vorgezogen werden. Speziell für komplex-symmetrische Eigenwertprobleme zugeschnittene Codes zielen darauf ab, Genauigkeit gegen Geschwindigkeit einzutauschen. Die Innovation wird durch neue algorithmische Ansätze belegt, welche die algebraische Struktur ausnutzen. Bezüglich der Berechnung von phylogenetischen Bäumen wird die Abbildung eines Workflows auf ein Campusgrid-System gezeigt. Die Innovation besteht in der anpassungsfähigen Implementierung des Workflows, der nebenläufige Instanzen von Computational Kernels in einem verteilten System darstellt. Die Adaptivität bezeichnet hier die Fähigkeit des Workflows, die Rechenlast hinsichtlich verfügbarer Rechner, Zeit und Qualität der phylogenetischen Bäume anzupassen.

Beiträge

Kontextadaptivität wird durch die Implementierung und Evaluierung von wissenschaftlichen Problemstellungen aus der Optoelektronik und aus der Phylogenetik gezeigt.

Für das Fachgebiet der Optoelektronik zielt eine Familie von Algorithmen auf die Lösung von verallgemeinerten komplex-symmetrischen Eigenwertproblemen ab. Unser alternativer Ansatz nutzt die symmetrische Struktur aus und spielt günstigere Laufzeit gegen eine geringere Genauigkeit aus. Dieser Ansatz ist somit schneller, jedoch (meist) ungenauer als der konventionelle Lösungsweg. Zusätzlich zum sequentiellen Löser wird eine parallele Variante diskutiert und teilweise auf einem Cluster mit bis zu 1024 CPU-Cores evaluiert. Die erzielten Laufzeiten beweisen die Überlegenheit unseres Ansatzes – allerdings sind weitere Untersuchungen zur Erhöhung der Genauigkeit notwendig.

Für das Fachgebiet der Phylogenetik zeigen wir, dass die phylogenetische Baum-Rekonstruktion mittels eines Condor-basierten Campusgrids effizient parallelisiert werden kann. Dieser parallele wissenschaftliche Workflow weist einen geringen parallelen Overhead auf, resultierend in exzellenter Effizienz.

Scientific Resume

Name Hannes Schabauer

Date and place of birth 1974/03/28 in Vienna (Austria); Austrian

Contact University of Vienna, Universitätsstraße 10/9, Vienna (Austria);
✉ mailto:hannes.schabauer@gmail.com

Education

Magister Mag. rer. soc. oec. in *business informatics (Wirtschaftsinformatik)* from University of Vienna and Vienna University of Technology (2004)

Employment in scientific projects

University of Lausanne *Selectome, Looking for Darwinian Evolution in the Tree of Life*; Evolutionary Bioinformatics Group; 2010/04 -

University of Vienna *Computing Paradigms and Algorithms for Molecular Modeling and Simulation (CPAMMS)*; Research Lab Computational Technologies and Applications; 2006/11 - 2009/12.

Austrian Research Centers (ARC) Seibersdorf *Computational Methods for Complex Symmetric Eigenproblems in Optoelectronics*; Smart Systems Division, Optoelectronic Engineering; 2006/07 - 2006/10.

University of Vienna *Advanced Models, Applications and Software Systems for High Performance Computing (AURORA)*; Department of Statistics and Decision Support Systems; 2004/09 - 2006/06.

Scientific publications

[**SPSG10**] *Toward a Parallel Solver for Generalized Complex Symmetric Eigenvalue Problems*; Schabauer, Pacher, Sunderland, Gansterer, 2010.

[**SGSP10**] *Parallel Solution of Generalized Complex Symmetric Eigenproblems – Evaluation on HPCx*; Schabauer, Gansterer, Sunderland, Pacher, 2010.

[**SZvH+08**] *Phylogenetic Quality Assessment for Campus Grids*; Schabauer, Zottl, Arndt von Haeseler, Gansterer, Schmidt, 2008.

[**GSPF08**] *Tridiagonalizing Complex Symmetric Matrices in Waveguide Simulations*; Gansterer, Schabauer, Pacher, Finger, 2008.

[**SHP08**] *Parallelization of Pricing Path-Dependent Financial Instruments on Bounded Trinomial Lattices*; Schabauer, Hochreiter, Pflug, 2008.

[**HGS+08**] *Enhancing ZRTP by using Computational Puzzles*; Hlavacs, Gansterer, Schabauer, Zottl, Petraschek, Hoeher, Jung, 2008.

[**VTS+07**] *On the elusive nature of the letters from the heart effect*; Voracek, Tran, Schabauer, Koenne, Glössl, 2007.

[**SSW05**] *Solving Very Large Traveling Salesman Problems by SOM Parallelization on Cluster Architectures*; Schabauer, Schikuta, Weishäupl, 2005.

[**Sch04**] *Parallelisierung von Kohonen-Netzen auf Cluster-Architekturen*; Schabauer, 2004.

[**SV01**] *LetterScan – Ein PC-Programm zur optimalen Selektion von Buchstabenkombinationen als Items oder Experimentalstimuli*; Schabauer, Voracek, 2001.

Talks at conferences and workshops

ICCS 2010 *Toward a Parallel Solver for Generalized Complex Symmetric Eigenvalue Problems*; 10$^{\text{th}}$ International Conference on Computational Science; Amsterdam (Netherlands).

TAM 2009 *Parallel Solution of Generalized Complex Symmetric Eigenproblems*; Transnational Access Meeting; Montpellier (France).

ICCS 2008 *Parallelization of Pricing Path-Dependent Financial Instruments on Bounded Trinomial Lattices*; 8th International Conference on Computational Science; Krakow (Poland).

ANADAY 2008 *Tridiagonalizing Complex Symmetric Matrices*; 4th Austrian Numerical Analysis Day; Linz (Austria).

PDCAT 2005 *Solving Very Large Traveling Salesman Problems by SOM Parallelization on Cluster Architectures*; 6th International Conference on Parallel and Distributed Computing; Dalian (China).

FRICO 2005 *A Parallel Heuristic Algorithm for the Traveling Salesman Problem, based on Neural Networks*; 9th Workshop Future Research in Combinatorial Optimization; Vienna (Austria).

Internal talks *Available Middleware for Computational Grids*; CPAMMS kickoff meeting (2006). *Advanced Pricing of Bounded Lattices*; Aurora meeting; (2006). *Tutorial on HPF*; University of Vienna; (2005). *High Performance Computing, Components, and Workflows*; Webopt / Riskplan project meeting; (2005). *Parallelization Techniques*; Brunel-Vienna Cooperation Day; (2005). *Parallelization of Net Present Value Calculations*; Aurora meeting; (2005). *Parallelization of Pricing Algorithms*; Aurora meeting; (2004).

Teaching experience

I have been teaching the following courses at the University of Vienna.

- *Introduction to Programming in C++*, *Internet Technologies*, *Network Performance Evaluation*, *Technical Basics and System Software.*

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 3rd edition, 1999. ISBN 9780898714470.

[ABCH+04] Louigi Addario-Berry, Benny Chor, Mike Hallett, Jens Lagergren, Alessandro Panconesi, and Todd Wareham. Ancestral Maximum Likelihood of Evolutionary Trees is Hard. *Journal of Bioinformatics and Computational Biology (JBCB)*, 2(2):257–271, 2004. DOI 10.1142/S0219720004000557.

[ABD+90] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. LAPACK: a portable linear algebra library for high-performance computers. In *Supercomputing (SC)*, pages 2–11. IEEE, 1990. DOI 10.1109/SUPERC.1990.129995.

[ABD+01] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf. Cactus Tools for Grid Applications. *Cluster Computing*, 4(3):179–188, 2001. DOI 10.1023/A:1011491422534.

[ABG+05] Mike Ashworth, Ian J. Bush, Martyn F. Guest, Andrew G. Sunderland, Stephen Booth, Joachim Hein, Lorna Smith, Kevin Stratford, and Alessandro Curioni. HPCx: towards capability computing. *Concurrency and Computation: Practice and Experience*, 17(10):1329–1361, 2005. DOI 10.1002/cpe.895.

[ABH+09] Jeanne C. Adams, Walter S. Brainerd, Richard A. Hendrickson, Richard E. Maine, Jeanne T. Martin, and Brian T. Smith. *The Fortran 2003 Handbook – The Complete Syntax, Features and Procedures*. Springer, 2009. ISBN 9781846283789.

[AC08] Peter Arbenz and Oscar Chinellato. On solving complex-symmetric eigenvalue problems arising in the design of axisymmetric VCSEL devices. *Applied Numerical Mathematics*, 58(4):381–394, 2008. DOI 10.1016/j.apnum.2007.01.019.

[ACF+07] Eric Allen, David Chase, Christine Flood, Victor Luchangco, Jan-Willem Maessen, Sukyoung Ryu, and Guy L. Stelle Jr. Project Fortress – A multicore language for multicore processors. *Linux Magazine*, pages 38–43, Sep 2007.

[AD89] Ed Anderson and Jack Dongarra. Evaluating Block Algorithm Variants in LAPACK. In *4th SIAM Conference on Parallel Processing for Scientific Computing*, pages 3–8. SIAM, 1989.

[AH04] Peter Arbenz and Michiel E. Hochstenbach. A Jacobi-Davidson Method for Solving Complex Symmetric Eigenvalue Problems. *SIAM Journal on Scientific Computing*, 25(5):1655–1673, 2004. DOI 10.1137/S1064827502410992.

[Air07] Airbus-led Research Programme to Improve the Design Process for Future Products. *Airbus Letter*, Feb/Mar 2007.

[AS88] William C. Athas and Charles L. Seitz. Multicomputers: Message-Passing Concurrent Computers. *Computer*, 21(8):9–24, 1988. DOI 10.1109/2.73.

[ASC+00] B. Aiken, J. Strassner, B. Carpenter, I. Foster, C. Lynch, J. Mambretti, R. Moore, and B. Teitelbaum. Network Policy and Services: A Report of a Workshop on Middleware, 2000. RFC 2768.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Parallel and Distributed Computing. Wiley / IEEE, 2nd edition, 2004. ISBN 9780471453246.

[Bai05] David H. Bailey. High-Precision Floating-Point Arithmetic in Scientific Computation. *Computing in Science and Engineering*, 7(3):54–61, 2005. DOI 10.1109/MCSE.2005.52.

[BB01] Peter Bienstman and Roel Baets. Optical modelling of photonic crystals and VCSELs using eigenmode expansion and perfectly matched layers. *Optical and Quantum Electronics*, 33(4/5):327–341, 2001. DOI 10.1023/A:1010882531238.

[BB05] Mary Bazire and Patrick Brézillon. Understanding Context Before Using It. In *5th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT)*, volume 3554 of *LNAI*, pages 29–40. Springer, 2005. DOI 10.1007/11508373_3.

[BBE+08] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. *PETSc Users Manual*, 2008.

[BC07] Richard Bronson and Gabriel B. Costa. *Linear Algebra – An Introduction*. Elsevier, 2nd edition, 2007. ISBN 9780120887842.

[BCC+96] L. S. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance. *Computer Physics Communications*, 97(1-2):1–15, 1996. DOI 10.1016/0010-4655(96)00017-3.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997. ISBN 9780898713978.

[BDD+00] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, 2000. ISBN 9780898714715.

[BDD+02] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *Transactions on Mathematical Software*, 28(2):135–151, 2002. DOI 10.1145/567806.567807.

[BDM89] Z. Bai, J. Demmel, and A. McKenney. On Floating Point Errors in Cholesky. Technical Report UT-CS-89-87, UTK, 1989. LAWN 14.

[Ber03] Natacha Beréux. A Cholesky algorithm for some complex symmetric systems. Technical Report 515, École Polytechnique, 2003.

[Ber05] Natacha Beréux. Fast direct solvers for some complex symmetric block Toeplitz linear systems. *Linear Algebra and Its Applications*, 404:193–222, 2005. DOI 10.1016/j.laa.2005.02.028.

[BFW92] Hans L. Bodlaender, Mike R. Fellows, and Tandy J. Warnow. Two strikes against perfect phylogeny. In *19th International Colloquium on Automata, Language, and Programming (ICALP)*, volume 623 of *LNCS*, pages 273–283. Springer, 1992. DOI 10.1007/3-540-55719-9_80.

[BGMS97] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. 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. Birkhäuser Press, 1997.

[BGW04] Yihua Bai, Wilfried N. Gansterer, and Robert C. Ward. Block Tridiagonalization of "Effectively" Sparse Symmetric Matrices. *ACM Transactions on Mathematical Software*, 30(3):326–352, 2004. DOI 10.1145/1024074.1024078.

[BH98] Henri E. Bal and Matthew Haines. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, 6:74–84, 1998. DOI 10.1109/4434.708258.

[Bis88] Christian H. Bischof. A Parallel QR Factorization Algorithm using Local Pivoting. In *Supercomputing (SC)*, volume 1, pages 400–407. IEEE, 1988. DOI 10.1109/SUPERC.1988.44678.

[Bis04] Rob H. Bisseling. *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*. Oxford University Press, 2004. ISBN 9780198529392.

[BK77] James R. Bunch and Linda Kaufman. Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems. *Mathematics of Computation*, 31(137):163–179, 1977.

[BLA01] Basic Linear Algebra Subprograms Technical Forum Standard. *International Journal of High Performance Applications and Supercomputing*, 16(1/2):1–111/115–199, 2001.

[BLB+05] M. R. Benioff, E. D. Lazowska, R. Bajcsy, J. C. Beese, P. Celis, P. Thomas Evans, M. A. Fernandez, L. E. Fiallo, J.-M. Griffiths, W. J. Hannigan, J. C. Javitt, J. L. Klavans, F. Thomson Leighton, H. Mortazavian, R. D. Mott, P. M. Neupert, E. M. Noam, D. A. Patterson, A. G. Quintanilla, D. A. Reed, E. H. Spafford, D. H. Staelin, P. S. Tippett, and G. Yang. Report To The President – Computational Science: Ensuring America's Competitiveness. Technical report, The President's Information Technology Advisory Committee (PITAC), June 2005.

[BLK+07] Alfredo Buttari, Piotr Luszczek, Jakub Kurzak, Jack Dongarra, and George Bosilca. A Rough Guide to Scientific Computing On the PlayStation 3. Technical Report UT-CS-07-595, UTK, Departmenf of Computer Science, 2007.

[BLS94] Christian Bischof, Bruno Lang, and Xiaobai Sun. Parallel Tridiagonalization through Two-Step Band Reduction. In *Scalable High-Performance Computing Conference*, pages 23–27. IEEE, 1994. DOI 10.1109/SHPCC.1994.296622.

[BOP98] Ilan Bar-On and Marcin Paprzycki. High Performance Solution of the Complex Symmetric Eigenproblem. *Numerical Algorithms*, 18(2):195–208, 1998. DOI 10.1023/A:1019121515827.

[BOR97] Ilan Bar-On and Victor Ryaboy. Fast Diagonalization of Large and Dense Complex Symmetric Matrices, with Applications to Quantum Reaction Dynamics. *SIAM Journal on Scientific Computing*, 18(5):1412–1435, 1997. DOI 10.1137/S1064827594269056.

[Bre06] Claude Brezinski. The life and work of André Cholesky. *Numerical Algorithms*, 43(3):279–288, 2006. DOI 10.1007/s11075-006-9059-x.

[Bud63] C. Donald La Budde. The Reduction of an Arbitrary Real Square Matrix to Tridiagonal Form Using Similarity Transformations. *Mathematics of Computation*, 17:433–437, 1963. DOI 10.2307/2004005.

[Bä94] Thomas Bäck. Parallel Optimization of Evolutionary Algorithms. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *3rd Conference on Parallel Problem Solving from Nature (PPSN)*, LNCS, pages 418–427. Springer, 1994. DOI 10.1007/3-540-58484-6_285.

[CBK+04] M. Calleja, B. Beckles, M. Keegan, M. A. Hayes, A. Parker, and M. T. Dove. CamGrid: Experiences in constructing a university-wide, Condor-based, grid at the University of Cambridge. In *UK e-Science All Hands Meeting*. Engineering and Physical Sciences Research Council (EPSRC), 2004.

[CCB99] William W. Carlson, David E. Culler, and Eugene Brooks. Introduction to UPC and Language Specification. Technical Report CCS-TR-99-157, NCCS, 1999.

[CCZ07] Bradford L. Chamberlain, David Callahan, and Hans P. Zima. Parallel Programmability and the Chapel Language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007. DOI 10.1177/1094342007078442.

[CD04] Joel Cracraft and Michael J. Donoghue, editors. *Assembling the Tree of Life*. Oxford University Press, 2004. ISBN 9780195172355.

[CDK05] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems – Concepts and Design*. Addison-Wesley, 4th edition, 2005. ISBN 9780321263544.

[CDM03] Stefano Ceri, Florian Daniel, and Maristella Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *4th International Conference on Web Information Systems Engineering (WISE)*, pages 225–233. IEEE, 2003. DOI 10.1109/WISEW.2003.1286806.

[CDO+96] Jaeyoung Choi, Jack J. Dongarra, L. Susan Ostrouchov, Antoine P. Petitet, David W. Walker, and R. Clint Whaley. The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines. *Scientific Programming*, 5(3):173–184, 1996.

[CFK+94] Mani Chandy, Ian T. Foster, Ken Kennedy, Charles Koelbel, and Chau-Wen Tseng. Integrated Support for Task and Data Parallelism . *International Journal of High Performance Computing Applications*, 8(2):80–98, 1994. DOI 10.1177/109434209400800202.

[CFP08] Srinivas Chellappa, Franz Franchetti, and Markus Püschel. How To Write Fast Numerical Code: A Small Introduction. In *Summer School on Generative and Transformational Techniques in Software Engineering*, LNCS. Springer, 2008. DOI 10.1007/978-3-540-88643-3_5.

[CGdMD05] Paulo C. Carvalho, Rafael V. Glória, Antonio B. de Miranda, and Wim M. Degrave. Squid – a simple bioinformatics grid. *BMC Bioinformatics*, 6(197), 2005. DOI 10.1186/1471-2105-6-197.

[CGS+05] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: An Object-Oriented Approach to Non-Uniform Cluster Computing. In *20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications (OOPSLA)*, pages 519–538. ACM, 2005. DOI 10.1145/1094811.1094852.

[Cip00] Barry A. Cipra. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News*, 33(4), 2000.

[CJvdP08] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. Scientific and Engineering Computation. MIT Press, 2008. ISBN 9780262533027.

[CKPN00] Junwei Cao, Darren J. Kerbyson, Efstathios Papaefstathiou, and Graham R. Nudd. Performance Modelling of Parallel and Distributed Computing Using PACE. In *19th International Performance, Computing, and Communications Conference (ICCC)*, pages 485–492. IEEE, 2000. DOI 10.1109/PCCC.2000.830354.

[CNYM00] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer, 2000. ISBN 9780792386667.

[Cou07] The National Science Foundation's Cyberinfrastructure Council. Cyberinfrastructure Vision for 21st Century Discovery. Technical report, National Science Foundation (NSF), 2007.

[Cro10] Benjamin Crowell. *Simple Nature – An Introduction to Physics for Engineering and Physical Science Students*. Fullerton, 2010. ISBN 9780970467072.

[CT05] Benny Chor and Tamir Tuller. Maximum Likelihood of Evolutionary Trees Is Hard. In *9th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, volume 3500 of *LNCS*, pages 296–310. Springer, 2005. DOI 10.1007/11415770_23.

[CUSD88] H. Y. Chang, S. Utku, M. Salama, and D. Drapp. A parallel Householder tridiagonalization stratagem using scattered square decomposition. *Parallel Computing*, 6(3):297–311, 1988. DOI 10.1016/0167-8191(88)90071-3.

[CW96] Jane K. Cullum and Ralph A. Willoughby. A QL Procedure for Computing the Eigenvalues of Complex Symmetric Tridiagonal Matrices. *SIAM Journal on Matrix Analysis and Applications*, 17(1):83–109, 1996. DOI 10.1137/S0895479894137639.

[CW02] Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Classics in Applied Mathematics. SIAM, 2002. ISBN 9780898715231.

[dAAL+04] Eduardo Santana de Almeida, Alexandre Alvaro, Daniel Lucredio, Vinicius Cardoso Garcia, and Silvio Romero de Lemos Meira. RiSE Project: Towards a Robust Framework for Software Reuse. In *International Conference on Information Reuse and Integration (IRI)*, pages 48–53. IEEE, 2004. DOI 10.1109/IRI.2004.1431435.

[Dav08] Erlend Davidson. *Message-passing for Lattice Boltzmann*. PhD thesis, University of Edinburgh, 2008.

[Day87] William H. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987. DOI 10.1016/S0092-8240(87)80007-1.

[DBC+06] J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar. Self-adapting numerical software (SANS) effort. *IBM Journal of Research and Development*, 50(2/3):223–238, 2006. DOI 10.1147/rd.502.0223.

[DCHD90] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *Transactions on Mathematical Software*, 16(1):1–17/18–28, 1990. DOI 10.1145/77626.79170.

[DCHH85] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. A proposal for an extended set of fortran basic linear algebra subprograms. *SIGNUM Newsletter*, 20(1):2–18, 1985. DOI 10.1145/1057935.1057936.

[DCHH88] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An Extended Set of FORTRAN Basic Linear Algebra Subprograms. *Transactions on Mathematical Software*, 14(1):1–17/18–32, 1988. DOI 10.1145/42288.42291.

[DDP+07] James W. Demmel, Jack Dongarra, Beresford Parlett, W. Kahan, Ming Gu, David Bindel, Yozo Hida, Xiaoye S. Li, Osni A. Marques, E. Jason Riedy, Christof Vömel, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, Julie Langou, and Stanimire Tomov. Prospectus for the Next LAPACK and ScaLAPACK Libraries. In *8th International Workshop on Applied Parallel Computing (PARA)*, volume 4699 of *LNCS*, pages 11–23. Springer, 2007. DOI 10.1007/978-3-540-75755-9_2.

[Dem91] James W. Demmel. LAPACK: A portable linear algebra library for high-performance computers. *Concurrency and Computation: Practice and Experience*, 3(6):655–666, 1991. DOI 10.1002/cpe.4330030610.

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997. ISBN 9780898713893.

[DER86] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Numerical Mathematics and Scientific Computation. Oxford University Press, 1986. ISBN 9780198534082.

[DFF+03] Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors. *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, 2003. ISBN 9781558608719.

[DGFK07] Jack J. Dongarra, Dennis Gannon, Geoffrey Fox, and Ken Kennedy. The Impact of Multicore on Computational Science Software. *CTWatch Quarterly*, 3(1):3–10, 2007.

[DGG+07] Jack J. Dongarra, Gene H. Golub, Eric Grosse, Cleve B. Moler, and Keith Moore. Netlib and NA-Net: building a scientific computing community. *IEEE Annals of the History of Computing*, 30(2):30–41, 2007. DOI 10.1109/MAHC.2008.29.

[DGH+08] J. Dongarra, R. Graybill, W. Harrod, R. Lucas, E. Lusk, P. Luszczek, J. McMahon, A. Snavely, J. Vetter, K. Yelick, S. Alam, R. Campbell, L. Carrington, T. Chen, O. Khalili, J. Meredith, and M. Tikir. DARPA's HPCS Program: History, Models, Tools, Languages. In M. Zelkowitz, editor, *Advances in Computers*, volume 72 of *High Performance Computing*. Elsevier, 2008.

[Dhi97] Inderjit S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, University of California, Berkeley, 1997.

[Dik07] Marios D. Dikaiakos. Grid benchmarking: vision, challenges, and current status. *Concurrency and Computation: Practice and Experience*, 19(1):89–105, 2007. DOI 10.1002/cpe.1086.

[DJS86] William H. E. Day, David S. Johnson, and David Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81(1):33–42, 1986. DOI 10.1016/0025-5564(86)90161-6.

[DL06] Jack Dongarra and Alexey Lastovetsky. A Survey of Heterogeneous High Performance and Grid Computing. In Beniamino Di Martino, Jack Dongarra, Adolfy Hoisie, Laurence Tianruo Yang, and Hans Zima, editors, *Engineering The Grid: Status and Perspective*. American Scientific Publishers, 2006.

[DLP03] Jack J. Dongarra, Piotr Luszczek, and Antoine Petite. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003. DOI 10.1002/cpe.728.

[DM84] Jack J. Dongarra and Cleve B. Moler. *EISPACK – A Package for Solving Matrix Eigenvalue Problems*, chapter 4, pages 68–87. Prentice-Hall, 1984.

[Don91] Jack J. Dongarra. Workshop on the BLACS. Technical Report UT-CS-91-134, UTK, 1991. LAWN 34.

[DS84] Jack J. Dongarra and G. W. Stewart. *LINPACK – A Package for Solving Linear Systems*, chapter 2, pages 20–48. Prentice-Hall, 1984.

[DS86] William H. E. Day and David Sankoff. Computational Complexity of Inferring Phylogenies by Compatibility. *System Zoology*, 35(2):224–229, 1986. DOI 10.2307/2413432.

[DS00] Jack J. Dongarra and Francis Sullivan. Guest Editors' Introduction to the Top 10 Algorithms. *Computing in Science and Engineering*, 2(1):22–23, 2000. DOI 10.1109/MCISE.2000.814652.

[DSSS05] Jack J. Dongarra, Thomas Sterling, Horst Simon, and Erich Strohmaier. High-Performance Computing: Clusters, Constellations, MPPs, and Future Directions. *Computing in Science and Engineering*, 7(2):51–59, 2005. DOI 10.1109/MCSE.2005.34.

[dV94] Eric F. Van de Velde. *Concurrent Scientific Computing*. Springer, 1994. ISBN 9780387941950.

[DvdGW92] Jack J. Dongarra, Robert A. van de Geijn, and David Walker. A Look at Scalable Dense Linear Algebra Libraries. In *Scalable High Performance Computing Conference (SHPCC)*, pages 372–379. IEEE, 1992. DOI 10.1109/SHPCC.1992.232670.

[DW93] Jack J. Dongarra and R. Clint Whaley. A User's Guide to the BLACS. Technical Report UT-CS-95-281, UTK, 1993. LAWN 94.

[DW95] Jack J. Dongarra and David W. Walker. Software Libraries for Linear Algebra Computations on High Performance Computers. *SIAM Review*, 37(2):151–180, 1995. DOI 10.1137/1037042.

[EBH08] John W. Eaton, David Bateman, and Søren Hauberg. *GNU Octave Manual*, 3$^{\text{3rd}}$ edition, 2008. ISBN 9780954612061.

[Efr79] Bradley Efron. Bootstrap Methods: Another Look at the Jackknife. *Annals of Statistics*, 7(1):1–26, 1979. DOI 10.1214/aos/1176344552.

[EGKU99] Harald J. Ehold, Wilfried N. Gansterer, Dieter F. Kvasnicka, and Christoph W. Überhuber. HPF and Numerical Libraries. In *4th International ACPC Conference*, LNCS, pages 140–152. Springer, 1999. DOI 10.1007/3-540-49164-3_14.

[EHH96] Bradley Efron, Elizabeth Halloran, and Susan Holmes. Bootstrap confidence levels for phylogenetic trees. *Proceedings of the National Academy of Sciences (PNAS) of the United States of America*, 93:13429–13434, 1996. PMID 8917608.

[FC96] Olivier P. Franza and Weng C. Chew. Recursive Mode Matching Method for Multiple Waveguide Junction Modeling. *IEEE Transactions on Microwave Theory and Techniques*, 44(1):87–92, 1996. DOI 10.1109/22.481389.

[FCS+94] Jerome Faist, Federico Capasso, Deborah L. Sivco, Carlo Sirtori, Albert L. Hutchinson, and Alfred Y. Cho. Quantum Cascade Laser. *Science*, 264(5158):553–556, 1994. DOI 10.1126/science.264.5158.553.

[Fel85] Joseph Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4):783–791, 1985. DOI 10.2307/2408678.

[Fel89] Joseph Felsenstein. PHYLIP – Phylogeny Inference Package. *Cladistics*, 5:164–166, 1989.

[FG82] Leslie R. Foulds and Ronald L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982. DOI 10.1016/S0196-8858(82)80004-3.

[FJ98] Matteo Frigo and Steven G. Johnson. FFTW: An Adaptive Software Architecture for the FFT. In *International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998. DOI 10.1109/ICASSP.1998.681704.

[FK97] Ian T. Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. DOI 10.1177/109434209701100205.

[FK01] Ian T. Foster and Carl Kesselman. Computational Grids. In *4th International Conference on Vector and Parallel Processing (VECPAR)*, volume 1981 of *LNCS*, pages 3–37. Springer, 2001. DOI 10.1007/3-540-44942-6_2.

[FMM00] Michael C. Ferris, Michael P. Mesnier, and Jorge J. Moré. NEOS and Condor: Solving Optimization Problems Over the Internet. *ACM Transactions on Mathematical Software*, 26(1):1–18, 2000. DOI 10.1145/347837.347842.

[For97] High Performance Fortran Forum. High Performance Fortran Language Specification Version 2.0. Technical Report CRPC-TR92225, Rice University, 1997.

[For03] Fortran 2003 Final Committee Draft. Technical Report FCD 1539-1, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), SC 22, Oct 2003.

[Fos06] Ian T. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006. DOI 10.1007/s11390-006-0513-y.

[FPB07] Norman Finger, Christoph Pacher, and Winfried Boxleitner. Simulation of Guided-Wave Photonic Devices with Variational Mode-Matching. In W. Jantsch and F. Schäffler, editors, *28th International Conference on the Physics of Semiconductors (ICPS)*, volume 893, pages 1493–1494. American Institute of Physics, 2007. DOI 10.1063/1.2730473.

[Fre07] Free Software Foundation (FSF). *Using GNU Fortran*, 2007.

[Gan06] Wilfried N. Gansterer. *Context Adaptive Algorithms in Scientific Computing*. 2006.

[Gat00] Kang Su Gatlin. *Portable High Performance Programming via Architecture-Cognizant Divide-and-Conquer Algorithms*. PhD thesis, University of California, San Diego, 2000.

[GD89] A. Greenbaum and Jack J. Dongarra. Experiments with QR/QL Methods for the Symmetric Tridiagonal Eigenproblem. Technical Report UT-CS-89-92, UTK, 1989. LAWN 17.

[GDS+06] Francois Gygi, Erik W. Draeger, Martin Schulz, Bronis R. de Supinski, John A. Gunnels, Vernon Austel, James C. Sexton, Franz Franchetti, Stefan Kral, Christoph W. Überhuber, and Jürgen Lorenz. Large-scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform. In *Supercomputing (SC)*. ACM/IEEE, 2006. DOI 10.1145/1188455.1188502.

[Gen02] Wofgang Gentzsch. Grid Computing, A Vendor's Vision. In *2nd International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 290–295. IEEE/ACM, 2002. DOI 10.1109/CCGRID.2002.1017148.

[GG08] Kazushige Goto and Robert Van De Geijn. High-Performance Implementation of the Level-3 BLAS. *Transactions on Mathematical Software*, 35(1):1–14, 2008. DOI 10.1145/1377603.1377607.

[GGK03] Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing*. Addison Wesley, 2nd edition, 2003. ISBN 9780201648652.

[GGP09] Wilfried N. Gansterer, Andreas R. Gruber, and Christoph Pacher. Non-Splitting Tridiagonalization of Complex Symmetric Matrices. In *International Conference of Computational Science (ICCS)*, volume 5544 of *LNCS*, pages 481–490. Springer, 2009. DOI 10.1007/978-3-642-01970-8_47.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, 1979. ISBN 9780716710455.

[GJRB02] Fabrizio Gagliardi, Bob Jones, Mario Reale, and Stephen Burke. European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications. In *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *LNCS*, pages 255–264. Springer, 2002. DOI 10.1007/3-540-45798-4_20.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996. ISBN 9780801854132.

[GSPF08] Wilfried N. Gansterer, Hannes Schabauer, Christoph Pacher, and Norman Finger. Tridiagonalizing Complex Symmetric Matrices in Waveguide Simulations. In M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *8th International Conference on Computational Science (ICCS)*, volume 5101 of *LNCS*, pages 945–954. Springer, 2008. DOI 10.1007/978-3-540-69384-0_99.

[Gus97] John Gustafson. The Program of Grand Challenge Problems: Expectations and Results. In *2nd International Symposium on Parallel Algorithms / Architecture Synthesis (PAS)*, pages 2–7. IEEE, 1997. DOI 10.1109/AISPAS.1997.581619.

[GvdGKQO01] John A. Gunnels, Robert A. van de Geijn, Daniel S. Katz, and Enrique S. Quintana-Ortí. Fault-Tolerant High-Performance Matrix Multiplication: Theory and Practice. In *International Conference on Dependable Systems and Networks (DSN)*, pages 47–56. IEEE, 2001. DOI 10.1109/DSN.2001.941390.

[GWMG04] Galen R. Gisler, Robert P. Weaver, Charles L. Mader, and Michael L. Gittings. Two- and Three-Dimensional Asteroid Impact Simulations. *Computing in Science and Engineering*, 6(3):46–55, 2004. DOI 10.1109/MCSE.2004.55.

[HA91] Joyce M. Hawkins and Robert Allen, editors. *The Oxford encyclopedic English dictionary*. Oxford University Press, 1991. ISBN 9780198612667.

[Hea97] Michael T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, 1997. ISBN 9780070276840.

[HGS+08] Helmut Hlavacs, Wilfried Gansterer, Hannes Schabauer, Joachim Zottl, Martin Petraschek, Thomas Hoeher, and Oliver Jung. Enhancing ZRTP by using Computational Puzzles. *Journal of Universal Computer Science (J.UCS)*, 14(5):693–716, 2008. DOI 10.3217/jucs-014-05-0693.

[HHL07] Sven Hammarling, Nicholas J. Higham, and Craig Lucas. LAPACK-Style Codes for Pivoted Cholesky and QR Updating. In *8th International Workshop on Applied Parallel Computing (PARA) – State of the Art in Scientific Computing*, volume 4699 of *LNCS*, pages 137–146, 2007. DOI 10.1007/978-3-540-75755-9_17.

[Hig96] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996. ISBN 9780898713558.

[HJ85] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. ISBN 9780521305860.

[HJ91] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991. ISBN 9780521305877.

[HM05] Tony Hoare and Robin Milner. Grand Challenges for Computing Research. *The Computer Journal*, 48(1):49–52, 2005. DOI 10.1093/comjnl/bxh065.

[Hou58] Alston S. Householder. Unitary Triangularization of a Nonsymmetric Matrix. *Journal of the ACM*, 5(4):339–342, 1958. DOI 10.1145/320941.320947.

[HP07] John L. Hennessy and David A. Patterson. *Computer Architecture – A Quantitative Approach*. Morgan Kaufmann Publishers, 4th edition, 2007. ISBN 9780123735904.

[HRGB07] E. N. Houstis, J. R. Rice, E. Gallopoulos, and R. Bramley, editors. *Enabling Technologies for Computational Science – Frameworks, Middleware and Environments*, volume 548 of *International Series in Engineering and Computer Science*. Springer, 2nd edition, 2007. ISBN 9780792378099.

[HRTV07] Vicente Hernández, José E. Román, Andrés Tomás, and Vicente Vidal. *SLEPc Users Manual*. Universidad Politécnica de Valencia, 2007. Technical Report DSIC-II/24/02.

[HRV05] Vicente Hernández, José E. Román, and Vicente Vidal. SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, 2005. DOI 10.1145/1089014.1089019.

[IEE85] IEEE Standard for Binary Floating-Point Arithmetic. American National Standard 754–1985, ANSI / IEEE, 1985. DOI 10.1109/IEEESTD.1985.82928.

[Ips97] Ilse C. F. Ipsen. Computing an Eigenvector with Inverse Iteration. *SIAM Review*, 39(2):254–291, 1997. DOI 10.1137/S0036144596300773.

[Jac99] John D. Jackson. *Classical Electrodynamics*. John Wiley & Sons, 3rd edition, 1999. ISBN 9780471309321.

[JCC+04] N. Jacq, C. Blanchet C. Combet, E. Cornillot, L. Duret, K. Kurata, H. Nakamura, T. Silvestre, and V. Breton. Grid as bioinformatic tool. *Parallel Computing*, 30(9-10):1093–1107, 2004. DOI 10.1016/j.parco.2004.07.013.

[Jin05] Hai Jin. ChinaGrid: Making Grid Computing a Reality. In Zhaoneng Chen, Hsinchun Chen, Qihao Miao, Yuxi Fu, Edward Fox, and Ee peng Lim, editors, *7th International Conference on Asian Digital Libraries (ICADL)*, volume 3334 of *LNCS*, pages 13–24, 2005. DOI 10.1007/b104284.

[JP94] Mark T. Jones and Merrell L. Patrick. Factoring Symmetric Indefinite Matrices on High-Performance Architectures. *SIAM Journal on Matrix Analysis and Applications*, 15(1):273–283, 1994. DOI 10.1137/S089547989018008X.

[JWG+04] A. Jones, R. White, W. Gray, F. Bisby, N. Caithness, N. Pittas, X. Xu, T. Sutton, N. Fiddian, A. Culham, M. Scoble, P. Williams, O. Bromley, P. Brewer, C. Yesson, and S. Bhagwat. Building a Biodiversity GRID. In *1st International Workshop on Life Science Grid (LSGRID)*, volume 3370 of *LNBI*, pages 140–151. Springer, 2004. DOI 10.1007/b106923.

[KBD08] Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization. *IEEE Transactions on Parallel and Distributed Systems*, 19(9):1175–1186, 2008. DOI 10.1109/TPDS.2007.70813.

[KK01] Kenji Kawano and Tsutomu Kitho. *Introduction to Optical Waveguide Analysis – Solving Maxwell's Equations and the Schrödinger Equation.* Wiley-Interscience, 2001. ISBN 9780471406341.

[Kle85] Leonard Kleinrock. Distributed Systems. *Communications of the ACM, special issue on computing in the frontiers of science and engineering*, 28(11):1200–1213, 1985. DOI 10.1145/4547.4552.

[KLS+94] Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook.* Scientific and Engineering Computation. MIT Press, 1994. ISBN 9780262610940.

[Knu98] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley, 2nd edition, 1998. ISBN 9780201896855.

[KU93] Arnold R. Krommer and Christoph W. Überhuber. Architecture Adaptive Algorithms. *Parallel Computing*, 19(4):409–435, 1993. DOI 10.1016/0167-8191(93)90055-P.

[Las06] Alexey Lastovetsky. Scientific Programming for Heterogeneous Systems – Bridging the Gap between Algorithms and Applications. In *International Symposium on Parallel Computing in Electrical Engineering (PARELEC)*, pages 3–8. IEEE, 2006. DOI 10.1109/PARELEC.2006.72.

[Leu95] Andrew Y. T. Leung. Subspace Iteration for Complex Symmetric Eigenproblems. *Journal of Sound and Vibration*, 184(4):627–637, 1995. DOI 10.1006/jsvi.1995.0337.

[Lev89] Eugene Levin. Grand Challenges to Computational Science. *Communications of the ACM*, 32(12):1456–1457, 1989. DOI 10.1145/76380.76385.

[LFF+06] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, and A. Edlund. Programming The Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[LHA+04] E. Laure, F. Hemmer, A. Aimar, M. Barroso, P. Buncic, A. Di Meglio, L. Guy, P. Kunszt, S. Beco, F. Pacini, F. Prelz, M. Sgaravetto, A. Edlund, O. Mulmo, D. Groep, S. M. Fisher, and M. Livny. Middleware for the next Generation Grid Infrastructure. In *Computing in High Energy Physics Conference (CHEP)*, 2004.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979. DOI 10.1145/355841.355847.

[LJ93] Woody Lichtenstein and S. Lennart Johnsson. Block-Cyclic Dense Linear Algebra. *SIAM Journal on Scientific Computing*, 14(6):1259–1288, 1993. DOI 10.1137/0914075.

[LKJ03] Glenn R. Luecke, Marina Kraeva, and Lili Ju. Comparing the performance of MPICH with Cray's MPI and with SGI's MPI. *Concurrency and Computation: Practice and Experience*, 15(9):779–802, 2003. DOI 10.1002/cpe.719.

[LL92] Andrew Y. T. Leung and Yan-Fang Liu. A generalized complex symmetric eigensolver. *Computers and Structures*, 43(6):1183–1186, 1992. DOI 10.1016/0045-7949(92)90018-U.

[LLM88] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor – A Hunter of Idle Workstations. In *8th International Conference of Distributed Computing Systems*, pages 104–111. IEEE, 1988. DOI 10.1109/DCS.1988.12507.

[Loa96] Charles F. Van Loan. *An Introduction to Computational Science and Mathematics*. Jones and Bartlett, 1996. ISBN 9780867204735.

[Lov93] David B. Loveman. High Performance Fortran. *Parallel and Distributed Technology: Systems and Applications*, 1(1):25–42, 1993. DOI 10.1109/88.219857.

[LQ97] Franklin T. Luk and Sanzheng Qiao. Using Complex-Orthogonal Transformations to Diagonalize a Complex Symmetric Matrix. In Franklin T. Luk, editor, *Advanced Signal Processing: Algorithms, Architectures, and Implementations VII*, volume 3162, pages 418–425. SPIE, 1997. DOI 10.1117/12.284193.

[LSY98] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998. ISBN 9780898714074.

[Luc04] Craig Lucas. LAPACK-Style Codes for Level 2 and 3 Pivoted Cholesky Factorizations. Numerical Analysis Report 442, Manchester Centre for Computational Mathematics, University of Manchester, 2004. LAWN 161.

[Man08] Svetlin A. Manavski. Cuda Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography. In *International Conference on Signal Processing and Communications (ICSPC)*, pages 1175–1186. IEEE, 2008. DOI 10.1109/TPDS.2007.70813.

[McE97] Hugh McEvoy. Context sensitivity and synchronisation as taxonomics for parallel languages. In *30th International Conference on System Sciences (HICSS) (1): Software Technology and Architecture*, pages 369–378. IEEE, 1997. DOI 10.1109/HICSS.1997.667284.

[Meu08] Hans W. Meuer. The TOP500 Project: Looking Back Over 15 Years of Supercomputing Experience. *Informatik-Spektrum*, 31(3):203–222, 2008. DOI 10.1007/s00287-008-0240-6.

[MFS94] Carlos R. Mechoso, John D. Farrara, and Joseph A. Spahr. Achieving Superlinear Speedup on a Heterogeneous, Distributed System. *IEEE Concurrency*, 2(2):57–61, 1994. DOI 10.1109/88.311573.

[MMMP04] Stefano Modafferi, Enrico Mussi, Andrea Maurino, and Barbara Pernici. A Framework for Provisioning of Complex e-Services. In *International Conference on Services Computing (SCC)*, pages 81–90. IEEE, 2004. DOI 10.1109/SCC.2004.1357993.

[MNW+04] Bernard M.E.Moret, Luay Nakhleh, Tandy Warnow, C. Randal Linder, Anna Tholse, Anneke Padolina, Jerry Sun, and Ruth Timme. Phylogenetic Networks: Modeling, Reconstructibility, and Accuracy. *IEEE Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004. DOI 10.1109/TCBB.2004.10.

[Moi98] Nimrod Moiseyev. Quantum theory of resonances: calculating energies, widths and cross-sections by complex scaling. *Physics Reports*, 302(5–6):212–293, 1998. 10.1016/S0370-1573(98)00002-7.

[Mol04] Cleve B. Moler. *Numerical Computing with Matlab*. SIAM, 2004. ISBN 9780898715606.

[MRZ98] Piyush Mehrotra, John Van Rosendale, and Hans Zima. High Performance Fortran: History, status and future. *Parallel Computing*, 24(3–4):325–354, 1998. DOI 10.1016/S0167-8191(98)00016-7.

[MS73] Cleve B. Moler and G. W. Stewart. An Algorithm for Generalized Matrix Eigenvalue Problems. *SIAM Journal on Numerical Analysis*, 10(2):241–256, 1973. DOI 10.1137/0710024.

[MSM07] David R. Maddison, Katja-Sabine Schulz, and Wayne P. Maddison. The Tree of Life Web Project. *Zootaxa*, 1668:19–40, 2007.

[MVHS05] Bui Quang Minh, Le Sy Vinh, Arndt Von Haeseler, and Heiko A. Schmidt. pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796, 2005. DOI 10.1093/bioinformatics/bti594.

[MVSvH06] Bui Quang Minh, Le Sy Vinh, Heiko A. Schmidt, and Arndt von Haeseler. Large Maximum Likelihood Trees. In Gernot Münster, Dietrich Wolf, and Manfred Kremer, editors, *NIC Symposium*, volume 32 of *NIC*, pages 357–366. John von Neumann Institut for Computing, 2006.

[MW68] R. S. Martin and J. H. Wilkinson. Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form. *Numerische Mathematik*, 11(2):99–110, 1968. DOI 10.1007/BF02165306.

[NK00] Masatoshi Nei and Sudhir Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, 2000. ISBN 9780195135855.

[NR98] Robert W. Numrich and John Reid. Co-array Fortran for parallel programming. *ACM SIGPLAN Fortran Forum*, 17(2):1–31, 1998. DOI 10.1145/289918.289920.

[NS03] Nicholas Nethercote and Julian Seward. Valgrind: A Program Supervision Framework. *Electronic Notes in Theoretical Computer Science*, 89(2):44–66, 2003. DOI 10.1016/S1571-0661(04)81042-9.

[NS07] Nicholas Nethercote and Julian Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. *ACM SIGPLAN Notices*, 42(6):89–100, 2007. DOI 10.1145/1273442.1250746.

[OBF+06] J. T. Oden, T. Belytschko, J. Fish, T. J. R. Hughes, C. Johnson, D. Keyes, A. Laub, L. Petzold, D. Srolovitz, and S. Yip. Simulation Based Engineering Science – Revolutionizing Engineering Science through Simulation. Report of the findings and recommendations on simulation-based engineering science, National Science Foundation (NSF), May 2006.

[OM92] Kazumoto Ohnami and Yasushi Mikami. Resonance Scattering in a Two-Dimensional Non-integrable System. *Journal of Physics A*, 25(18):4903–4912, 1992. DOI 10.1088/0305-4470/25/18/022.

[Pag03] Roderic D. M. Page. Introduction to Inferring Evolutionary Relationships. In Andreas D. Baxevanis, Daniel B. Davison, Roderic D. M. Page, Gary Stormo, and Lincoln Stein, editors, *Current Protocols in Bioinformatics*, pages 6.1.1–6.1.13. John Wiley & Sons, supplement 3 edition, 2003. ISBN 9780471250937.

[Par64] Beresford N. Parlett. A Note on La Budde's Algorithm. *Mathematics of Computation*, 18(87):505–506, 1964. DOI 10.2307/2003776.

[Par80] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, 1980. ISBN 9780138800475.

[Par98] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics. SIAM, corrected reprinted edition, 1998. ISBN 9780898714029.

[PKD97] James S. Plank, Youngbae Kim, and Jack J. Dongarra. Fault-tolerant matrix operations for networks of workstations using diskless checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, 1997. DOI 10.1006/jpdc.1997.1336.

[PMJ+05] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo. SPIRAL: Code Generation for DSP Transforms. *Proceedings of the IEEE, special issue on program generation, optimization, and adaptation*, 93(2):232–275, 2005. DOI 10.1109/JPROC.2004.840306.

[Pre01] Roger S. Pressman. *Software Engineering – A Practitioner's Approach*. McGraw-Hill, 5th edition, 2001. ISBN 9780073655789.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes – The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007. ISBN 9780521880688.

[PV05] Douglass E. Post and Lawrence G. Votta. Computational Science Demands a New Paradigm. *Physics Today*, 58:35–41, 2005. DOI 10.1063/1.1881898.

[PW70] G. Peters and James H. Wilkinson. $Ax = \lambda Bx$ and the Generalized Eigenproblem. *SIAM Journal on Numerical Analysis*, 7(4):479–492, 1970. DOI 10.1137/0707039.

[PW79] G. Peters and James H. Wilkinson. Inverse Iteration, Ill-Conditioned Equations and Newton's Method. *SIAM Review*, 21(3):339–360, 1979. DOI 10.1137/1021052.

[PZA86] Ronald H. Perrott and Adib Zarea-Aliabadi. Supercomputer languages. *ACM Computing Surveys*, 18(1):5–22, 1986. DOI 10.1145/6462.6463.

[RB96] John R. Rice and Ronald F. Boisvert. From Scientific Software Libraries to Problem-Solving Environments. *Computational Science and Engineering*, 3(3):44–53, 1996. DOI 10.1109/99.537091.

[Rei82] W. P. Reinhardt. Complex Coordinates in the Theory of Atomic and Molecular Structure and Dynamics. *Annual Review of Physical Chemistry*, 33(1):223–255, 1982. DOI 10.1146/annurev.pc.33.100182.001255.

[Ric95] John R. Rice. Computational Science and the Future of Computing Research. *Computational Science and Engineering*, 2(4):35–41, 1995. DOI 10.1109/99.476366.

[RM93] U. V. Riss and H.-D. Meyer. Calculation of resonance energies and widths using the complex absorbing potential method. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 26(23):4503–4535, 1993. DOI 10.1088/0953-4075/26/23/021.

[RSW+08] M. Riedel, A. Streit, F. Wolf, T. Lippert, and D. Kranzlmüller. Classification of Different Approaches for e-Science Applications in Next Generation Computing Infrastructures. In *Proceedings of the 4th International Conference on eScience*, pages 198–205. IEEE, 2008. DOI 10.1109/eScience.2008.56.

[Saa92] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992. ISBN 9780470218204.

[Sch03] Heiko A. Schmidt. *Phylogenetic Trees from Large Datasets*. PhD thesis, Heinrich-Heine-Universität Düsseldorf, 2003.

[Sch04] Hannes Schabauer. Parallelisierung von Kohonen-Netzen auf Cluster-Architekturen. Master's thesis, University of Vienna and Vienna University of Technology, Vienna, Austria, 2004. http://aleph.onb.ac.at/ (number 1738262-C).

[Sea69] M. J. Seaton. Diagonalisation of complex symmetric matrices using a modified Jacobi method. *Computer Journal*, 12(2):156–157, 1969. DOI 10.1093/comjnl/12.2.156.

[Ser80] Steven M. Serbin. On Factoring a Class of Complex Symmetric Matrices Without Pivoting. *Mathematics of Computation*, 35(152):1231–1234, 1980.

[SG09] Dru Sepulveda and Sebastien Goasguen. The Deployment and Maintenance of a Condor-Based Campus Grid. In *Advances in Grid and Pervasive Computing*, number 5529 in LNCS, pages 165–176. Springer, 2009. DOI 10.1007/978-3-642-01671-4_16.

[SGSP10] Hannes Schabauer, Wilfried N. Gansterer, Andrew G. Sunderland, and Christoph Pacher. Parallel Solution of Generalized Complex Symmetric Eigenproblems – Evaluation on HPCx. In *Science and Supercomputing in Europe – Report 2009*, HPC Europa2. CINECA, 2010.

[SHP08] Hannes Schabauer, Ronald Hochreiter, and Georg Ch. Pflug. Parallelization of Pricing Path-Dependent Financial Instruments on Bounded Trinomial Lattices. In M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *8th International Conference on Computational Science (ICCS)*, volume 5102 of *LNCS*, pages 408–415. Springer, 2008. DOI 10.1007/978-3-540-69387-1_46.

[SIU08] The International System of Units (SI). Technical Report 330, National Institute of Standards and Technology, 2008.

[SKDO07] Sabine Schweizer, Jörg Kussmann, Bernd Doser, and Christian Ochsenfeld. Linear-Scaling Cholesky Decomposition. *Journal of Computational Chemistry*, 29(6):1004–1010, 2007. DOI 10.1002/jcc.20862.

[SL01] Peter E. Strazdins and John G. Lewis. An Efficient and Stable Method for Parallel Factorization of Dense Symmetric Indefinite Matrices. In *5th International Conference on High Performance Computing in the Asia-Pacific Region (HPC Asia), CD-ROM*. Griffith University, 2001.

[SLMW02] Alexandros P. Stamatakis, Thomas Ludwig, Harald Meier, and Marty J. Wolf. AxML: A Fast Program for Sequential and Parallel Phylogenetic Tree Calculations Based on the Maximum Likelihood Method. In *CSB Conference on Bioinformatics*, pages 21–28. IEEE, 2002. DOI 10.1109/CSB.2002.1039325.

[SMLK06] Sulev Sild, Uko Maran, Andre Lomaka, and Mati Karelson. Open Computing Grid for Molecular Science and Engineering. *Journal of Chemical Information and Modeling*, 46(3):953–959, 2006. DOI 10.1021/ci050354f.

[SMR+05] Sulev Sild, Uko Maran, Mathilde Romberg, Bernd Schuller, and Emilio Benfenati. OpenMolGRID: Using Automated Workflows in GRID Computing Environment. In *European Grid Conference (EGC) 2005*, volume 3470 of *LNCS*, pages 464–473, 2005. DOI 10.1007/b104284.

[SNP+03] Timothée Silvestre, Estelle Nugues, Guy Perrière, Manolo Gouy, and Laurent Duret. Phylojava: a Generic Client-Server Tool for Phylogenetic Tree Reconstruction – Application to Grid Computing. Poster contribution at European Conference on Computational Biology (ECCB), 2003.

[SO96] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Courier Dover Publications, 1996. ISBN 9780486691862.

[SO08] Alexandros Stamatakis and Michael Ott. Efficient computation of the phylogenetic likelihood function on multi-gene alignments and multicore architectures. *Philosophical Transactions of the Royal Society B*, 363(1512):3977–3984, 2008. DOI 10.1098/rstb.2008.0163.

[SOHL+98] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core of *Scientific and Engineering Computation*. MIT Press, 2nd edition, 1998. ISBN 9780262692151.

[SP88] Robert Schreiber and Beresford Parlett. Block reflectors: Theory and Computation. *SIAM Journal on Numerical Analysis*, 25(1):189–205, 1988. DOI 10.1137/0725014.

[SPSG10] Hannes Schabauer, Christoph Pacher, Andrew G. Sunderland, and Wilfried N. Gansterer. Toward a parallel solver for generalized complex symmetric eigenvalue problems. In *10^{th} International Conference on Computational Science (ICCS)*, Procedia Computer Science. Elsevier, 2010.

[SSB+95] Thomas L. Sterling, Daniel Savarese, Donald J. Becker, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. BEOWULF: A Parallel Workstation for Scientific Computation. In *24^{th} International Conference on Parallel Processing (ICPP)*, volume 1: Architecture, pages 11–14. CRC Press, 1995.

[SSB+08] B. De Supinski, M. Schulz, V. Bulatov, W. Cabot, B. Chan, A. Cook, E. Draeger, J. Glosli, J. Greenough, K. Henderson, A. Kubota, S. Louis, B. Miller, M. Patel, T. Spelce, F. Streitz, P. Williams, R. Yates, A. Yoo, G. Almasi, G. Bhanot, A. Gara, J. Gunnels, M. Gupta, J. Moreira, J. Sexton, B. Walkup, C. Archer, F. Gygi, T. Germann, K. Kadau, P. Lomdahl, C. Rendleman, M. Welcome, W. Mclendon, B. Hendrickson, F. Franchetti, S. Kral, J. Lorenz, C. Überhuber, E. Chow, and Ü. Çatalyürek. BlueGene/L Applications: Parallelism On a Massive Scale. *International Journal of High Performance Computing Applications*, 22(1):33–51, 2008. DOI 10.1177/1094342007085025.

[SSG+98] Joel Saltz, Alan Sussman, Susan Graham, James Demmel, Scott Baden, and Jack Dongarra. Programming Tools and Environments. *Communications of the ACM*, 41(11):64–73, 1998. DOI 10.1145/287831.287841.

[SSHG93] Jaspal Subhlok, James M. Stichnoth, David R. O' Hallaron, and Thomas Gross. Exploiting Task and Data Parallelism on a Multicomputer. *ACM SIGPLAN Notices*, 28(7):13–22, 1993. DOI 10.1145/173284.155334.

[SSVvH02] Heiko A. Schmidt, Korbinian Strimmer, Martin Vingron, and Arndt von Haeseler. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3):502–504, 2002. DOI 10.1093/bioinformatics/18.3.502.

[SSW05] Hannes Schabauer, Erich Schikuta, and Thomas Weishäupl. Solving Very Large Traveling Salesman Problems by SOM Parallelization on Cluster Architectures. In H. Shen and K. Nakano, editors, *6^{th} International Conference on Parallel and Distributed Computing, Applications, and Technologies (PDCAT)*, pages 954–958. IEEE, 2005. DOI 10.1109/PDCAT.2005.223.

[Ste94] Daniel E. Stevenson. Science, Computational Science, and Computer Science: At a Crossroads. *Communications of the ACM*, 37(12):85–96, 1994. DOI 10.1145/198366.198386.

[Str00] Peter E. Strazdins. Accelerated methods for performing the $LDL^{\top}$ decomposition. *Australian & New Zealand Industrial and Applied Mathematics (ANZIAM) Journal*, 42:1328–1355, 2000.

[Sud93] Aasmund S. Sudbø. Film mode matching: a versatile numerical method for vector mode field calculations in dielectric waveguides. *Pure and Applied Optics*, 2(3):211–233, 1993. DOI 10.1088/0963-9659/2/3/007.

[SV01] Hannes Schabauer and Martin Voracek. LetterScan – Ein PC-Programm zur optimalen Selektion von Buchstabenkombinationen als Items oder Experimentalstimuli. poster contribution at 42nd Kongreß der Deutschen Gesellschaft für Psychologie, 2001. ISBN 9783935357333.

[SZvH+08] Hannes Schabauer, Joachim Zottl, Arndt von Haeseler, Wilfried N. Gansterer, and Heiko A. Schmidt. Phylogenetic Quality Assessment for Campus Grids. In J. Küng, K. Schneider, and R. Wagner, editors, *2nd International Conference on Bioinformatics Research and Development (BIRD) – Poster Presentations*, number 26 in Schriftenreihe Informatik, pages 109–114. Trauner Verlag, 2008. ISBN 9783854994220.

[TB97] LLoyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997. ISBN 9780120887842.

[TC98] Fernando L. Teixeira and Weng C. Chew. General Closed-Form PML Constitutive Tensors to Match Arbitrary Bianisotropic and Dispersive Linear Media. *IEEE Microwave Guided Wave Letters*, 8(6):223–225, 1998. DOI 10.1109/75.678571.

[Thi08] Patrick Thibodeau. IBM breaks petaflop barrier. *InfoWorld*, June 2008.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience, special issue on grid performance*, 17(2–4):323–356, 2005. DOI 10.1002/cpe.938.

[TTL06] Douglas Thain, Todd Tannenbaum, and Miron Livny. How to measure a large open-source distributed system. *Concurrency and Computation: Practice and Experience*, 18(15):1989–2019, 2006. DOI 10.1002/cpe.1041.

[TWN04] Yong-Meng Teo, Xianbing Wang, and Yew-Kwong Ng. GLAD: a system for developing and deploying large-scale bioinformatics grid. *Bioinformatics*, 21(6), 2004. DOI 10.1093/bioinformatics/bti034.

[Ü97a] Christoph W. Überhuber. *Numerical computation 1 – Methods, Software, and Analysis*. Springer, 1997. ISBN 9783540620587.

[Ü97b] Christoph W. Überhuber. *Numerical computation 2 – Methods, Software, and Analysis*. Springer, 1997. ISBN 9783540620570.

[VD00] Richard Vuduc and James Demmel. Code Generators for Automatic Tuning of Numerical Kernels: Experiences with FFTW. In *International Workshop on Semantics, Applications, and Implementation of Program Generation (SAIG)*, volume 1924 of *LNCS*, pages 190–211. Springer, 2000. DOI 10.1007/3-540-45350-4_14.

[VKC98] John L. Volakis, Leo C. Kempel, and A. Chatterjee. *Finite Element Method for Electromagnetics: Antennas, Microwave Circuits, and Scattering Applications*. Electromagnetic Wave Theory. IEEE, 1998. ISBN 9780780334250.

[VT98] Divakar Viswanath and LLoyd N. Trefethen. Condition Numbers of Random Triangular Matrices. *SIAM Journal on Matrix Analysis and Applications*, 19(2):564–581, 1998. DOI 10.1137/S0895479896312869.

[VTS+07] Martin Voracek, Ulrich S. Tran, Hannes Schabauer, Georg Koenne, and Barbara Glössl. On the elusive nature of the letters from the heart effect. *Perceptual and Motor Skills*, 104(3):803–814, 2007. DOI 10.2466/PMS.104.3.803-814,PMID 17688137.

[VvH04] Le Sy Vinh and Arndt von Haeseler. IQPNNI: Moving Fast Through Tree Space and Stopping in Time. *Molecular Biology and Evolution*, 21(8):1565–1571, 2004. DOI 10.1093/molbev/msh176.

[Vö03] Christof Vömel. *Contributions to research in high performance scientific computing for sparse matrices*. PhD thesis, CERFACS, 2003.

[Wat08] David S. Watkins. The QR Algorithm Revisited. *SIAM Review*, 50(1):133–145, 2008. DOI 10.1137/060659454.

[Wei03] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. Chapman & Hall / CRC, 2nd edition, 2003. ISBN 9781584883470.

[Wha04] R. Clint Whaley. *Automated Empirical Optimization of High Performance Floating Point Kernels*. PhD thesis, Florida State University, 2004.

[Wha08] R. Clint Whaley. Empirically Tuning LAPACK's Blocking Factor for Increased Performance. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 303–310. IEEE, 2008. DOI 10.1109/IMCSIT.2008.4747256.

[Wil88] Kenneth G. Wilson. Grand Challenges to Computational Science. In William Fickinger and Kenneth L. Kowalski, editors, *Michelson-Morley Centennial Symposium*, volume 169, pages 158–168. American Institute of Physics, 1988. DOI 10.1063/1.37199.

[Wil89] Kenneth G. Wilson. Grand Challenges to Computational Science. *Future Generation Computer Systems*, 5(2-3):171–189, 1989. DOI 10.1016/0167-739X(89)90038-1.

[WKW90] Carl R. Woese, Otto Kandler, and Mark L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences (PNAS) of the United States of America*, 87(12):4576–4579, 1990. PMID 2112744.

[WL91] Michael E. Wolf and Monica S. Lam. A Data Locality Optimizing Algorithm. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 30–44. ACM, 1991. DOI 10.1145/113446.113449.

[WPD01] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1–2):3–35, 2001. DOI 10.1016/S0167-8191(00)00087-9.

[WSBL03] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003. DOI 10.1109/TCSVT.2003.815165.

[WSCBF91] E. O. Wiley, D. Siegel-Causey, D. R. Brooks, and V. A. Funk. *The Compleat Cladist – A Primer of Phylogenetic Procedures*. University of Kansas, 1991. ISBN 9780893380359.

[YSP+98] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: a high-performance Java dialect. *Concurrency and Computation: Practice and Experience*, 10(11–13):825–836, 1998.

[ZAB+99] F. Zepparelli, F. Alimenti, P. Bassi, P. Mezzanotte, L. Roselli, and R. Sorrentino. FDTD-Analysis of 3D-Optical and Optoelectronic Waveguide-Based Devices. In *International Microwave Symposium Digest*, volume 3, pages 1257–1260. IEEE, 1999. DOI 10.1109/MWSYM.1999.779615.

[ZKK04] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kalé. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 78–87. IEEE, 2004. DOI 10.1109/IPDPS.2004.1303013.