Solving Symmetric Eigenvalue Problem using LAPACK and ScaLAPACK routines and Performance comparison

MSc Dissertation(COMM002)

URN: 6106000

ASLI EYECIOGLU

Supervised by: JOHANN BRIFFA

Submitted for the Degree of

Master of Science in Internet Computing from

Department of Computing

Faculty of engineering and Physical Sciences

University of Surrey

December 2010

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end

William Shakespeare

ACKNOWLEDGEMENT

I am grateful to my supervisor, Johann Briffa. He guide during my dissertation process and also encourage me how to think as a programmer.

I am also grateful to Fahad. He is my friend and my first teacher in IT, who taught me everything from scratch.

My friends are always in the core of my world that my world would stop turning if they were not with me.

Sure, I will not forget my family, because they have created my small world and made it meaningful.

Thanks

ABSTRACT

The debate on solving the symmetric eigenvalue problem continues in this piece of work. The approach of the solving symmetric eigenvalue problem with LAPACK routines SSYEV and DSYEV and its corresponding ScaLAPACK routines PSSYEV and PDSYEV, respectively, are discussed the relation between execution time while the research on going to find eigenvalues and eigenvectors of a symmetric matrix with QR algorithm. Fundamental linear algebra packages BLAS (with reference to ATLAS) and Parallel BLAS (PBLAS) are used to build linear algebra packages LAPACK and ScaLAPACK. The execution time of these specified single and double precision routines are examined between sequential and parallel implementation and performance issues are are discussed in comparable manner with respect to their results. To obtain best performance the ScaLAPACK environment exploits the parallelism on distributed shared memory with the help of the components ; MPI (Message Passing Interface) while BLACS (Basic Linear Algebra Communication Subroutines) is applied as communicator.

Key Words: *The Symmetric Eigenvalue Problem, LAPACK, ScaLAPACK, Execution Time*

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ABBREVIATION

BLAS	Basic Linear Algebra Sub-Programs
PBLAS	Parallel Basic Linear Algebra Sub-Programs
BLACS	Basic Linear Algebra Communication Sub-Programs
LAPACK	Linear Algebra Packages
ScaLAPACK	Saleable Linear Algebra packages
MPI	Message Passing Interface
SSYEV	Serial Symmetric Simple Drivers Routine
DSYEV	Double Symmetric Simple Drivers Routine
PSSYEV	Parallel Single Symmetric Simple Drivers Routine
PDSYEV	Parallel Double Symmetric Simple Drivers Routine
SSYTRD	Single Symmetric Tridiagonal Drivers Routine
DSYTRD	Double Symmetric Tridiagonal Drivers Routine
EISPACK	Eignen System Package Sub-routine Computing Facility
ARPACK	Arnoldi Package
PARPACK	Parallel Arnoldi Package
SIMD	Single Instruction Multiple-Data
MIMD	Multiple Instruction Multiple-Data

CHAPTER 1

1.Introduction

The history of the eigenvalue starts few century ago with the discovery of digonalization by Augustin Louis Cauchyin when he was working to find normal forms of quadratic functions. It follows proving the spectral theorem for self-adjoint[I] matrices by Cauchy that he was the first systematic about determinants. And John von Neumann generalized the spectral theorem that becomes the most important result of operator theory[1]. And still it continues. The importance of finding eigenvalues and corresponding eigenvectors appears not only in Linear Algebra also, it is addressed widely as solving systems of differential equations, analysing population growth models, and calculating powers of matrices (in order to define the exponential matrix) in other branch of science ,

I2 Self-adjoint matrix (Hermitian matrix) is a square matrix with complex entries that is equal to its own conjugate transpose.

such as physics, sociology, biology, economics and statistics. Accepting the fact of the importance of eigenvalue problems lead us to consider methods comprehensively to find the better solutions of this crucial problem.

Probably, the most popular example from real life is the Google Page Rank Application that applies to Numerical Algebra. Google's success arises with the Page Rank Algorithm, which ranks the importance of the web pages according to an eigenvector of a weighted link matrix[2]. Originally it is the area of Maths for differential equations and the area of physics for quantum mechanics like where the motion is involved in a problem. In other words, eigenvectors points that where there is no direction. Despite, they are used in Structural Geology[I]; eigenpairs are used to determine the principal strain directions where angles are not changing. Eigenvalues are the change in length of the eigenvector from the original length[3]. Car designers analyse eigenvalues in order to damp out the noise so that the occupants have a quiet ride[4]. Eigenvalue analysis is also used in the design of car stereo systems so that the sounds are directed correctly for the listening pleasure of the passengers and driver. When you see a car that vibrates because of the loud booming music, think of eigenvalues. Eigenvalue analysis can indicate what needs to be changed to reduce the vibration of the car due to the music[4].

Fundamentally, *The Symmetric Eigenvalue Problem* has become the problem when we wish to compute the all eigenvalues and eigenvectors of a $n \times n$ matrix. The way of the solving problem has two phases in general [5] . First step is to reduce a symmetric matrix to tridiagonal form. The second step is to find all eigenvalues and eigenvectors of tridiagonal matrix. It can be all done by linear algebra packages which is provided specific function for each eigenvalue problem.

For eigenvalue problem there are many libraries built apart form LAPACK [6] and

I2 **Structural geology** is the study of the three-dimensional distribution of rock units with respect to their deformational histories.

ScaLAPACK [7]. EISPACK, ARPACK, PARPACK etc. However, ScaLAPACK is known as the largest and most flexible public domain library for distributed memory parallel systems up to now. Also, Many LAPACK routines are associated with ScaLAPACK for distributed memory computers using MPI [8]. The basic routines of ScaLAPACK are the PBLAS [9] which contain parallel versions of the BLAS [10] paralleled using BLACS [11] for communication. Thus the PBLAS deliver very good performance on most parallel computers.

In this project we focus on the solving the The Real Symmetric Eigenvalue Problems with QR algorithm using simple driver routines of the LAPACK. The main package BLAS can be said as the core of the Linear Algebra Packages which is written in Fortran Language. The other Packages is built using BLAS routines for fundamental matrix-vector operations are categorized as three levels(Vector-Vector , Vector-Matrix and Matrix-Matrix) can be seen in BLAS quick reference guide [Appendix A]. The demand of using the BLAS library necessitate to increase of the portability in many platform and it is rewritten for other languages such as C, Java. And we will be using the ATLAS version of the BLAS in order to obtain faster programmes.

The LAPACK routines are designed to solve complete problem that is nearly not more than the combination of BLAS routines. LAPACK simple driver routines, SSYEV and DSYEV compute all eigenvalues and optionality all eigenvectors are addressed on solving the problem of a real symmetric matrix via QR algorithm. While the importance of the High Performance Computing is becoming important, BLAS routines are evaluated as PBLAS routines which are used on distributed memory computers and its sub-routines BLAS. Its parallel implementation, the PSSYEV and PDSYEV are the routines of the ScaLAPACK based on PBLAS. The results for the solution of symmetric eigenvalue problems are taken into consideration for performance issues between BLAS and PBLAS libraries.

Apart from the symmetric eigenvalue problems and its computational procedures we will talk about the eigenvalues and eigenvectors of a matrix in algebra, linear transformations of a matrices, definition eigenvector and eigenvalues, their properties with formulas in Chapter 2; which is really lead us to learn computation of the matrices during computational process of the symmetric eigenvalue problem. Detailed definition and explanation is given to related eigensystem[I], the reader can use this fundamental information on other eigenvalue problems, although, they are not addressed in this research such as linear least square, singular value decomposition etc. Moreover, description and general overview of the Linear Algebra Packages are provided in this project.

Chapter 3 indicates solving the symmetric eigenvalue problems using LAPACK routines; SSYEV,DSYEV and ScaLAPACK routines PSSYEV,PDSYEV. Linear algebra packages use the same way two to solve symmetric eigenvalue problems follows two basic steps as it is mentioned above, however they use different method is explained in Chapter 2. The usage of serial and parallel routines and their properties such as matrix storage type, calling scheme are described in this chapter. Detailed information of the environment to implement these routines expressed according to requirements. The idea of the High Performance Computing with 2D block-cyclic distribution is maintained by the combination of the MPI and BLACS.

Discussion of the serial and parallel implementation solving symmetric eigenvalue problems reflected to Chapter 4 from our results. Our findings assessed in a comparable way of the serial and parallel implementation for execution time of the routines. The parallel and serial implementation results discussed indicating the core libraries BLAS and PBLAS which maintain the routines of LAPACK and ScaLAPACK,respectively. To obtain the better performance from ScaLAPACK routines, affects of the different block size for matrices on 2D block-cyclic distribution are con-

I2 **Eigensystems:** The term refers eigenvalue, eigenvector, eigenspace and related concepts in Linear Algebra.

sidered. Graphical representation is given for the execution time of the serial and parallel routines. In addition, execution time of the routines discussed for different block size.

In conclusion, we tried to summarize our findings from our experimental results from Chapter 3. There are some inferences from our results to obtain better performance and the issues related what are the key elements that affects the performance of the routines are discussed briefly in this Chapter. The inferences are based on our system specifications and also methods are used in routines.

Briefly, objectives of this project to compare execution time of the serial and parallel routines for solving the symmetric eigenvalue problems.

It is suggested to the readers to go throughout “Abbreviations” before reading, because abbreviations will not be explained through unless if it is not needed. Some terms are attached at the and of the page as endnote.

CHAPTER 2

2.Background Information

2.1 Overview

The Symmetric Eigenvalue Problem can be defined as the problem of finding eigenvalues and eigenvectors of n*n square matrix A. Theoretically solving the symmetric eigenvalue is related mathematical formulas that can be done manually if the matrix A is (2*2) or (3*3) via roots of the characteristic polynomial. The problem appears if the matrix A is (5*5) or more. In order to obtain eigenpairs[I] of a symmetric matrix many algorithms implemented to find the fastest and more accurate results. Although, different algorithms are implemented, they all follows the same route to obtain eigenvalues and eigenvectors of a symmetric matrix. The process of the solving symmetric eigenvalue has two phases in general

a. Reduction of symmetric matrix to tridiagonal form

b. Solution of the tridiagonal symmetric matrix to find eigenpairs

These two process requires deep understanding in the area of Linear Algebra in or-

I2 Eigenpairs is the term used instead “eigenvalues and eigenvectors”.

der to solve the problem computationally. It requires to understand the type of matrices and as well as their properties. To understand the concept of the eigenvalues and eigenvectors, we will start with linear transformation of matrices. This follows the definition of the eigenvalue problem and the problem will be specified to the solution of the symmetric eigenvalue problem with QR algorithm. Also, brief explanation of other methods will be discussed. For the motivation to next chapter linear algebra packages are determined as our target in this chapter.

2.2 Linear Transformation of Matrices

We will start discussing by linear transformation of a matrices based on geometrical meaning of eigenpairs shown below.

Definition: Let $T: V \to V$ be a linear transformation. If λ is a number and $v \neq 0$ is a non-zero vector such that

$$T(v) = \lambda v \quad (2.1)$$

then λ is an *eigenvalue* and v is an *eigenvector.*

$v \neq 0$ is required and zero vector is not considered to be an eigenvector at all because it has no direction. Moreover, an eigenvalue is always the scaling factor in some eigenvector direction. It can be said an eigenvector is always associated with an eigenvalue.

2.3 What is Eigenvalue Problem?

From previous definition simply gives first insight that finding eigenvalues and finding eigenvectors are equivalent. A linear transformation $T: \mathbb{R}^n \to \mathbb{R}^n$ is given n by n matrix A. The eigenvalue λ and the eigenvector v of T are defined by

$$Av = \lambda v \quad (2.2)$$

The number λ and the vector v are called the eigenvalue and the eigenvector of A. The equation (2.2) can hold only if

$$det|A - \lambda I| = 0 \quad (2.3)$$

and it shows nth degree polynomial which is called characteristic polynomial of square matrix A, in whose roots are the eigenvalues. The results of these definitions follow as ;

λ *is an eigenvalue of matrix A* $\Leftrightarrow Av = \lambda v \, (v \neq 0)$ (2.4)

$\Leftrightarrow (A - \lambda I)x = 0$ *has* $x = v$ *as a non-trivial solution* (2.5)

$\Leftrightarrow A - \lambda I$ *is not invertible*[I] *for square matrices* (2.6)

$\Leftrightarrow det(A - \lambda I) = 0$ *between determinant and invertibility* (2.7)

The way of finding eigenvalues is done simply by root-searching method from the characteristic polynomial. Solving the equation (2.3) will give the eigenvalues $\lambda_1, \lambda_2, \lambda_3, \ldots$ of matrix A. As it explained in before, an eigenvector is always associated with an eigenvalue, from the results (2.4) solving the second equation $(A - \lambda_i I)x = 0$ gives the eigenvectors of matrix A, for each eigenvalue λ_i .

As a result of the explanation, "If A is the matrix of a linear transformation $T : V \to V$ on a finite dimensional vector space with respect to one basis according to;

Theorem 2.1 : *Two square matrices* ***A*** *and* ***B*** *are said to be similar if there is an invertible* ***P****, such that* $\boldsymbol{B = PAP^{-1}}$.

I2 An invertible n*n matrix A satisfies the equation AB=BA=I if there exist a n*n matrix B and I denotes the identity matrix.

From the theorem, the matrix T with respect to another basis is $B=PAP^{-1}$. By the Theorem below,

Theorem 2.2 :*Let* ***A*** *and* ***B*** *be square matrices of the same size. Then det****AB*** *= det****A*** *det****B.***

From two theorems the inference;

$$\boldsymbol{det}(\boldsymbol{B}-\boldsymbol{\lambda}\boldsymbol{I})=\boldsymbol{det}(\boldsymbol{PAP}^{-1}-\boldsymbol{\lambda}\boldsymbol{I})=\boldsymbol{det}(\boldsymbol{P}(\boldsymbol{A}-\boldsymbol{\lambda}\boldsymbol{I})\boldsymbol{P}^{-1})=\boldsymbol{det}(\boldsymbol{P}^{-1}\boldsymbol{P}(\boldsymbol{A}-\boldsymbol{\lambda}\boldsymbol{I}))=\boldsymbol{det}(\boldsymbol{A}-\boldsymbol{\lambda}\boldsymbol{I}) \quad (2.8).$$

Thus the characteristic polynomial of a linear transformation does not depend on the choice of the basis. We also call $det(A-\lambda I)$ the characteristic polynomial of **T**."[13]

Theorem 2.3 Basic Eigenvalue Properties

Let $A\in\mathbb{R}^{n*n}$ be given. Then we have the following properties:

1. There are exactly n eigenvalues, counting multiplicities; complex eigenvalues will occur in conjugate pairs.
2. Eigenvectors corresponding the distinct eigenvalues are independent.
3. If $n*n$ matrix A has n independent eigenvectors, then there exists a non-singular matrix P such that $P^{-1}AP=D$ is diagonal, and A is called diagonalizable. Moreover, the columns of P are the eigenvectors of A, and the elements $d_{ii}=\lambda_i$ are the eigenvalues of A.
4. If A is symmetric ($A=A^T$), then the eigenvalues are real and we can choose the eigenvectors to be real and orthogonal.
5. If A is symmetric, then there is an orthogonal matrix Q such that $Q^TAQ=D$ is diagonal, where the elements $d_{ii}=\lambda_i$ are the eigenvalues of A.

6. If A is triangular, then the eigenvalues are the diagonal elements. $\lambda_i = a_{ii}$.

2.4 Symmetric Eigen Problems

A matrix is called ***symmetric*** if it is equal to its transpose, and for every pair of indices *i* and *j;*

$$A = A^T \quad \text{or} \quad a_{ij} = a_{ji} \quad (2.9)$$

It is called ***self-adjoint*** (***Hermitian)*** if it equals the complex-conjugate of its transpose (its Hermitian conjugate, denoted by "H") ,

$$A = A^H \quad \text{or} \quad a_{ij} = a_{ji} \quad (2.10)$$

It is termed ***orthogonal*** if its transpose equals its inverse,

$$A A^T = A^T A = 1 \quad (2.11)$$

and ***unitary*** if its Hermitian conjugate equals its inverse. Finally, a matrix is called ***normal*** if it commutes with its Hermitian conjugate,

$$A A^H = A^H A \quad (2.12).$$

According to these definitions, Hermitian matrices are symmetric; unitary matrices are orthogonal for real matrices.

With reference to information about eigenvalues, *The Symmetric Eigenvalue Problem* appears on the calculation of eigenpairs of a *Symmetric Matrix* A. Theorem 2.3 shows that the symmetric matrix has n eigenvalues and the number of corresponding vectors are n ; if the eigen-

values are not repeated. Also, all eigenvalues of the symmetric matrix are real, and a complete set of n mutually orthogonal and normalized real eigenvectors exists[13]:

$$q_i q_j = 0 \quad \text{for} \quad i = j \quad \text{and} \quad q_i q_i = 0 \quad \text{for} \quad i = 1, \ldots, n \quad (2.13).$$

From that, this indicates that A has an eigen decomposition;

$$A = Q^T DQ \quad (2.14),$$

where Q = (q1 | . . . | qn) ∈ $\mathbb{R}^{n*n}$ (Theorem 2.3) is an orthogonal matrix, and

$$D = \text{diag}(\lambda_1, \ldots \lambda_n) = \begin{matrix} \lambda_1 & & \cdot \\ & \ddots & \\ \cdot & & \lambda_n \end{matrix} \quad (2.15)$$

The eigenvalues can find ascendantly such that $\lambda_1 < \ldots < \lambda_n$.

2.5 There are some approaches ...

Characteristic polynomial is one of the basic solution and the problem can be solved finding the roots of this equation where the roots are eigenvalues. It seems to reasonable to use this solution for small matrices not for large matrices.

In order to solve this problem for large matrices there are many efficient method have arisen. These methods generally categorized in two different ways; direct and iterative eigensolvers. As with linear systems, while the original matrix is tried to extract selected eigenvalues and eigenvectors from appropriate low-dimensional subspaces of $\mathbb{R}^n$ **i**s called iterative solvers; direct solvers work with the transformation of matrix after a fixed number of operations to obtain eigensystem. In reality, every eigensolver has an iterative component even it is called as direct solvers. There are variety of methods such as; The Power Method, The inverse power method, The Jacobi Algorithm, The QR method, The QR algorithm, Similarity Transformations an so on. [13]

Jacobi Iteration: Joacobi's algorithm solves using diagonalization is the given $n \times n$ symmetric matrix A. This method makes the given matrix more and more diagonal [13] by similarity transformations . Diagonal entries of the diagonal matrix are the eigenvalues of the matrix when it reaches finite diagonal form. And symmetric matrix is always diagonalizable as we know(Theorem 2.3).

Gauss Elimination: This method is one of the popular methods of the linear equations to find the solution 'x' where A is a matrix of order n, and b is a vector of length. It decomposes the original matrix (iterative) into three matrix. The matrix A becomes,

$$A = P * L * U \quad (2.16)$$

P is the permutation matrix, L is the lower triangular matrix and, U is the upper triangular matrix. The solution becomes,

$$P * (L * (U * x)) = b \quad (2.17)$$

From this equation, the determinant of A must equal to the determinant of P*L*U. Determinant of is equal to 1 or -1 and L is 1. And iteration can be done using the new form of A [13].

2.6 Linear Algebra Packages

Practical point of view of the solving eigenproblems, there are many free software applied the methods in one function instead of implementing the methods ourselves: EISPACK, ScaLAPACK, PARPACK, PETSc etc. The advantages of these softwares, they are already implemented and tested in many architectures with the program requirements. The user just need to fit the problem according to their requirements.

2.7 Using Linear Algebra Packages

The linear algebra packages (Table 2.1) written to reach high performance that we used for our implementation to solve the linear algebra problems directly. The methods implemented efficiently for solving the linear algebra problems. The method that we are using is the QR algorithm to find eigenvalues and optionally eigenvectors of a symmetric matrix which is implemented in simple driver routines of the LAPACK; SSYEV and DSYEV.

Software Abbreviation	**Software Name**
BLAS	Basic Linear Algebra Subroutine Programs
CBLAS	Legacy BLAS (Basic Linear Algebra Subroutine Programs with C interface)
PBLAS	Parallel Basic Linear Algebra Subroutine Programs
LAPACK	Linear Algebra PACKage
ATLAS	Automatically Tuned Linear Algebra Software
ScaLAPACK	Scalable Linear Algebra Subprograms
BLACS	Basic Linear Algebra Communication Subprogram

Table 2.1 Linear Algebra Packages used in this project.

LAPACK and BLAS are serial libraries to obtain efficient performance on single processor. Due to low performance of the BLAS we used ATLAS-BLAS version which gives efficient performance and makes easy to use C interface. We just need to include CBLAS and CLAPACK header files in our applications. However, ATLAS has not implemented all LAPACK routines, so we just used for BLAS library. ScaLAPACK routines are corresponding routines of LAPACK for distributed memory parallel systems via PBLAS, which is collection of the BLAS routines parallelized

using BLACS for communication. There are many reasons to use ScaLAPACK for parallel and LAPACK for serial, they have widely available, portable with many systems that consist of a variety of linear algebra routines to solve linear algebra problem. Another reason for that, ScaLAPACK does not change the routines as much as possible, the user should change only few parameters for parallel implementation to use distributed memory computers. The relation between these libraries is shown in Figure.

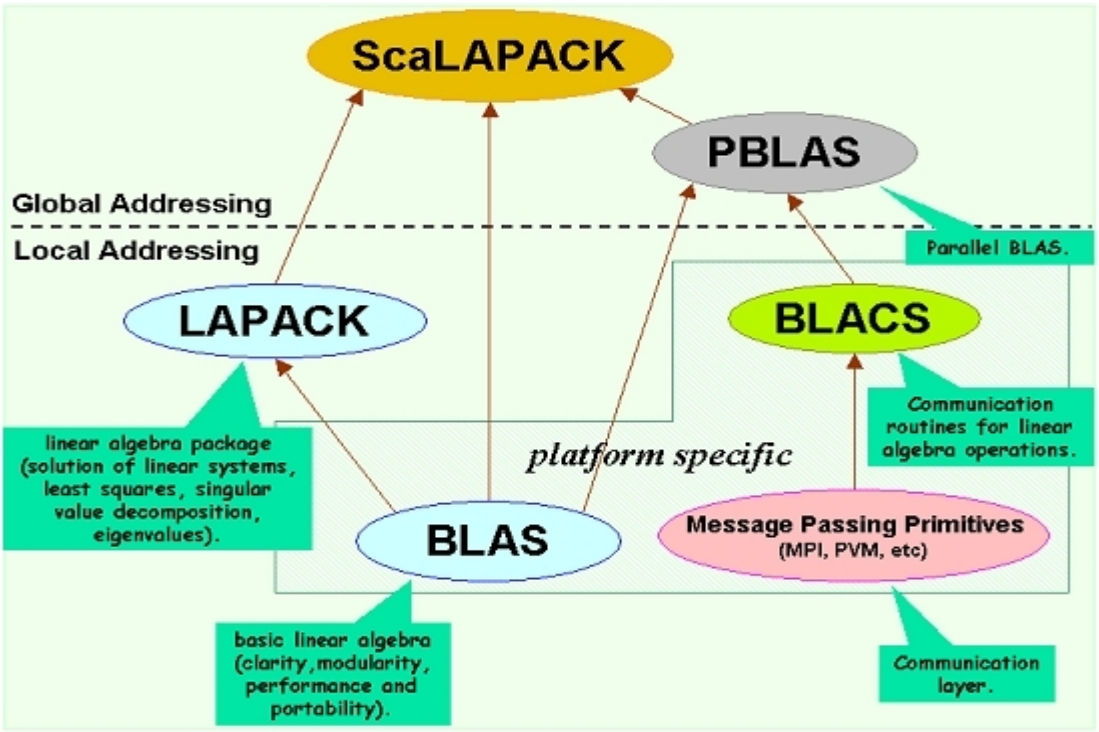


Figure 2.1 *Components of Linear Algebra Packages[14]*

Building Blocks

The key to using Linear algebra Packages effectively is to try to reduce unnecessary memory references. Machine specific data flows between memory, registers and from registers, functional units, which perform the given instructions on the data. To avoid the such as data flows the use of

the linear algebra packages are improved to perform building blocks. Results for many architectures the partitioning of a matrix or performing the computation dividing the matrix into blocks by doing the matrix-matrix operation on blocks shows that it is worthy to build blocks on solving the problems efficiently. In addition, using blocks provides parallel processing for many architectures. Now we will give a brief description of these libraries mentioning the usage of in this project.

The importance of BLAS

BLAS [10] is defined as routines that provide standard building blocks for performing basic vector-matrix operations and it is probably most important part of solving our linear algebra problems. Many applications commonly use BLAS routines to develop high quality linear algebra softwares. It is free and widely available. Moreover, the efficiency of the LAPACK routines actually depend on the BLAS routines, provided by the computer vendors for a specific machine. Thus the BLAS form a low-level interface between LAPACK software and different machine architectures. Above this level, almost all of the LAPACK software is truly portable. As a consequence of this, the performance of the PBLAS, hence ScaLAPACK, depend upon the BLAS. BLAS routines are first written in Fortran but it provides binding for other languages (eg. C, Java) , we have used C interface in this project. Consequently, it can be said its performance, accuracy, and the results that we obtain from our application, partially depend upon the routines which perform in 3 levels as described in Table 2.2.

Level 1	Scalar, Vector and Vector-Vector operations ($y=\alpha x+y$)
Level 2	Matrix-Vector Operations ($y=\alpha Ax+\beta y$)
Level 3	Matrix-Matrix Operations ($y=\alpha AB+C$)

Table 2.2:Level of BLAS Routines

If we go one step further to find the reason that which level of the BLAS effects the performance of routines; with Level 3, it can achieve high performance and Level 1 gives the lowest performance due to the high number of operations.

ATLAS_BLAS: The ATLAS is an open-source package that empirically tunes the library to the machine it is being compiled on. The version of the ATLAS library used in this project for high performance BLAS libraries. Available functions is given in quick reference, please see Appendix A.

LAPACK and ScaLAPACK

LAPACK project consists of wide range of routines for solving linear algebra problems such as eigenvalue problems, linear equation, least-square solutions of linear systems of equations, and singular value decomposition. LAPACK routines provides highly efficient machine-specific implementation of the BLAS for high-performance computers. The evaluation of its routines appear for distributed memory architectures which becomes ScaLAPACK (Scalable LAPACK) which is portable for many architectures. The main idea of the designation of the ScaLAPACK underlines for heterogeneous computing and its portability with many system which supports MPI and PVM make use of this library widely. The ScaLAPACK is consisted for parallel implementation of the LAPACK routines as we mentioned before. It has performed on PBLAS routines (parallel version of the BLAS) communicating with BLACS (Basic Linear Algebra Communication Sub-programs). Although, it has designed as Multiple-Instruction-Multiple-Data (MIMD) ,the method of implementation is written as Single-Instruction -Multiple-Data (SIMD) and the way of the computing the matrices is called “two dimensional block cyclic decomposition". This method is accomplished via distribution of the global matrices building blocks.

Parallel implementation of routines provided in two ways in ScaLAPACK. Operations on

distinct blocks may be performed in parallel and within the operations on each block, scalar or vector operations may be performed in parallel. This is done internally by routines when we call the function. However, the performance of the program can be tuned by the user by changing some variables. While on shared memory machines, only the block size can be defined by the user, on distributed memory machines, either block size or process grid can be defined by the user [15].

MPI and BLACS

BLACS is simply communication library which makes portable the use of ScaLAPACK for distributed memory architectures. MPI provides easy-use of BLACS and one of the key point to obtain efficiency with ScaLAPACK relies on BLACS of the communication between BLAS and PBLAS.

Symmetric Eigenvalue Problem with LAPACK and ScaLAPACK

Theoretically, the solution come up with the completing the two steps for symmetric eigenproblems with linear algebra packages as it mentioned here. Simple driver routines are implemented to solve complete problem differs than computational routines they call the functions separately for each steps. The simple driver routines SSYEV and DSYEV theoretical solution explained with the usage of other LAPACK routines.

1. ***The reduction of the symmetric matrix to tridiagonal form***

The matrix is illustrated symmetric tridiagonal matrix of order n, If A is real symmetric matrix, it can be decomposed as $A = QTQ^T$ to obtain the real symmetric tridiagonal matrix T, and Q is the orthogonal matrix. According to definition of the tridiagonal matrix: “Tridiagonal mat-

rix **A** is also symmetric if and only if its non-zero elements are found only on the diagonal, sub-diagonal, and super-diagonal of the matrix, and its sub-diagonal elements and super-diagonal elements are equal" [13]; that is:

$$a_{ij}=0 \quad \text{if} \quad |i-j|>1 \quad \text{and} \quad a_{ij}=a_{ji} \quad \text{if} \quad |i-j|=1 \quad (2.18)$$

$$\begin{pmatrix} a_{11} & a_{21} & 0 & . & . & . & . & 0 \\ a_{21} & a_{22} & a_{32} & 0 & . & . & . & . \\ 0 & a_{32} & a_{33} & a_{43} & 0 & . & . & . \\ . & 0 & a_{43} & a_{44} & a_{54} & . & . & . \\ . & . & 0 & a_{54} & a_{55} & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & a_{nn} \end{pmatrix} \quad (2.19)$$

The reason we transform the tridiagonal form is; if a symmetric matrix A can diagonalizable, it has the real eigenvalues and eigenvectors can be found respectively. Also, tridiagonal form has zero-entries except its sub-diagonal, super-diagonal and diagonal entries that is reduced the half work.

To reducing the matrix tridiagonal form is done by the routine SSYTRD and DSYTRD. When we called the function SSYEV, it will call the function SSYTRD likewise DSYEV will call DSYTRD function.

Type of the matrix and storage type	Single precision	Double Precision
Dense Symmetric	SSYTRD	DSYTRD

Table 2.3 Routines to reduce to tridiagonal form

The Table indicates the LAPACK routines for the specific symmetric matrices are used to transform to tridiagonal form. There are 3 types of routines are given according to matrix storage type in LAPACK. However, we are implementing the Dense Symmetric Matrix which has no special pattern , or which is known nothing about its pattern and it can be called also full or general matrix. In our case, it can be said it has just one special pattern that is the symmetric is the only

thing we know upper and lower part entries are equal symmetrically which means $A_{ij} = A_{ji}$.

2. *Eigenpairs of the Symmetric Tridiagonal Matrix*

After obtaining the tridiagonal symmetric matrix T, we can compute the eigenvalues and its corresponding eigenvectors if it is desired via implicit QR algorithm. The standard algorithm for the QR decomposition involves successive Householder transformations. An appropriate Householder matrix applied to a given matrix can zero all elements in a column of the matrix situated below a chosen element. Thus we arrange for the first Householder matrix Q_1 to zero all elements in the first column of A. Similarly Q_2 zeros all elements in the second column below the second element, and so on up to $Q_{(n-1)}$. Thus

$$R = Q_{(n-1)} \ldots Q_1 . A \qquad (2.20)$$

Since the Householder matrices are orthogonal,

$$Q = (Q_{(n-1)} \ldots Q_1)^{-1} = Q_{1.} .. Q_{(n-1)} \qquad (2.21)$$

In most applications we don't need to form Q explicitly; we instead store it in the factored form (2.21). In LAPACK, the computation of the matrices are done by building blocks that means the block reduction algorithm as described above to introduce zeros in a block of the matrix, say we are at the k -th stage and have just introduced zeros into the k -th block. As we start the next block reduction, on the k +1-st block, we can start in parallel the eigenvalue computation on that part of the tridiagonal matrix generated from the k -th block reduction of the matrix.

After generating the orthogonal matrix, the QR routine are called to find the eigenpairs. Table 2.6 shows the routines below.

Type of the matrix and storage type		Single precision	Double Precision
Finds only eigenvalues		SSTERF	DSTERF
Eigenvalues and Eigen-vectors	Generates orthogonal matrix	SORGTR	DORGTR
	Symmetric Tridiagonal	SSTEQR	DSTEQR

Table 2.4 Computational routines for tridiagonal matrix via QR method

CHAPTER 3

Implementation Details

Overview

The idea of solving the Symmetric eigenvalue problems is not just to find eigenvalues and eigenvectors of a symmetric matrix in our project. Our aim to find the difference between execution time of the ScaLAPACK routines against LAPACK routines using the one of the driver routines to solve eigenvalue problems of a Real Symmetric Matrix. Apart from that, our program finds eigenvalues and corresponding eigenvectors first with one of the LAPACK routines, then implementation of the corresponding routine in ScaLAPACK gives the same results utilizing the BLAS and PBLAS subroutines while they use the same method which is QR algorithm. Moreover, the parallelism of the ScaLAPACK routines with MPI environment decreases the execution time.

Thereby, our expectation is to obtain efficient results which makes ScaLAPACK performance more efficient than LAPACK that is also one of the aims of ScaLAPACK Project.

Hardware and Software Requirements

Implementation of related linear algebra packages (LAPACK and ScaLAPACK) and their dependency packages (Table 3.2) can be downloaded [12]. Operating system Ubuntu 9.10 (Karmic)

on the AMD Athlon 64 dual-core processor machine is used during the project implementation. The information about the system architecture was gathered using the command line;

cat /proc/cpuinfo (3.1)

The system information is available in Table 3.1. More detail system information can be found in Appendix B. For compilation of the programmes GCC compiler is used that is free and commonly available and for many programmes as well as High Performance Linear Algebra Programmes

SYSTEM	
Operating System	UBUNTU Release 9.10 (Karmic) Kernel Linux 2.6.31-22-generic GNOME 2.28.1
Vendor ID	AuthenticAMD
Model Name	Processor 0 : AMD 64 Athlon(tm) X2 Dual-Core QL 62 Processor 1 : AMD 64 Athlon(tm) X2 Dual-Core QL 62
CPU (MHz)	1000

Table 3.1 *System Information*

SOFTWARE PACKAGES		
Packages and Dependency Packages	**LAPACK** liblapack-test,liblapack-doc, liblapack-pic, liblapack3gf, liblapack-dev, libatlas-base-dev, libblas-doc, libatlas-headers, libblas-dev,libblas3gf,libatlas3g	**ScaLAPACK** libscalapack-mpi-dev 1.8.03.1build1 , libscalapack-mpi1 1.8.0-3.1build1 , libscalapack-pvm-dev 1.8.03.1build1 , libscalapack-pvm1 1.8.0-3.1build1, scalapack-mpi-test 1.8.0-3.1build1 scalapack-pvm-test 1.8.0-3.1build1
Other Packages	OpenMPI, BLACS	

***Table 3.2** : Required packages for implementation*

For the users who wants to build the libraries locally, a python based installer package is provided by ScaLAPACK. The source file name is “scalapack_installer.tgz” [7] provides libraries BLAS, LAPACK, BLACS, ScaLAPACK with all configurations. In this project, we installed the package “scalapack_installer.tgz” the output of the files can be seen with all instructions in Appendix B. In our implementation we will call only the BLACS library from this package. For the compilation of the programme, we should link all these packages to our programme. ScaLAPACK and BLACS should be linked with MPI. C language is used in both, serial and parallel implementations.

Single and Double Precision

Floating point arithmetic derives from the storage of numbers with an exponential notation which shows the numerically equivalent number. It means that the number x to be stored in floating point form is normally separated into its components; a and b such that,

$$x = a.2^{b} \quad (3.2)$$

whereas b is called the exponent, is the smallest possible integer for which mantissa[I], a, satisfies the restriction

$$-1 \leq a \leq 1 \quad (3.3).$$

From the brief description above, the only difference between double and single precision, storage of a decimal number of bits in memory. It is description always bring to mind round off error.

In this project, we are not measuring the accuracy of the eigenpairs. Instead, we are interested in the execution time of the routines. In some resources, it is mentioned that the type of precision effects the performance of the matrix, while some articles says, the difference between double and single precision storage format can not be related with execution time.

I2 Mantissa is the part of a floating point number that contains its significant digits.

Moreover, In C programming, all floating arithmetic are represented in double precision; whenever float number is appears in the code it is lengthened to double by zero-padding its fraction[20].

Calling Fortran functions in C

Before start explanation of serial and parallel implementation, it should be mentioned that the libraries which we are using are written in Fortran. There are some difference in the implementation of the routines in calling methods, passing arguments and definition of an array is shown Table 3.3

C Language	
Calling method	ssyev_
Passing arguments	dgetrf_(&M, &N, A, &LDA, ipiv, &info);
Definition of an array	Row-Major-Order

Table 3.3 *Function call in C*

Serial Implementation

LAPACK has three levels of routines; Driver Routines, Computational Routines and Auxiliary Routines[I]. Each routine has written in two different storage schema[II]; double and single precision. Under each schema type of the matrix is described as real and complex. Implementation of the driver routines SSYEV (single precision) and DSYEV (double precision) has chosen for the type of real matrix (Table 3.4) to examine the simple driver routines execution time for the specific matrix size of a symmetric matrix NxN.

I2 Auxiliary routines are helper routines in LAPACK.
II2 Single and Double Precision referred as storage scheme in LAPACK.

Driver Routines	
Single Precision	Double Precision
Real	Real
SSYEV	DSYEV

Table 3.4 *A small hierarchy of LAPACK routines*

Implementation of double and single precision is same in both, SSYEV and DSYEV. The only difference is the data type of the arrays defined as float in SSYEV and double in DSYEV.

Single Precision: SSYEV and Double Precision: DSYEV

As we mentioned before, the only difference between double and single precision is the data type of the array parameters A, W and Work. All other properties are same in both.

The following calling scheme for SSYEV

$$ssyev(jobz, uplo, n, A, lda, W, work, lwork, info)$$

The following calling scheme for DSYEV;

$$dsyev(jobz, uplo, n, A, lda, W, work, lwork, info)$$

where ***'jobz'*** indicates whether computation of the only eigenvalues or both eigenvalues and the eigenvectors; ***'uplo'*** is the provided upper or lower triangular part of the symmetric matrix for the method. And they are both characters. ***'lda'*** is the leading dimension of A. And W is the array of eigenvalues of A. The leading dimension has to be chosen according to the formula,

LDA >= max(1,N)

where we already use the matrix size as leading dimension in our serial programme.

Symmetric Matrix Storage

Matrix storage is the way of keeping the matrix in memory which is economically done for symmetric matrices. The method of storing the symmetric matrix is to omit repeated values follow as,

UPLO	Triangular Matrix	Packed Storage in Array
U	$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ . & a_{22} & a_{23} & a_{24} \\ . & . & a_{33} & a_{34} \\ . & . & . & a_{44} \end{bmatrix}$	$a_{11}\underbrace{a_{12}a_{22}}\ \underbrace{a_{13}a_{23}a_{33}}\ \underbrace{a_{14}a_{24}a_{34}a_{44}}$
L	$\begin{bmatrix} a_{11} & . & . & . \\ a_{21} & a_{22} & . & . \\ a_{31} & a_{32} & a_{33} & . \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$	$\underbrace{a_{11}a_{21}a_{31}a_{41}}\ \underbrace{a_{22}a_{32}a_{42}}\ \underbrace{a_{33}a_{43}}\ a_{44}$

Table 3.5 *Symmetric Storage Schema*

The formula [7]to store the

'U', *upper* part of the matrix ; AP (i+(j-1)/2) (i<=j)

'L' ,*lower* part of the matrix ;AP (i+(2n-j)(j-1)/2) (j<=i)

Linear Algebra Programmes does not store redundant values in a symmetric matrix because values are repeated. Upper triangle part of the matrix packed into a linear vector length, (n*(n+1)/2), where n is the size of matrix.

Input /Output Arguments

The function takes the symmetric matrix as one dimensional array as the order of the symmetric matrix is “n”. A is input parameter for the specifying upper or lower part of the triangular matrix according to values of “uplo” ; and also output parameter, on exit ,it contains the orthonormal eigenvectors of the matrix if the value of jobz = V. if the These two input parameters; “jobz” finds only eigenvalues if the given value is “N” or eigenvalues and eigenvectors if the given value “N”. “uplo” specifies whether the symmetric matrix A, contains the upper triangular part of the matrix with the value “U” or lower part of the matrix with the value “V”. “lda” is the input para-

meter leading dimension of the symmetric matrix A. On exit , “W” is an array that gives the eigenvalues of the symmetric matrix. Before that the input parameter “lwork” is defined the length of the array “W” and it determines the necessary minimum workspace to perform the operations. The last argument “info” is just indicates the whether computation of the routine is successful or not.

Algorithm for SSYEV and DSYEV

```
Begin
      define the array dimension
      if uplo=U
           take the upper triangular part of the matrix
      else if uplo=L
           take the lower triangular part of the matrix
       allocate space for symmetric matrix
       initialize symmetric matrix
       allocate the optimal workspace
       call function (ssyev_ /dsyev_)
        if jobz=U
            calculate all eigenvalues and eigenvectors
       else if jobz=V
            calculate only eigenvalues
       free memory
End
```

Parallel Implementation

In this part, ScaLAPACK (Scalable LAPACK) is used to perform the routines of the LAPACK. ScaLAPACK is the redesign of LAPACK routines for distributed memory MIMD parallel computers.

The process of parallel implementation of routines based on the matrices layout which has accomplished using 2D block-cyclic method to obtain the high performance on calculation of the eigenpairs of a symmetric matrix. ScaLAPACK uses BLACS for communication and MPI is the

message passing interface to set the parallel computing environment.

Similar to LAPACK, ScaLAPACK routines linked with PBLAS which is a parallel implementation of the BLAS routines. The process of calculating the eigenvalues using LAPACK routines is slightly different than ScaLAPACK routines.

The general frame of the using ScaLAPACK routines appears in four steps. Initializing the process grid first thing has to be done that we can distribute the matrix on the process grid. After calling the routine from ScaLAPACK, process grid can be released. Apart from these steps, we will cover our frame with a outer frame is called MPI. MPI is an portable interface for ScaLAPACK and suggested to use by ScaLAPACK.

It is defined functionality and mentioned the main purpose of the MPI and interaction with BLACS is mentioned in Chapter 2 because ScaLAPACK is required to inter-communication of MPI and BLACS. Setting up the communication bridge between these two libraries is required to make some changes when building BLACS due to inter-language translations of BLACS between Fortran and C. A communicator translator take place only when the user calls BLACS_GRIDINIT and BLACS_GRIDMAP from a different language interface than the MPI interface[16]. So, we should tell that we are using MPI to the BLACS. This can be done by changing the Bmake.inc file is in the main directory of BLACS where we installed. Following line,

- TRANSCOMM = -DuseMpi2

will tell directly that we are using MPI.

We examined the corresponding ScaLAPACK routines PSSYEV and PDSYEV to compare their execution time between LAPACK routines.

Driver Routines	
Single Precision	Double Precision
Real	Real
PSSYEV	PDSYEV

Table 3.6 *A small hierarchy of ScaLAPACK routines*

Single Precision: PSSYEV and Double Precision: PDSYEV

Calling Scheme for SSYEV[ ]

$$ssyev(jobz, uplo, n, A, IA, JA, descA, W, Z, IZ, JZ, descZ, work, lwork, info)$$

Calling scheme for DSYEV[ ]

$$dsyev(jobz, uplo, n, A, IA, JA, descA, W, Z, IZ, JZ, descZ, work, lwork, info)$$

"The term global data mapped to the local memories of processes and the term local is the data on each process is referred as the local array [17]" is defined by ScaLAPACK ,and ,input/output parameters called as global input and global output.

The corresponding ScaLAPACK routines for the parallel implementation PSSYEV and PDSYEV follows the same structure of routines is explained in explanation of the SSYEV LAPACK routine in general for some parameters such as "jobz", "uplo", some work space variables. The differences are as it can be seen from the calling scheme above, the leading dimension is replaced by IA and JA indices which is starting position of the global matrix A and usually IA=1 and JA=1. 'descA' is descriptor of A, which holds all information about distribution of the global matrix. 'Z' is the output array which return the eigenvectors. IZ, JZ and descZ have the same duty like explained above details for matrix A. The description of the routines can be obtain from the source file of the ScaLAPACK and also ScaLAPACK routines require one or more work arrays to be passed as arguments.

Algorithm For PSSYEV and PDSYEV

```
Begin
    define the array dimension
    if uplo=U
        take the upper triangular part of the matrix
    else if uplo=L
        take the lower triangular part of the matrix
     Initialize MPI
        Find processes and their id in MPI environment
        Create PrxPc Process Grid
        find processes in Cartesian topology (myrow)x(mycol)
        choose id of the process to distinguish the roles of other processes
        Find local dimension of the distributed matrix with "numroc" function
        allocate memory for arrays
        initialize symmetric matrix
        call function (pssyev_ /pdsyev_)
        if jobz=U
            calculate all eigenvalues and eigenvectors
        else if job=V
            calculate only eigenvalues
      free memory
       Release Process Grid
    Finalize MPI
End
```

I. Initialize the MPI

The duty of the MPI is to exchange the data cooperatively between processes. Basic MPI functions are accessible including the "mpi.h" header file. An it should be included in programme. Then, we set the MPI environment initialization and finalizing the MPI with C interface. Initialization of the MPI start with these lines of code ;

1. MPI_Init(&argc,&argv);
2. MPI_Comm_size(MPI_COMM_WORLD,&size);
3. MPI_Comm_rank(MPI_COMM_WORLD,&rank);

All MPI programmes start with the first line of code. The second line determines the how many processes we have in this environment and the third line gives id to each processes. And it is limited as

0<=rank<= size -1.

Now, all programs are running equivalently, the rank assigned the job to the processes.

II. Initialize the process grid

In one dimensional grid, process is referred as $0,1,2,3,...P-1$ for the process ' P where P_r and P_c determine the rows and columns in two dimensional array in process grid. BLACS set up the process grid for the ScaLAPACK routines. The process grid A can be referred as $P_r * P_c = PA$. A process grid has p_r rows and p_c columns itself which is

$$0 < p_r < P_r \quad and \quad 0 < p_c > P_c \quad .$$

BLACS mapped to processes to process grid in ***row-major order*** as default but it can be changed as column-major order since we use MPI Cartesian Topology only allows row major order [18]. BLACS allow a process to be a member of several overlapping or disjoint process grids, each one labelled by a **context** that means in ScaLAPACK each process is enclosed in a context which is associated with every global matrix in ScaLAPACK. The BLACS provide safe inter-operation with its contexts and system context such as MPI includes this context concept.

The two grid creation routines in BLACS; BLACS_GRIDINIT and BLACS_GRIDMAP. Both they create a process grid and its enclosing context. These routines return context handles, which are simple integers, assigned by the BLACS to identify the context and

they should not be manipulate by the user because they are called “opaque objects[1]”. Subsequent routines if the BLACS are passed these handles, which allow the BLACS to determine a routine from which context grid is being called. After the determined context used resources it should be released via routine BLACS_GRIDEXIT and the contexts will be free after the routine BLACS_EXIT is called to shut down the system [11].

ScaLAPACK TOOLS routine initialize the process grid is referred as Pr x Pc as shown in our code “nprow” and “npcol” in column-major order (whereas it is row-major order in Fortran) to obtain its 'context' which is default in BLACS. Identifiying the coordinates of the process grid we used the “myrow” (identifier rows) and “mycol”(identifier columns) that their size ,

$$0 <= myrow <= nrows-1$$

$$0 <= mycols <= ncols-1$$

like that size of the number of processes can be determined,

$$number\ of\ processes = nrows * ncols$$

One thing we should mention here, the C interface for the BLACS functions are different .It uses the “Cblacs_gridifno, Cblacs_gridmap,Cblacs_exit” like we implemented in our code.

III. Distribution of the matrix on the process grid

After initializing the process grid, matrices should be distributed on the process grid before invocation of the ScaLAPACK routines. The only thing is ScaLAPACK does not do the distribution of the matrix, user has to do this regarding the requirements that mentioned the setting up the process grid and initializing the number of processors rows (nprows) and the number of processors columns (npcols).

I2 Opaque objects hold default values by BLACS and it is not recommended to change.

- **2D Block-Cyclic Distribution and Performance**

Block-cyclic distribution consists of two common decompositions: Block and Cyclic Decomposition. Block decomposition[Table 3.10] assigns entries contiguously in the global vector to the processes in blocks which is a distribution of the computational load homogeneously over the grids that is mentioned as a regular data structure. Cyclic decomposition [Table 3.11]assigns the consecutive entries in the global vector to different processes. Heterogeneous distribution of the computational load over a regular data structure is used to improved load balance [lawn58]. Decomposition of M= 10 (data objects) and P=3 (processes) shown in Table 3.10 and 3.11 as an block and cyclic example. 'm' indicates $m \rightarrow (p, b, i)$ where,

- p is the process number
- b is the block number in process p
- i is the index within block b to which is mapped to m.

m	0 1 2 3 4 5 6 7 8 9	$m \rightarrow ([m/L] m \bmod L)$
p	0 0 0 0 1 1 1 1 2 2	where $L=(M/P)$
i	0 1 2 3 0 1 2 3 0 1	

Table 3.7 *Block Decomposition*

m	0 1 2 3 4 5 6 7 8 9	$m \rightarrow (m \bmod P[m/P])$
p	0 0 0 0 1 1 1 1 2 2	
i	0 1 2 3 0 1 2 3 0 1	

Table 3.8 *Cyclic Decomposition*

The combination of the block and cyclic decomposition creates new debate for distribution of the data is called Block-Cyclic Distribution to obtain best performance suits for most systems. The Fig-

ure 3.1 shows how the data distributed on a process grid to decompose the matrix as block-cyclicly. As an example, the MxN=9x9 matrix into MBxNB=2x2 block sizes starts from upper left corner of the matrix. Then the bocks are mapped to process grid PrxPc=2x3 that means each process owns a collection of blocks, which are locally and contiguously stored in a two dimensional “column major” array. The Figure shows:

S Y S Y	M M M M	E T E T	R I R I	C C
S Y S Y	M M M M	E T E T	R I R I	C C
S Y S Y	M M M M	E T E T	R I R I	C C
S Y S Y	M M M M	E T E T	R I R I	C C
S Y	M M	E T	R I	C

Figure 3.1 *Partitioning of global matrix*

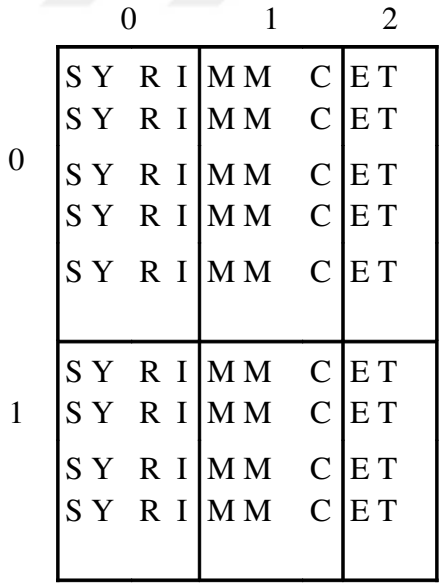


Figure 3.2 *Mapping matrix onto process grid (Pr=2, Pc=3)*

There are 6 processes that we figured out ;P(0,0), P(0,1), P(0,2), P(1,0), P(1,1), P(1,2).

Block-cyclic distribution from the figure depends on the process grid size for the chosen block size of the matrix. Matrix will be divided to small matrices according to applicable block size, small matrices are called blocks will be mapped to process grid. ScaLAPACK uses 2D block-cyclic distribution for the implementation of the routines in distributed systems that user has to choose good compromise for the size of blocks to achieve [19]

- good computation,
- communication efficiency,
- good load balancing.

Arrays are wrapped by blocks in all dimensions corresponding to the process grid [19]. Sequentially, We are able to defined the process grid as 1x1 for our implementation machine AMD Athlon we can just increase the efficiency using appropriate block size. So, we followed the other explanations in S*caLAPACK User's Guide* explains how to obtain higher performance on the distributed memory computer follows;

1. Using the right number of processors. The rule is P=MxN/1000000 for MxN matrix size. This provides a local matrix of size nearly 1000 by 1000. According to this if the symmetric matrix size 1000, processors should be 1000x1000 /1000000= 1

    1.1. A small problem should not be solved with too many processors.

    1.2. Physical memory should not be exceed.

    1.3. Should not be solved a small problem on too many processors.

2. Using an efficient data distribution.

    2.1. Block size should be 32 or 64 for the best performance.

2.2. Square process grid should be Pr = Pc (Pr: number of process row ,Pc:number of process column)

3. Efficient machine-specific BLAS should be used, Fortran 77 reference implementation BLAS should not be used due to performance issues.

4. BLACS settings should change in the Bmake.inc as non-debug: BLACSDBGLVL=0

Performance issues are mentioned in [B] that says the ScaLAPACK library is not sensitive to the block size, as long as the extreme cases are avoided. However,very small matrices size leads to poorer performance and leads to use BLAS 2 operations while ScaLAPACK and LAPACK use BLAS 3 operations. Moreover, very large block size is not a appropriate choice because it causes computational imbalance [15].

- **Matrix Descriptors**

This process follows with array description that is assigned to each matrix across the process grid. ScaLAPACK TOOLS routine call DESCINIT is set to the routine PxDSYEV before it invokes. PDSYEV routine is required associated a *description vector* for each 2D block-cyclic distribution matrix. The information to determine the mapping between a matrix entry and its corresponding process and memory location is enclosed in the description vector[dsyev.f]. That means the global matrix A is assigned to an "*array descriptor*" with the notation DESC_ on the process grid that becomes DESCA for the global matrix A.

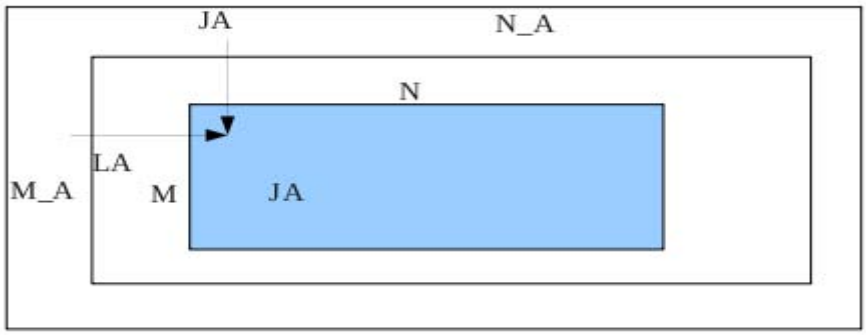


Figure 3.3 *Global view of the matrix operands[7]*

The Figure indicates the global matrix, and number of rows/columns of the global matrix ,M_A, N_A ; number of rows/columns of the sub-matrix, M ,N ; row/column indexes of the global matrix IA and JA. According to this figure, while global matrix A is shown as A(IA : IA+M-1, JA:JA+M-1), the sub-array described as (M,N,A,IA,JA,DESCA).

Descriptor Vector, A, encapsulate the matrix information 9 integer vector shown in Table 3.5

Descriptor Vector Elements	**Name**	***Definition***
descA(1)	dtype	*Descriptor type (=1 for dense matrices)*
descA(2)	ctxt	*BLACS context handle for the process grid*
descA(3)	m	*Number of rows in global matrix*
descA(4)	n	*Number of columns in global matrix*
descA(5)	mb	*Row blocking factor*
descA(6)	nb	*Column blocking factor*
descA(7)	rsrc	*Process row over which the first row of the global array is distributed*
descA(8)	csrc	*Process column over which the first column of the global array is distributed*
descA(9)	lld	*Leading dimension of the local matrix*

Table 3.9 : *Descriptor Vector Elements, Names, and Definitions[7]*

Array descriptors are provided for 4 types of the matrix in ScaLAPACK which are differentiated by the DTYPE_ entry in the descriptor (Table 4.2). And each descriptor arrays has different storage for the global data.

DESC_(DTYPE_)	Designation
1	Dense Matrices
501	Narrow Band and Tridiagonal coefficient matrices
502	Narrow Band and Tridiagonal right-hand-side matrices
601	Out-of-core dense matrices

Table 3.10: *Valid values of DESC_(DTYPE_)*

IV. Calling the routines

After distribution of the matrix, we can call the functions as we showed in Table 3.5 how to call functions with their parameters. Function will do its job calling the subroutines(is explained in Chapter 2) to process the two main steps for finding the eigenpairs of the symmetric matrix.

V. Exit Process Grid and Finalize MPI

Finally, routines executed the results and BLACS release the process grid. After that MPI close the environment.

Here, it should be mention again the importance of the BLAS for both libraries as it is told "*the performance of LAPACK and ScaLAPACK largely depends on the performance of the BLAS library*"[15]. So, we will be using the BLAS(ATLAS) to obtain better performance. However, we use the same BLAS library for both so the performance results are not related to BLAS libraries. ScaLAPACK functionality has not evaluated for all LAPACK routines so before implementation of the LAPACK routine, ScaLAPACK quick reference guide (Appendix A) is suggested to find available corresponding routines.

CHAPTER 4

Results and Comparison

The comparison of the execution times of serial and parallel implementations are addressed in this chapter. We compare the execution time of LAPACK routines against ScaLAPACK routines for different size of the symmetric matrices, considering the block size of the ScaLAPACK routines which is defined by the user explicitly.

We examined the symmetric matrices (all non-zero elements) generating randomly values between 1 and 10; up-to 1500 matrix size increment by 100. Matrices are compared for different block-sizes are mapped to the process grid PrxPc=1x1.

The time is calculated as seconds using the C timer for the execution of the functions. The average is calculated after 100 iterations for each routine.

The idea behind LAPACK and ScaLAPACK solving the linear algebra problems via building blocks explained in Chapter 2. BLAS and LAPACK solves the problems using the blocks externally already implemented, no need to be specified by the user. On the other hand, PBLAS and ScaLAPACK routines are left to the user. However, the experimental results of ScaLAPACK sug-

gested that the optimal block size should be 32 or 64[2]. During testing process of the routines, we tried to find the appropriate block size of the matrices for ScaLAPACK functions.

Our expectation is to obtain significant results between serial and parallel implementation. In general, execution time of the LAPACK simple driver routines, SSYEV and DSYEV, seems to more than ScaLAPACK routines PSSYEV and PDSYEV. Comparison is done among the following drivers routines;

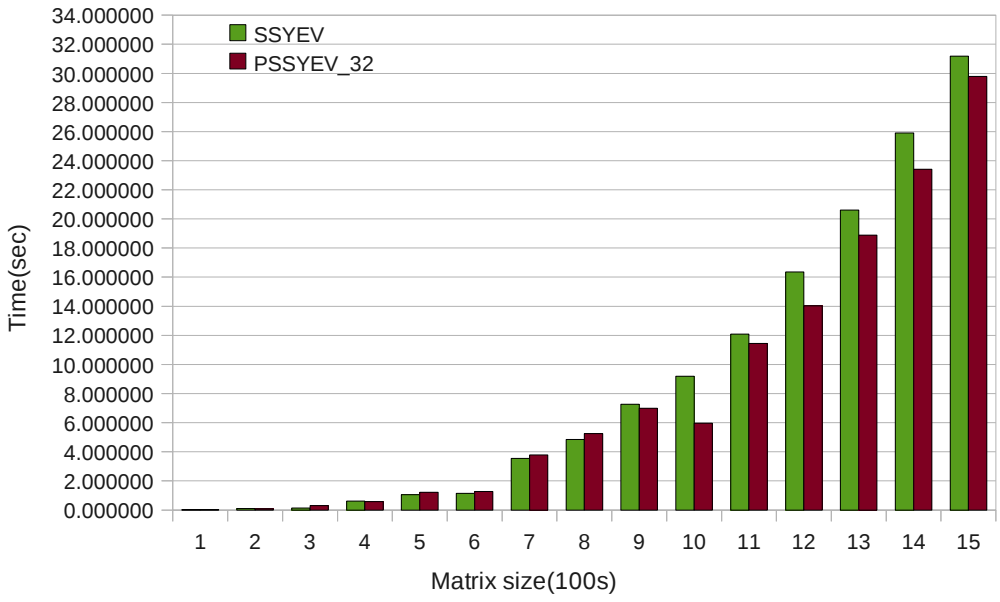
1. Comparison of SSYEV and PSSYEV

2. Comparison of DSYEV and PDSYEV

3. Comparison of PSSYEV for block size 32 and 64

4. Comparison of PDSYEV for block size 32 and 64

Exact values from results can be seen in Appendix C. Graph 1 shows the execution time of the routines SSYEV and PSSYEV, the block size of the PSSYEV is 32.

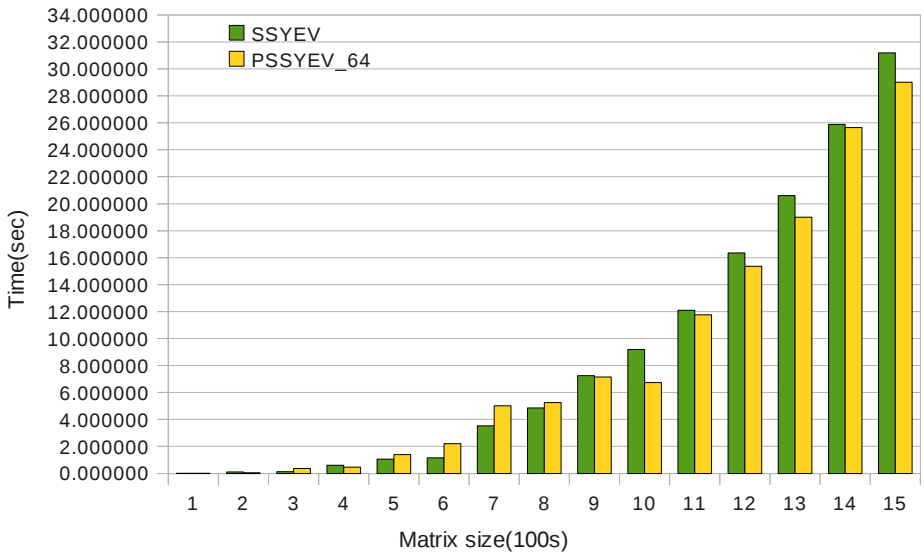


Graph 1: Comparison between execution time of SSYEV and PSSYEV

From the Graph 2, for matrix size between 100 and 900, there is no much difference. But, matrix

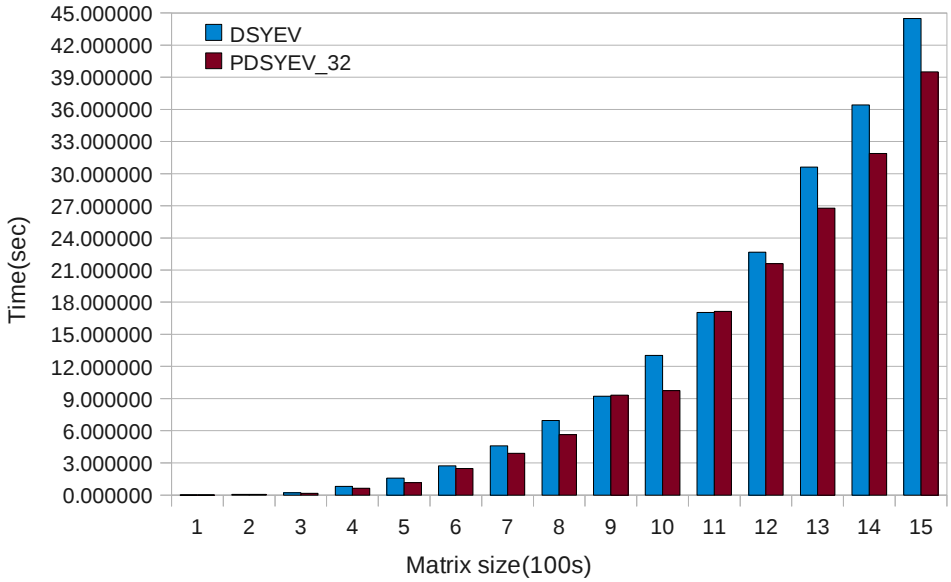
size 1000, PSSYEV has significantly decrease. The execution time for PSSYEV are less than SSYEV routines for 1000-1500 matrix size. SSYEV compared with PSSYEV,for block size is 64;

Block size 64 for PSSYEV routines seem taken more time than the 32 block size. Execution time of the routines becoming closer. Especially, matrix size 1400 shows that PSSYEV nearly takes the same time. Thus, the optimal block size for PSSYEV seems 32, up-to 1500 matrix size in general.



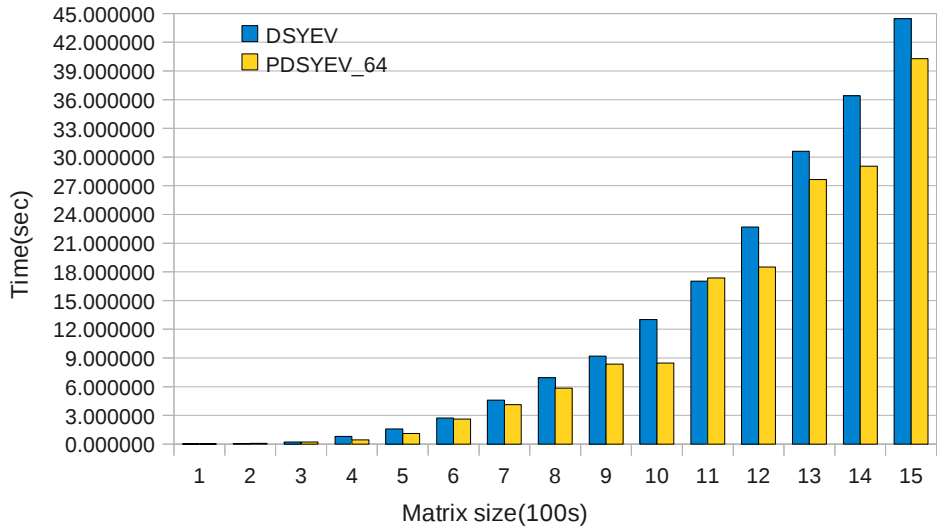
Graph 2:Comparison between the execution time of SSYEV and PSSYEV

Graph 3 shows the execution time of the DSYEV routine and PDSYEV for 32 block size.



Graph 3:Comparison between the execution time of DSYEV and PDSYEV

For double precision routines, first clear difference the execution time of the functions. Last matrix size of 1500, execution time reaches nearly 45 seconds for DSYEV function while PDSYEV is nearly 40 seconds. Graph 1 and Graph 2 show that the highest time has just reached approximately 31 seconds. The next graph, double precision DSYEV and PDSYEV for the block size 64. Nearly, double precision routines take more time to execute than single precision.



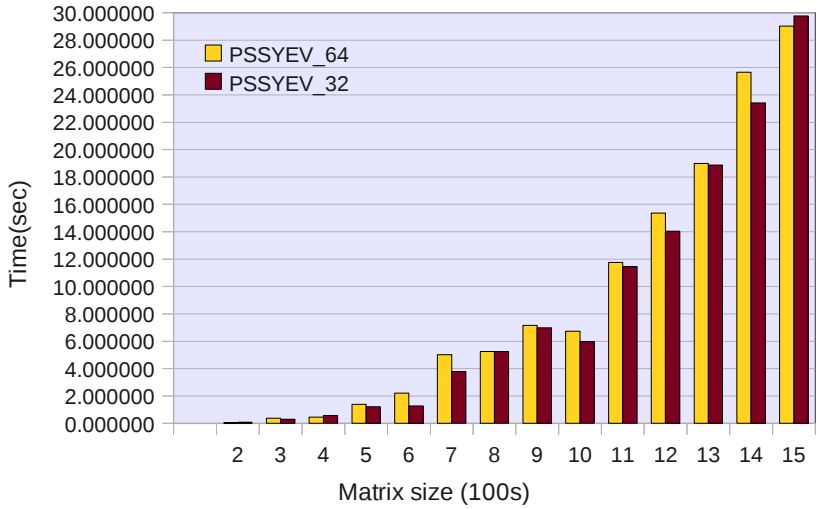
Graph 4:Comparison between the execution time of DSYEV and PDSYEV

Exact values reflected on graphs from our results is shown on the Table 4.1 for large matrices.

Matrix Size	SSYEV	PSSYEV_32	PSSYEV_64	DSYEV	PDSYEV_32	PDSYEV_64
1000	8.654348	5.973473	6.736662	13.026798	9.738768	8.472084
1100	12.090567	11.437959	11.757214	16.126046	17.131805	17.375900
1200	16.360285	14.034716	15.357893	22.677908	21.585387	18.493318
1300	20.610235	18.882274	18.993767	30.613422	26.775023	27.659624
1400	25.891382	23.408131	25.653571	36.405305	31.879645	29.052275
1500	31.181097	29.777485	29.017521	44.465295	39.506742	40.267772

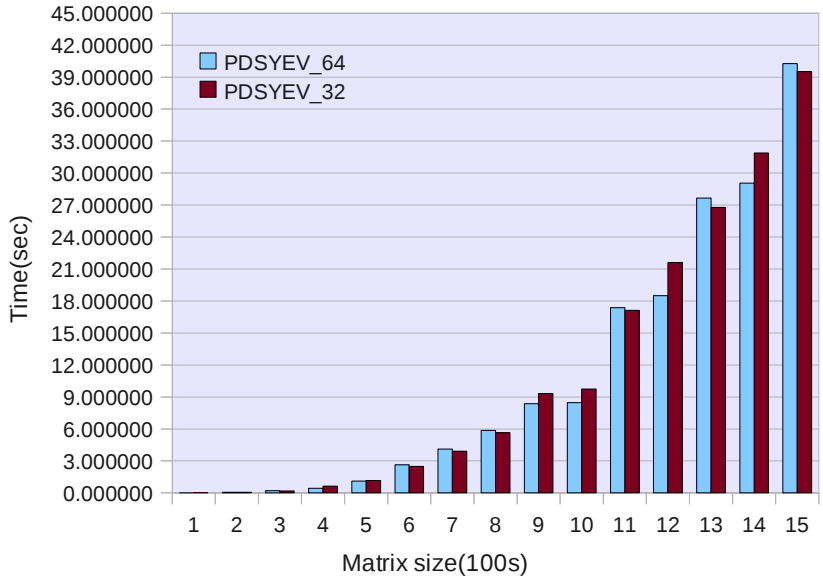
Table 4.1 *Execution time of the routines for the matrix size 1000 to 1500*

Next graph (Graph 5) is just to see how the block size effecting the execution time of the routine PSSYEV. The execution time is mostly decreasing when the block size is chosen 32 for the PSSYEV routines. For the matrix whose block size is 64 taken more time. Still, it is hard to relate the increment or decrement on the block size with matrix size. Our observation, block size depend on distribution of the matrix and also process grid. Because, when matrix size 1500, PSSYEV routine executed in 29 seconds for the block size 32 while execution time is 28 for block size 64.



Graph 5:Comparison between 32 and 64 block size for PSSYEV

Moreover, 32 block size of the PSSYEV faster than 64 block size in overall.



Graph 6:Comparison between 32 and 64 block size for PDSYEV

As we seen from the Graph 6, block size does not seem effective on matrix size. We can not relate with the matrix size up-to 1500.

One important comparison come up with the single and double precision. Some research claim that these is no difference between double and single precision. However, our results show that the execution time of the double precision routines 15 seconds slower than single precision routines. If the storage type of the eigenpairs is double precision, execution time of the routines in-crease, otherwise execution time is 15 seconds less in single precision.

To sum up the results from our findings, we can say execution time of the ScaLAPACK routines faster than LAPACK routines. Although, our expectation was higher than our results, we already know that the simple driver routines of the LAPACK is the slowest it is explained in LAPACK user guide. And ScaLAPACK is not recommended for small matrices if we wish to get efficient results. However, we compare our results with LAPACK benchmark and image taken from

LAPACK User Guide. It is also showed that the DSYEV is the slowest routine to compute all ei-

genpairs of a symmetric matrix.

Figure 3.2: Timings of driver routines for computing all eigenvalues and eigenvectors of a dense symmetric matrix. The upper graph shows times in seconds on an IBM Power3. The lower graph shows times relative to the fastest routine DSYEVR, which appears as a horizontal line at 1.

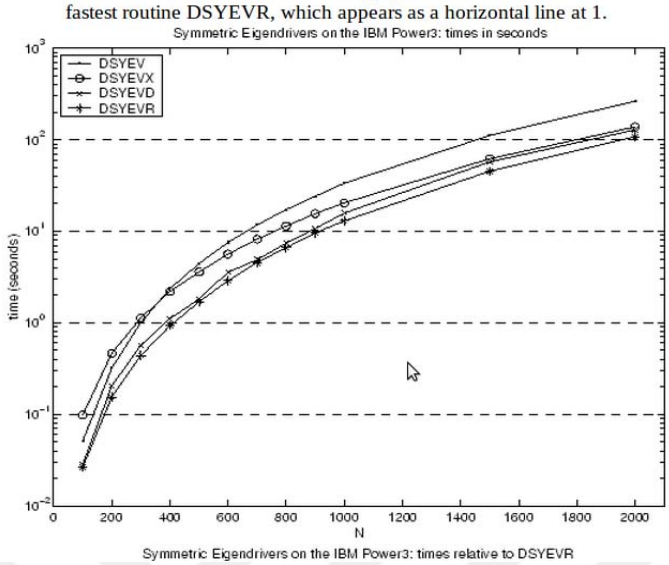


Figure 4.1 *LAPACK Benchmark to compare the driver routine function for finding eigenpairs[6]*

	Process Grid	Block Size	Values of N				
			1000	2000	3000	4000	5000
Cray T3E[a]	1×4	32	38	263			
PSSYEV	2×8	32	15	81	239	552	
	4×16	32	9	34	102	183	366
IBM SP2[b]	1×4	50	49				
PDSYEV	2×8	50	27	121			
	4×16	50	23	75	162	344	
Intel XP/S MP Paragon[c]	1×4	32	189				
PDSYEV	2×8	32	84	645			
	4×16	32	60	344	980		
Cluster of Sun Ultra 2s[d]	2×2	64	81				
PDSYEV	2×4	64	66				
	2×6	64	110	933	2308		
Berkeley NOW[e]	1×4	32	42				
PDSYEV	1×8	32	30				
	2×8	32	19	99	285		
	4×8	32	15	67	271	829	650

Figure 4.2 Execution time in seconds of PSSYEV/PDSYEV for square matrices of order N[7]

It is hard to compare our results from LAPACK and ScaLAPACK standards, because routines of linear algebra packages are implemented high performance computing architectures, workstations, clusters. And portability of linear algebra packages enables machine specific imple-

mentations.

Our results shows that as we increase the matrix size, execution time difference between LAPACK and ScaLAPACK routine increase. So better results can be obtained if we use the large matrix size like 10000 or more. Due to our hardware limitation we are unable to do these tests.

CHAPTER 5

Conclusion

A performance comparison between serial and parallel implementation of the LAPACK and ScaLAPACK routines reflected in this project. The project examined the execution time of the SSYEV and DSYEV routines against PSSYEV and PDSYEV routines to solve eigenvalue and eigenvectors of a symmetric matrix. Our results show that the ScaLAPACK routines take less time in execution of the routines. However, our results are under expectation. One of the reason is the simple driver routines are the slowest routines of the LAPACK to compute all eigenvalues and eigenvectors of a real symmetric matrix that we mentioned in Chapter 4. The important issue about performance of the libraries is the method they implemented to solve eigenvalue problem and each of them uses different methods. The method QR algorithm implemented in simple driver routines. Previous researches generally implemented the routines on high performance computers such as IBM-SP2, CRAY(using more than 8 nodes)[22]. Their results show that the QR algorithm takes more time than the other methods in parallel implementation due to lack of sensitivity to distribution of eigenvalues. This is one of the reason why the simple driver routines are slow. It is ad-

dressed of bottleneck to compute eigenpairs of a symmetric matrix.[22] during the tridioganalization of the symmetric matrices. And another research [21] experimented the parallel implementation of Householder Reduction and back-transformation do not achieve high performance on modern microprocessors. Another reason, QR algorithm use more low level BLAS libraries than other methods, the performance of the Level 1 and Level 2 already less than Level 3, mentioned in Chapter 2.

Also, communication between libraries is another reason which makes slow parallel routines. ScaLAPACK routines required integration of MPI and BLACS libraries while LAPACK routines communicate with BLAS library only.

While examining the performance of execution time of the routines, we discovered that the block size of the routines can make slight difference for the execution time of the routines with the same matrix size. But, the results for block size are varied, it can not be related directly with matrix size. And better results can be obtained, if the suitable block size is chosen for specific matrix size.

Future Work

There are some valuable points which we ignored during this project. They might be;

- The accuracy of the eigenvalues and the eigenvectors can be examined with respect to single and double precision.
- The methods which we implemented has given a good aspect and encourage to go a bit deep finding the time consumed in each step. It will give a good insight to observe the problem partially in each step.

REFERENCES

[1] Evans M. Harrell II, "*A Short History of Operator Theory*", 2004. Last access:10.10.2010. [Online]. http://www.mathphysics.com/opthy/OpHistory.html.

[2] S. Brin and L. Page, "*The anatomy of a large-scale hyper-textual web search engine*", Last access:14.10.2010 [Online]. http://infolab.stanford.edu/~backrub/google.html.

[3] P. K. Sharma, "*Eigenvalues and Eigenvectors and their Applications*", http://www.scribd.com/doc/23290356/Eigenvalues-and-Eigenvectors-and-Their-Applications.

[4] R. A. Tapia, "*Eigenvalues and Eigenvectors*", 2000. Last access:25.11.2010 [Online]. http://www.mathphysics.com/opthy/OpHistory.html.

[5] B. Lang. "*Direct Solvers for Symmetric Eigenvalue Problems*",Modern Methods and Algorithms of Quantum Chemistry, Proceedings. John von Neumann Institute for Computing, J¨ulich,NIC Series, Vol. 3, ISBN 3-00-005834-6, pp. 231-259, 2000.

[6] 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*. Third edition, SIAM, Philadelphia, 1999.

[7] 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 User's Guide"*. A portable linear algebra library for distributed memory computers- design issues and performance. Technical Report TN 37996-1301, University of Tennessee,1996.

[8] Evans M. Harrell II, "*Open Source High Performance Computing*", 2010. [Online]. http://www.open-mpi.org.

[9] J. Choi, J. Dongarra, and D. Walker, PB-BLAS: *A Set of Parallel Block Basic Linear Algebra Subroutines*, Proceedings of Scalable High Performance Computing Conference (Knoxville, TN), pp. 534-541, IEEE Computer Society Press, May 23-25, 1994.

[10]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, R. C. Whaley, *An Updated Set of Basic Linear Algebra Subprograms (BLAS)*, ACM Trans. Math. Soft., 28-2 (2002), pp. 135--151.

[11] J. Dongarra and R. C. Whaley, LAPACK, Working Note 94, *A User's Guide to the BLACS* v1.0, Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, March, 1995.

[12]Canonical Ltd, UBUNTU Linux Launchpad. *Lapack 3.2.2-1 Source Code* [Online]

https://launchpad.net/ubuntu/+source/lapack/3.2.2-1/+build/2000655

[13] Roman, S. " *Advanced Linear Algebra 2nd ed.*", Springer Science Plus Business media, 2005.

[14] Blackford, S."*ScaLAPACK Tutorial*", 1998. [Online]. Last access:17.11.2010 http://www.netlib.org/scalapack/tutorial.

[15] L. S. Blackfordy J. Choiz A. Clearyx J. Demmel, I. Dhillon, J. Dongarrak , S. Hammarling, G. Henry , "*A Portable Linear Algebra Library for Distributed Memory Computers* ", IEEE, 1996.

[16] Whaley, R. C, "*Outstanding issues in MPIBLACS*", IEEE, 1997.

[17] Blackford,S. "*Process grid basics*", 1997. Last access:27.11.2010[online]. http://www.netlib.org/scalapack/slug/node69.html#SECTION04410000000000000000.

[18] Deshpande, V, Sawyer, W. "*An MPI implementation of the BLACS*". Pages 463-468, IEEE, 1996.

[19] Prylli, L. and Tourancheau, B. *"Efficient block cyclic data redistribution"*,in Report de re-cherch No 2766, Janvier, 1996.

[20] D. M. Ritchie. *C Reference Manual*, New Jersey

[21] Bientinesi P., Dhillon I. S., and Robert A. V. "*A Parallel Eigensolver For Dense Symmetric Matrices based on Multiple Relatively Robust Representation* ", SIAM J. SCI. COMPUT. Vol. 27, No. 1, pp. 43–66, 2005.

[22] Tisseur F. and Dongarra J. "*Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem*", IEEE, 1998.

APPENDIX A

1. BLAS QUICK REFERENCE GUIDE
2. ATLAS QUICK REFERENCE GUIDE
3. ScaLAPACK QUICK REFERENCE GUIDE

APPENDIX B

1. **CPU Information**

```
asli@asli-laptop:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 17
model           : 3
model name      : AMD Athlon(tm) X2 Dual-Core QL-62
stepping        : 1
cpu MHz                 : 1000.000
cache size      : 512 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
hlt_bug                 : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush
mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt rdtscp lm 3dnowext 3dnow constant_tsc
nonstop_tsc extd_apicid pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch osvw
skinit
bogomips        : 4000.17
clflush size    : 64
power management: ts ttp tm stc 100mhzsteps hwpstate

processor       : 1
vendor_id       : AuthenticAMD
cpu family      : 17
model           : 3
model name      : AMD Athlon(tm) X2 Dual-Core QL-62
stepping        : 1
cpu MHz                 : 1000.000
cache size      : 512 KB
physical id     : 0
siblings        : 2
```

core id : 1
cpu cores : 2
apicid : 1
initial apicid : 1
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 1
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt rdtscp lm 3dnowext 3dnow constant_tsc nonstop_tsc extd_apicid pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch osvw skinit
bogomips : 3990.37
clflush size : 64
power management: ts ttp tm stc 100mhzsteps hwpstate

asli@asli-laptop:~$

2. ScaLAPACK Python Installer Output

asli@asli-laptop:~/numalg_aso/scalapack_installer_0.96$ ./setup.py --f90=gfortran-4.4 --downblas --downblacs --downlapack --mpiincdir=/usr/include/mpi
==========
Setting up the framework

The Fortran 90/95 compiler is gfortran-4.4
MPI include dir is /usr/include/mpi
Looking for MPI binaries... mpicc and mpif77 found.
mpicc : /usr/bin/mpicc
mpif77 :/usr/bin/mpif77
Checking if mpicc works.../home/asli/numalg_aso/scalapack_installer_0.96/script/utils.py:39: DeprecationWarning: The popen2 module is deprecated. Use the subprocess module.
import popen2
yes
Checking if mpif77 works... yes
Checking if the Fortran 90/95 compiler works... yes
Setting Fortran mangling... -DAdd_
Setting download command...
Checking availability of urllib... available
Testing urllib... not working
Checking availablility of wget... available

```
Testing wget... working
Setting ranlib command... /usr/bin/ranlib
Detecting Fortran 77 compiler... unknown
Detecting C compiler... unknown
Selected C compiler flags: -O3
Selected Fortran77 compiler flags: -O3
Selected loader flags (C main):
Selected loader flags (F77 main):
Selected NOOPT flags:
Checking loader... works

========================================
  BLAS installation/verification
========================================

The reference BLAS library is being installed.
Don't expect high performance from this reference library!
If you want performance, you need to use an optimized BLAS library and,
to avoid unnecessary complications, if you need to compile this optimized BLAS
library, use the same compiler you're using here.
Downloading reference BLAS... done
Unzip and untar reference BLAS... done
Compile and generate reference BLAS... done
Installation of reference BLAS successful.
(log is in  /home/asli/numalg_aso/scalapack_installer_0.96/log/blaslog )

========================================
  Lapack installation/verification
========================================
 Download and install LAPACK from netlib.org
Download LAPACK... done
Unzip and untar LAPACK... done
Installing  lapack-3.2.2 ...
Setting ETIME... INT_CPU_TIME
Writing make.inc... done.
Compiling  lapack-3.2.2  (this will take several minutes)... Installation of LAPACK successful.
(log is in  /home/asli/numalg_aso/scalapack_installer_0.96/log/laplog )

========================================
  BLACS installation/verification
========================================
Downloading BLACS... done
Unzip and untar BLACS... done
Setting TRANSCOMM...  -DCSameF77
```

```
Writing Bmake.inc... done.
Compiling BLACS... done.
 (log is in  /home/asli/numalg_aso/scalapack_installer_0.96/log/blacslog )

========================================
ScaLAPACK installer is starting now. Buckle up!
========================================
Installing  scalapack-1.8.0 ...
Writing SLmake.inc... done.
Compiling ScaLAPACK... Installation of ScaLAPACK successful..
(log is in  /home/asli/numalg_aso/scalapack_installer_0.96/log/scalog )
Compiling test routines...
done
done. ScaLAPACK is installed. Use it in moderation :-)

**************************************************
**************************************************

ScaLAPACK installation completed.


Your BLAS library is:
/home/asli/numalg_aso/scalapack_installer_0.96/lib/librefblas.a

Your BLACS libraries are:
/home/asli/numalg_aso/scalapack_installer_0.96/lib/blacs.a
/home/asli/numalg_aso/scalapack_installer_0.96/lib/blacsC.a
/home/asli/numalg_aso/scalapack_installer_0.96/lib/blacsF77.a

Your LAPACK library is:
/home/asli/numalg_aso/scalapack_installer_0.96/lib/libreflapack.a

Your ScaLAPACK library is:
/home/asli/numalg_aso/scalapack_installer_0.96/lib/libscalapack.a

Log messages are in the
/home/asli/numalg_aso/scalapack_installer_0.96/log
directory.

The Scalapack testing programs are in:
/home/asli/numalg_aso/scalapack_installer_0.96/build/scalapack-1.8.0/TESTING


The
/home/asli/numalg_aso/scalapack_installer_0.96/build
```

directory contains the source code of the libraries
that have been installed. It can be removed at this time.

****
****

asli@asli-laptop:~/numalg_aso/scalapack_installer_0.96$

APPENDIX C

Results

Results shows the execution time of the routines after 100 iterations done.PSSYEV and PD-SYEV results includes block size of 32 and 64.

1. **SSYEV**

Matrix Size	Seconds
100	0.022016
200	0.103595
300	0.137896
400	0.602607
500	1.055239
600	1.152291
700	3.538589
800	4.856286
900	7.262035
1000	9.182487
1100	12.090567
1200	16.360285
1300	20.610235
1400	25.891382
1500	31.181097

2. **PSSYEV**

Block size 32

Matrix Size	Seconds
100	0.022645
200	0.081102
300	0.291512
400	0.570114
500	1.212144
600	1.269146
700	3.782948
800	5.409299
900	6.993681
1000	5.973473
1100	11.437959
1200	14.034716
1300	18.882274
1400	23.408131
1500	29.777485

Block Size 64

Matrix Size	Seconds
100	0.016297
200	0.055213
300	0.374869
400	0.457154
500	1.395363
600	2.199395
700	5.008392
800	5.409299
900	7.154407
1000	6.736662
1100	11.757214
1200	15.357893
1300	18.993767
1400	25.653571
1500	29.017521

3. DSYEV

Matrix Size	Seconds
100	0.010370
200	0.063030
300	0.220097
400	0.818650
500	1.594463
600	2.719973
700	4.592790
800	6.945019
900	9.214669
1000	13.026798
1100	17.031517
1200	22.677908
1300	30.613422
1400	36.405305
1500	44.465295

4. PDSYEV

Block Size 32

Matrix Size	Seconds
100	0.012375
200	0.060182
300	0.171606
400	0.623751
500	1.151656
600	2.486539
700	3.796061
800	5.629110
900	9.316254
1000	9.738768
1100	17.131805
1200	21.585387
1300	26.775023
1400	31.879645
1500	39.506742

Block Size 64

Matrix Size	Seconds
100	0.014021
200	0.064540
300	0.212993
400	0.437140
500	1.113020
600	2.627503
700	4.305181
800	5.710276
900	8.393061
1000	8.472084
1100	17.375900
1200	18.493318
1300	27.659624
1400	29.052275
1500	40.267772

Level 1 BLAS

	dim	scalar	vector	vector	scalars		5-element array		prefixes
SUBROUTINE xROTG (						A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(					D1, D2, A, B,		PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,		C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		X, INCX,	Y, INCY,			PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (	N,		X, INCX,	Y, INCY)				$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (	N,	ALPHA,	X, INCX)					$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (	N,		X, INCX,	Y, INCY)				$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)				$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (	N,		X, INCX,	Y, INCY)				$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (	N,		X, INCX,	Y, INCY)				$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (	N,		X, INCX)					$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
FUNCTION xASUM (	N,		X, INCX)					$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(	N,		X, INCX)					$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\|$ $= max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

	options			dim	b-width	scalar	matrix	vector		scalar	vector	
xGEMV (	TRANS,			M, N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (	TRANS,			M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (	UPLO,			N, K,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (	UPLO,			N,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (	UPLO,			N, K,		ALPHA,	A, LDA,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (	UPLO,			N,		ALPHA,	AP,	X, INCX,		BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (	UPLO,	TRANS,	DIAG,	N, K,			A, LDA,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)				$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (	UPLO,	TRANS,	DIAG,	N,			A, LDA,	X, INCX)				$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z
xTBSV (	UPLO,	TRANS,	DIAG,	N, K,			A, LDA,	X, INCX)				$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z
xTPSV (	UPLO,	TRANS,	DIAG,	N,			AP,	X, INCX)				$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z

	options			dim	scalar	vector		vector		matrix			
xGER (				M, N,	ALPHA,	X, INCX,		Y, INCY,		A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (				M, N,	ALPHA,	X, INCX,		Y, INCY,		A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (				M, N,	ALPHA,	X, INCX,		Y, INCY,		A, LDA)		$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (	UPLO,			N,	ALPHA,	X, INCX,				A, LDA)		$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (	UPLO,			N,	ALPHA,	X, INCX,				AP)		$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (	UPLO,			N,	ALPHA,	X, INCX,		Y, INCY,		A, LDA)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (	UPLO,			N,	ALPHA,	X, INCX,		Y, INCY,		AP)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (	UPLO,			N,	ALPHA,	X, INCX,				A, LDA)		$A \leftarrow \alpha xx^T + A$	S, D
xSPR (	UPLO,			N,	ALPHA,	X, INCX,				AP)		$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (	UPLO,			N,	ALPHA,	X, INCX,		Y, INCY,		A, LDA)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (	UPLO,			N,	ALPHA,	X, INCX,		Y, INCY,		AP)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

	options			dim	scalar	matrix		matrix		scalar	matrix	
xGEMM (		TRANSA,	TRANSB,	M, N, K,	ALPHA,	A, LDA,		B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha op(A) op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM (	SIDE, UPLO,			M, N,	ALPHA,	A, LDA,		B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
xHEMM (	SIDE, UPLO,			M, N,	ALPHA,	A, LDA,		B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK (	UPLO,	TRANS,		N, K,	ALPHA,	A, LDA,				BETA,	C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK (	UPLO,	TRANS,		N, K,	ALPHA,	A, LDA,				BETA,	C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K(	UPLO,	TRANS,		N, K,	ALPHA,	A, LDA,		B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
xHER2K(	UPLO,	TRANS,		N, K,	ALPHA,	A, LDA,		B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
xTRMM (	SIDE, UPLO,	TRANSA,	DIAG,	M, N,	ALPHA,	A, LDA,		B, LDB)				$B \leftarrow \alpha op(A)B, B \leftarrow \alpha B op(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM (	SIDE, UPLO,	TRANSA,	DIAG,	M, N,	ALPHA,	A, LDA,		B, LDB)				$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha B op(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

Meaning of prefixes

S - REAL	C - COMPLEX
D - DOUBLE PRECISION	Z - COMPLEX*16 (this may not be supported by all machines)

For the Level 2 BLAS a set of extended-precision routines with the prefixes ES, ED, EC, EZ may also be available.

Level 1 BLAS

In addition to the listed routines there are two further extended-precision dot product routines DQDOTI and DQDOTA.

Level 2 and Level 3 BLAS

Matrix types:

GE - GEneral	GB - General Band	
SY - SYmmetric	SB - Sym. Band	SP - Sum. Packed
HE - HErmitian	HB - Herm. Band	HP - Herm. Packed
TR - TRiangular	TB - Triang. Band	TP - Triang. Packed

Level 2 and Level 3 BLAS Options
Dummy options arguments are declared as CHARACTER*1 and may be passed as character strings.

TRANx	= '**N**o transpose', '**T**ranspose', '**C**onjugate transpose' (X, X^T, X^H)
UPLO	= '**U**pper triangular', '**L**ower triangular'
DIAG	= '**N**on-unit triangular', '**U**nit triangular'
SIDE	= '**L**eft', '**R**ight' (A or op(A) on the left, or A or op(A) on the right)

For real matrices, TRANSx = 'T' and TRANSx = 'C' have the same meaning.
For Hermitian matrices, TRANSx = 'T' is not allowed.
For complex symmetric matrices, TRANSx = 'H' is not allowed.

References

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. on Math. Soft.* 5 (1979) 308-325

J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* 14,1 (1988) 1-32

J.J. Dongarra, I. Duff, J. DuCroz, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. on Math. Soft.* (1989)

Obtaining the Software via `netlib@ornl.gov`

To receive a copy of the single-precision software, type in a mail message:

```
send sblas from blas
send sblas2 from blas
send sblas3 from blas
```

To receive a copy of the double-precision software, type in a mail message:

```
send dblas from blas
send dblas2 from blas
send dblas3 from blas
```

To receive a copy of the complex single-precision software, type in a mail message:

```
send cblas from blas
send cblas2 from blas
send cblas3 from blas
```

To receive a copy of the complex double-precision software, type in a mail message:

```
send zblas from blas
send zblas2 from blas
send zblas3 from blas
```

Send comments and questions to `lapack@cs.utk.edu` .

Basic Linear Algebra Subprograms

A Quick Reference Guide

University of Tennessee
Oak Ridge National Laboratory
Numerical Algorithms Group Ltd.

May 11, 1997

ATLAS ANSI/ISO C LAPACK API REFERENCE

ROUTINE	(ARGUMENTS)	DESCRIPTION	PREFIXES
int clapack_◇gesv	(const enum CBLAS_ORDER Order, const int N, const int NRHS, TYPE *A, const int lda, int *ipiv, TYPE *B, const int ldb)	using $AP = LU$, $B \leftarrow A^{-1}B$, $A \leftarrow LU$, $ipiv \leftarrow P$ (U is unit diagonal, P pivots columns)	S, D, C, Z
int clapack_◇getrf	(const enum CBLAS_ORDER Order, const int M, const int N, TYPE *A, const int lda, int *ipiv)	using $AP = LU$, $A \leftarrow LU$, $ipiv \leftarrow P$ (U is unit diagonal, P pivots columns)	S, D, C, Z
int clapack_◇getrs	(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE Trans, const int N, const int NRHS, const TYPE *A, const int lda, const int *ipiv, TYPE *B, const int ldb)	$B \leftarrow op(A)^{-1}B$, assuming $A = LU$, $ipiv = P$, $op(X) = X, X^T, X^H$	S, D, C, Z
int clapack_◇getri	(const enum CBLAS_ORDER Order, const int N, TYPE *A, const int lda, const int *ipiv)	$A \leftarrow A^{-1}$, assuming on entry $A = LU$, $ipiv = P$	S, D, C, Z
int clapack_◇posv	(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const int N, const int NRHS, TYPE *A, const int lda, TYPE *B, const int ldb)	$B \leftarrow A^{-1}B$, using $A \leftarrow U^TU$ or $A \leftarrow LL^T$ or $A \leftarrow U^HU$ or $A \leftarrow LL^H$	S, D, C, Z
int clapack_◇potrf	(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const int N, TYPE *A, const int lda)	$A \leftarrow U^TU$ or $A \leftarrow LL^T$ or $A \leftarrow U^HU$ or $A \leftarrow LL^H$	S, D, C, Z
int clapack_◇potrs	(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const int N, const int NRHS, const TYPE *A, const int lda, TYPE *B, const int ldb)	$B \leftarrow op(A)^{-1}B$, assuming $A = U^TU$ or $A = LL^T$ or $A = U^HU$ or $A = LL^H$	S, D, C, Z
int clapack_◇potri	(const enum CBLAS_ORDER Order, const enum ATLAS_UPLO Uplo, const int N, TYPE *A, const int lda)	$A \leftarrow A^{-1}$, assuming on entry $A = U^TU$ or $A = LL^T$ or $A = U^HU$ or $A = LL^H$	S, D, C, Z
int clapack_◇lauum	(const enum ATLAS_ORDER Order, const enum ATLAS_UPLO Uplo, const int N, TYPE *A, const int lda)	$A \leftarrow UU^H$ or $A \leftarrow L^HL$	S, D, C, Z
int clapack_◇trtri	(const enum ATLAS_ORDER Order, const enum ATLAS_UPLO Uplo, const enum ATLAS_DIAG Diag, const int N, TYPE *A, const int lda)	$A \leftarrow A^{-1}$, given A is an Upper or Lower triangular matrix	S, D, C, Z

NOTES:

- C interface DESCRIPTIONs assume `Order == CblasRowMajor`. For column-major descriptions, consult the Fortran77 descriptions.
- All C functions return LAPACK's `INFO` parameter
- C Calling routines should include the BLAS header file, `cblas.h`.
- Cases seperated by *or* above depend on user input or data type.
- More information available at `http://math-atlas.sourceforge.net/`.

PREFIX RELATED DEFINITIONS :

◇is	Data operated	TYPE	UTYPE	SCALAR
s	single precision real	float	float	const float
d	double precision real	double	double	const double
c	single precision complex	void	float	const void*
z	double precision complex	void	double	const void*

ATLAS FORTRAN77 LAPACK API REFERENCE

SUBROUTINE	(ARGUMENTS)	DESCRIPTION	PREFIXES
♢GESV	(N, NRHS, A, LDA, IPIV, B, LDB, INFO)	using $PA = LU$, $B \leftarrow A^{-1}B$, $A \leftarrow LU$, $IPIV \leftarrow P$ (L is unit diagonal, P pivots rows)	S, D, C, Z
♢GETRF	(M, N, A, LDA, IPIV, INFO)	using $PA = LU$, $A \leftarrow LU$, $ipiv \leftarrow P$ (L is unit diagonal, P pivots rows)	S, D, C, Z
♢GETRS	(TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO)	$B \leftarrow op(A)^{-1}B$, assuming $A = LU$, $ipiv = P$, $op(X) = X, X^T, X^H$	S, D, C, Z
♢GETRI	(N, A, LDA, IPIV, WORK, LWORK, INFO)	$A \leftarrow A^{-1}$, assuming $A = LU$, $ipiv = P$	S, D, C, Z
♢POSV	(UPLO, N, NRHS, A, LDA, B, LDB, INFO)	$B \leftarrow A^{-1}B$, using $A \leftarrow U^TU$ or $A \leftarrow LL^T$ or $A \leftarrow U^HU$ or $A \leftarrow LL^H$	S, D, C, Z
♢POTRF	(UPLO, N, A, LDA, INFO)	$A \leftarrow U^TU$ or $A \leftarrow LL^T$ or $A \leftarrow U^HU$ or $A \leftarrow LL^H$	S, D, C, Z
♢POTRS	(UPLO, N, NRHS, A, LDA, B, LDB, INFO)	$B \leftarrow op(A)^{-1}B$, assuming $A = U^TU$ or $A = LL^T$ or $A = U^HU$ or $A = LL^H$	S, D, C, Z
♢POTRI	(UPLO, N, A, LDA, INFO)	$B \leftarrow op(A)^{-1}B$, assuming $A = U^TU$ or $A = LL^T$ or $A = U^HU$ or $A = LL^H$	S, D, C, Z
♢LAUUM	(UPLO, N, A, LDA, INFO)	$A \leftarrow UU^H$ or $A \leftarrow L^HL$	S, D, C, Z
♢TRTRI	(UPLO, DIAG, N, A, LDA, INFO)	$A \leftarrow A^{-1}$, given A is an Upper or Lower triangular matrix	S, D, C, Z

ScaLAPACK

Quick

Reference

Guide

to the

Driver

Routines

Release 1.5

Simple Drivers

Simple Driver Routines for Linear Equations

Matrix Type	Routine
General	PSGESV(N, NRHS, A, IA, JA, IPIV, B, IB, JB, INFO)
	PCGESV(N, NRHS, A, IA, JA, IPIV, B, IB, JB, INFO)
General Band (no pivoting)	PSDBSV(N, BWL, BWU, NRHS, A, JA, DESCA, B, IB, DESCB, WORK,LWORK, INFO)
	PCDBSV(N, BWL, BWU, NRHS, A, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
General Band (partial pivoting)	PSGBSV(N, BWL, BWU, NRHS, A, JA, DESCA, IPIV, B, IB, DESCB, WORK, LWORK, INFO)
	PCGBSV(N, BWL, BWU, NRHS, A, JA, DESCA, IPIV, B, IB, DESCB, WORK, LWORK, INFO)
General Tridiagonal (no pivoting)	PSDTSV(N, NRHS, DL, D, DU, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
	PCDTSV(N, NRHS, DL, D, DU, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
Symmetric/Hermitian Positive Definite	PSPOSV(UPLO, N, NRHS, A, IA, JA, DESCA, B, IB, JB, DESCB, INFO)
	PCPOSV(UPLO, N, NRHS, A, IA, JA, DESCA, B, IB, JB, DESCB, INFO)
Symmetric/Hermitian Positive Definite Band	PSPBSV(UPLO, N, BW, NRHS, A, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
	PCPBSV(UPLO, N, BW, NRHS, A, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
Symmetric/Hermitian Positive Definite Tridiagonal	PSPTSV(N, NRHS, D, E, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)
	PCPTSV(N, NRHS, D, E, JA, DESCA, B, IB, DESCB, WORK, LWORK, INFO)

Simple Driver Routines for Standard and Generalized Linear Least Squares Problems

Problem Type	Routine
Solve Using Orthogonal Factor, Assuming Full Rank	PSGELS(TRANS, M, N, NRHS, A, IA, JA, DESCA, B, IB, JB, DESCB, WORK, LWORK, INFO)
	PCGELS(TRANS, M, N, NRHS, A, IA, JA, DESCA, B, IB, JB, DESCB, WORK, LWORK, INFO)

Simple Driver Routines for Standard Eigenvalue and Singular Value Problems

Matrix/Problem Type	Routine
Symmetric/Hermitian Eigenvalues/vectors	PSSYEV(JOBZ, UPLO, N, A, IA, JA, DESCA, W, Z, IZ, JZ, DESCZ, WORK, LWORK, INFO)
General Singular Values/Vectors	PSGESVD(JOBU, JOBVT, M, N, A, IA, JA, DESCA, S, U, IU, JU, DESCU, VT, IVT, JVT, DESCVT, WORK, LWORK, INFO)

Expert Drivers

Expert Driver Routines for Linear Equations

Matrix Type	Routine
General	PSGESVX(FACT, TRANS, N, NRHS, A, IA, JA, DESCA, AF, IAF, JAF, DESCAF, IPIV, EQUED, R, C, B, IB, JB, DESCB, X, IX, JX, DESCX, RCOND, FERR, BERR, WORK, LWORK, IWORK, LIWORK, INFO)
	PCGESVX(FACT, TRANS, N, NRHS, A, IA, JA, DESCA, AF, IAF, JAF, DESCAF, IPIV, EQUED, R, C, B, IB, JB, DESCB, X, IX, JX, DESCX, RCOND, FERR, BERR, WORK, LWORK, RWORK, LRWORK, INFO)
Symmetric/Hermitian Positive Definite	PSPOSVX(FACT, UPLO, N, NRHS, A, IA, JA, DESCA, AF, IAF, JAF, DESCAF, EQUED, S, B, IB, JB, DESCB, X, IX, JX, DESCX, RCOND, FERR, BERR, WORK, LWORK, IWORK, LIWORK, INFO)
	PCPOSVX(FACT, UPLO, N, NRHS, A, IA, JA, DESCA, AF, IAF, JAF, DESCAF, EQUED, S, B, IB, JB, DESCB, X, IX, JX, DESCX, RCOND, FERR, BERR, WORK, LWORK, RWORK, LRWORK, INFO)

Expert Driver Routines for Standard and Generalized Symmetric Eigenvalue Problems

Matrix/Problem Type	Routine
Symmetric Eigenvalues/vectors	PSSYEVX(JOBZ, RANGE, UPLO, N, A, IA, JA, DESCA, VL, VU, IL, IU, ABSTOL, M, NZ, W, ORFAC, Z, IZ, JZ, DESCZ, WORK, LWORK, IWORK, LIWORK, IFAIL, ICLUSTR, GAP, INFO)
Hermitian Eigenvalues/vectors	PCHEEVX(JOBZ, RANGE, UPLO, N, A, IA, JA, DESCA, VL, VU, IL, IU, ABSTOL, M, NZ, W, ORFAC, Z, IZ, JZ, DESCZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, IFAIL, ICLUSTR, GAP, INFO)
Symmetric Eigenvalues/vectors	PSSYGVX(IBTYPE, JOBZ, RANGE, UPLO, N, A, IA, JA, DESCA, B, IB, JB, DESCB, VL, VU, IL, IU, ABSTOL, M, NZ, W, ORFAC, Z, IZ, JZ, DESCZ, WORK, LWORK, IWORK, LIWORK, IFAIL, ICLUSTR, GAP, INFO)
Hermitian Eigenvalues/vectors	PCHEGVX(IBTYPE, JOBZ, RANGE, UPLO, N, A, IA, JA, DESCA, B, IB, JB, DESCB, VL, VU, IL, IU, ABSTOL, M, NZ, W, ORFAC, Z, IZ, JZ, DESCZ, WORK, LWORK, RWORK, LRWORK, IWORK, LIWORK, IFAIL, ICLUSTR, GAP, INFO)

Meaning of prefixes

Routines beginning with "PS" are available in:

PS - REAL
PD - DOUBLE PRECISION

Routines beginning with "PC" are available in:

PC - COMPLEX
PZ - COMPLEX*16

Note: COMPLEX*16 may not be supported by all machines