

PLASMA Users' Guide

Parallel Linear Algebra Software for Multicore Architectures

Version 2.0

Electrical Engineering and Computer Science
University of Tennessee

Electrical Engineering and Computer Science
University California at Berkeley

Mathematical & Statistical Sciences
University of Colorado Denver

Contents

1	Essentials	1
1.1	PLASMA	1
1.2	Problems that PLASMA Can Solve	2
1.3	Computers for which PLASMA is Suitable	2
1.4	PLASMA vs. LAPACK and ScaLAPACK	2
1.5	PLASMA and the BLAS	3
1.6	Availability of PLASMA	3
1.7	Commercial Use of PLASMA	4
1.8	Installation of PLASMA	4
1.9	Documentation of PLASMA	4
1.10	Support for PLASMA	5
2	Contents of PLASMA	6
3	Use of PLASMA and Examples	7
3.1	Examples	7
3.2	PLASMA_dgesv example	7
4	Performance of PLASMA	11
4.1	A library for multi-core architectures	11
4.2	Comparison to other libraries	12
4.3	Tuning - Howto	14
5	Accuracy and Stability	15
5.1	Notations	15
5.2	Peculiarity of the error analysis of the tile algorithms	16

5.3	Tile Cholesky factorization	16
5.4	Tile Householder QR factorization	17
5.5	Tile LU factorization	17
6	Installing PLASMA	19
6.1	Getting the PLASMA Installer	19
6.2	PLASMA Installer flags	20
6.3	PLASMA Installer Usage	21
6.4	PLASMA Installer Support	22
6.5	Tips and Tricks	22
6.5.1	Testings are slow	22
6.5.2	For Mac	22
6.5.3	Processors with Hyper-threading	22
6.5.4	Problem with download	22
6.6	PLASMA under Windows	23
7	Troubleshooting	24

Preface

PLASMA version 1.0 was released in November 2008 as a prototype software providing *proof-of-concept* implementation of a linear equations solver based on LU factorization, SPD linear equations solver based on Cholesky factorization and least squares problem solver based on QR and LQ factorizations, with support for real arithmetic in double precision only. The publication of this users's guide coincides with the release of version 2.0 of the PLASMA software, which extends the functionality and robustness of the software by introducing the following features:

Multiple Precisions Support: Real arithmetic and complex arithmetic are supported in both single precision and double precision. Only complex-double code is developed. complex-single, real-double and real-single code is automatically generated from the complex-double *reference* implementation.

LAPACK Interface and Native Interface: All computational routines are available in two versions. One accepts input matrices in LAPACK column-major layout, second one accepts input matrices in tile layout, which is the native layout for PLASMA. The first option provides for convenience at the expense of translation overhead, the second one provides increased performance at the cost of ease of use. Conversion routines between the LAPACK layout and the tile layout are provided.

LAPACK-Compliant Error Handling: The error codes returned by the computational routines with LAPACK interface follow LAPACK convention for both numerical errors, related to the numerical properties of the input matrices, as well as errors caused by illegal values of input parameters.

LAPACK-Derived Testing Suite: The software is accompanied by a test suite derived from the one of LAPACK, which in an automated fashion generates thousands of calls, testing the software’s behavior under normal conditions, as well as in the presence of illegal arguments and numerically deficient matrices.

Convenient Workspace Allocation: Whenever possible, PLASMA allocates workspaces internally. In situations, where workspaces serve the purpose of passing data from one routine to another, e.g., from a factorization routine to a solve routine, PLASMA provides straightforward helper routines to take care of the allocations.

Thread Safety: PLASMA is thread safe, which means that multiple instances of PLASMA can co-exist within the address space of a single process. In other words, a single process can create multiple threads and each of these threads can use an independent instance and change its settings without conflicts with other threads.

Mixed-Precision Solver Prototype: A prototype of a mixed-precision solver based on the LU factorization is included. Currently, the implementation is not optimized for performance and does not provide performance benefits over the fixed-precision algorithm. It is only included to demonstrate the technique.

Currently, the PLASMA project is funded by the National Science Foundation through grants #CCF-0811642 and #CCF-0811520 and by the Microsoft Corporation.

A major effort is underway to implement PLASMA-type algorithms for hybrid systems equipped with accelerator devices, GPUs and alike.

The PLASMA team would like to thank the following people, who have significant contributions to the PLASMA project:

Alfredo Buttari.

CHAPTER 1

Essentials

1.1 PLASMA

PLASMA is a software library, currently implemented using the FORTRAN and C programming languages, and providing interfaces for FORTRAN and C. It has been designed to be efficient on *homogeneous* multicore processors and multi-socket systems of multicore processors. The name PLASMA is an acronym for *Parallel Linear Algebra Software for Multicore Architectures*.

PLASMA project website is located at:

<http://icl.cs.utk.edu/plasma>

PLASMA software can be downloaded from:

<http://icl.cs.utk.edu/plasma/software/>

PLASMA users' forum is located at:

<http://icl.cs.utk.edu/plasma/forum/>

and can be used to post general questions and comments as well as to report technical problems.

1.2 Problems that PLASMA Can Solve

PLASMA can solve dense systems of linear equations and linear least squares problems and associated computations such as matrix factorizations. Unlike LAPACK, currently PLASMA does not solve eigenvalue or singular value problems and does not support band matrices. Similarly to LAPACK, PLASMA does not support general sparse matrices. For all supported types of computation the same functionality is provided for real and complex matrices in single precision and double precision.

1.3 Computers for which PLASMA is Suitable

PLASMA is designed to give high efficiency on homogeneous multicore processors and multi-socket systems of multicore processors. As of today, majority of such systems are on-chip symmetric multiprocessors with classic *super-scalar* processors as their building blocks (x86 and alike) augmented with short-vector SIMD extensions (SSE and alike). A software project parallel to PLASMA, MAGMA, is being developed to address the needs of heterogeneous (hybrid) systems, equipped with a hardware accelerator, such as GPUs:

<http://icl.cs.utk.edu/magma>

The name MAGMA is an acronym for Matrix Algebra on GPU and Multicore Architectures.

1.4 PLASMA vs. LAPACK and ScaLAPACK

PLASMA has been designed to supercede LAPACK and ScaLAPACK, principally by restructuring the software to achieve much greater efficiency, where possible, on modern computers based on multicore processors. PLASMA also relies on new or improved algorithms.

Currently, PLASMA does not serve as a complete replacement of LAPACK due to limited functionality. Specifically, PLASMA does not support band matrices and does not solve eigenvalue and singular value problems. Also, PLASMA does not replace ScaLAPACK as software for distributed memory computers, since it only supports shared-memory machines.

1.5 PLASMA and the BLAS

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subroutines (BLAS). Highly efficient machine-specific implementations of the BLAS are available for most modern processors, including multi-threaded implementations.

The parallel algorithms in PLASMA are built using a small set of sequential routines as building blocks. These routines are referred to as *core BLAS*. Ideally, these routines would be implemented through monolithic machine-specific code, utilizing to the maximum a single processing core (through the use of short-vector SIMD extensions and appropriate cache and register blocking).

However, such implementations are extremely labor-intensive and covering the entire spectrum of available architectures is not feasible. Instead, the core BLAS routines are built, in a somewhat suboptimal fashion, by using the “standard” BLAS routines as building blocks. For that reason, PLASMA, same as LAPACK, requires a highly optimized implementation of the BLAS in order to deliver good performance.

Although the BLAS are not strictly speaking part of neither PLASMA nor LAPACK, FORTRAN code for the BLAS is distributed with LAPACK, or can be obtained separately from Netlib:

<http://www.netlib.org/blas/blas.tgz>

However, it has to be emphasized that this code is only the “model implementation” (the definition of the BLAS) and cannot be expected to deliver good performance. On most of today’s machines it will deliver performance an order of magnitude lower than that of optimized BLAS.

For information on available optimized BLAS libraries, as well as other BLAS-related questions, please refer to the BLAS FAQ:

<http://www.netlib.org/blas/faq.html>

1.6 Availability of PLASMA

PLASMA is distributed in source code and is, for the most part, meant to be compiled from source on the host system. In certain cases, a pre-built binary may be provided along with the source code. Such packages, built by the PLASMA developers, will be provided as separate archives on the PLASMA download page:

<http://icl.cs.utk.edu/plasma/software/>

The PLASMA team does not reserve exclusive right to provide such packages. They can be provided by other individuals or institutions. However, in case of problems with binary distributions acquired from other places, the provider needs to be asked for support rather than PLASMA developers.

1.7 Commercial Use of PLASMA

PLASMA is a freely available software package. Thus it can be included in commercial packages. The PLASMA team asks only that proper credit be given by citing this users' guide as the official reference for PLASMA.

Like all software, this package is copyrighted. It is not trademarked. However, if modifications are made that affect the interface, functionality, or accuracy of the resulting software, the name of the routine should be changed. Any modifications to the software should be noted in the modifier's documentation.

PLASMA team will gladly answer questions regarding this software. If modifications are made to the software, however, it is the responsibility of the individual or institution who modified the routine to provide support.

1.8 Installation of PLASMA

A PLASMA installer is available at:

<http://icl.cs.utk.edu/plasma/software/>

Further details are provided in the chapter 6 Installing PLASMA.

1.9 Documentation of PLASMA

PLASMA package comes with several pdf and html documentation. Available for the users are:

- the PLASMA users guide (this document)
- the PLASMA README
- the PLASMA Installation Guide
- the PLASMA Routine Description

- the PLASMA and Tau Guide
- the PLASMA Routine browsing

You will find all of those on the PLASMA documentation section from the PLASMA website <http://icl.cs.utk.edu/plasma>.

1.10 Support for PLASMA

PLASMA has been thoroughly tested before release, on multiple different types of computers. The PLASMA project supports the package in the sense that reports of errors or poor performance will gain immediate attention from the developers. Such reports – and also descriptions of interesting applications and other comments – should be posted to the PLASMA users' forum:

<http://icl.cs.utk.edu/plasma/forum/>

CHAPTER 2

Contents of PLASMA

PLASMA provides routines to solve dense general systems of linear equations, symmetric positive definite systems of linear equations and linear least squares problems, using LU, Cholesky, QR and LQ factorizations. Real arithmetic and complex arithmetic are supported in both single precision and double precision.

PLASMA has been designed to supercede LAPACK and ScaLAPACK, principally by restructuring the software to achieve much greater efficiency, where possible, on modern computers based on multicore processors. PLASMA also relies on new or improved algorithms. Currently, however, PLASMA does not serve as a complete replacement of LAPACK due to limited functionality. Specifically, PLASMA does not support band matrices and does not solve eigenvalue and singular value problems. Also, PLASMA does not replace ScaLAPACK as software for distributed memory computers, since it only supports shared-memory machines.

CHAPTER 3

Use of PLASMA and Examples

3.1 Examples

In the `./examples` directory, you will find simple example codes to call the `PLASMA_[sdcz]gels`, `PLASMA_[sdcz]gesv` and `PLASMA_[sdcz]posv` routines from a C program or from a Fortran program. In this section we explain further the necessary steps for a correct use of these PLASMA routines.

3.2 `PLASMA_dgesv` example

Before calling the PLASMA routine `PLASMA_dgesv`, some initializations are required.

Firstly, we set the dimension of the problem $Ax = B$. In our example, the coefficient matrix A is 10-by-10, and the right-hand side matrix B 10-by-5. We also allocate the memory space required for these two matrices.

```
in C:
    int N = 10;
    int NRHS = 5;
    double *A = (double *)malloc(N*N*sizeof(double));
```

```
double *B = (double *)malloc(N*NRHS*sizeof(double));  
in Fortran:  
    INTEGER          N, NRHS  
    PARAMETER        ( N = 10 )  
    PARAMETER        ( NRHS = 5 )  
    DOUBLE PRECISION A( N, N ), B( N, NRHS )
```

Secondly, we initialize the matrix *A* and *B* with random values. Since we cruelly lack imagination, we use the LAPACK function `dlarnv` for this task. For a starter, you are welcome to change the values in the matrix. Remember that the interface of `PLASMA_dgesv` uses column major format.

```
in C:  
    int IONE=1;  
    int ISEED[4] = {0,0,0,1}; /* initial seed for dlarnv() */  
    /* Initialize A */  
    dlarnv(&IONE, ISEED, &NxN, A);  
    /* Initialize B */  
    dlarnv(&IONE, ISEED, &NxNRHS, B);  
in Fortran:  
    INTEGER          I  
    INTEGER          ISEED( 4 )  
    EXTERNAL         DLARNV  
    DO I = 1, 4  
        ISEED( I ) = 1  
    ENDDO  
!-- Initialization of the matrix  
    CALL DLARNV( 1, ISEED, N*N, A )  
  
!-- Initialization of the RHS  
    CALL DLARNV( 1, ISEED, N*NRHS, B )
```

Thirdly, we initialize `PLASMA` by calling the `PLASMA_Init` routine. The variable `cores` specifies the number of cores `PLASMA` will use.

```
in C:  
    int cores = 2;  
    /* Plasma Initialize */  
    INFO = PLASMA_Init(cores);  
in Fortran:  
    INTEGER          CORES  
    PARAMETER        ( CORES = 2 )
```

```
INTEGER      INFO
EXTERNAL     PLASMA_INIT
! -- Initialize Plasma
CALL PLASMA_INIT( CORES, INFO )
```

Before we can call `PLASMA_dgesv`, we need to allocate some workspace necessary for the `PLASMA_dgesv` routine to operate. In `PLASMA`, each routine has its own routine to allocate its specific workspace arrays. Those routines are all defined in the `workspace.c` file. Their names are of the kind `PLASMA_Alloc_Workspace_` with the name of the associated routine in lower case for C and `PLASMA_ALLOC_WORKSPACE_` with the name of the associated routine upper case for Fortran. You will also need to check the interface since they are different from one routine to the other. For the `PLASMA_dgesv` routine, two arrays need to be initialized, the workspace (here it is called `L` in the C code and `HL` in the Fortran Code) and the pivots (here it is called `IPIV` in the C code and `HPIV` in the Fortran Code). Note that in C, you need to use standard pointers; while in Fortran, you need to use handle defines as an array of two elements of `INTEGER*4`.

```
in C:
double *L;
int *IPIV;
PLASMA_Alloc_Workspace_dgesv(N, &L, &IPIV);
in Fortran:
INTEGER*4      HL( 2 ), HPIV( 2 )
EXTERNAL       PLASMA_ALLOC_WORKSPACE_DGESV
! -- Allocate L and IPIV
CALL PLASMA_ALLOC_WORKSPACE_DGESV( N, HL, HPIV, INFO )
```

Finally, we can call the `PLASMA_dgesv` routine. You can report to the header of the routine for a complete description of the arguments. The description is also available online in the `PLASMA` routine description html page: <http://icl.cs.utk.edu/projectsfiles/plasma/html/routine.html>. As in `LAPACK`, `PLASMA` is returning a return variable, `INFO`, that indicates if the exit was successful or not. A successful exit is indicated by `INFO` equal to 0. In a case of `INFO` different from 0, the value of `INFO` should help you to understand the reason of the failure. (See the return value section in the above cited documentation.)

```
in C:
/* Solve the problem */
INFO = PLASMA_dgesv(N, NRHS, A, N, L, IPIV, B, N);
if (INFO < 0)
    printf("-- Error in DGESV example ! \n");
else
```

```

        printf("-- Run successful ! \n");
in Fortran:
! -- Perform the LU solve
    CALL PLASMA_DGESV( N, NRHS, A, N, HL, HPIV, B, N, INFO )
    IF ( INFO < 0 ) THEN
        WRITE(*,*) " -- Error in DGESV example !"
    ELSE
        WRITE(*,*) " -- Run successful !"
    ENDIF

```

Before exiting the program, we need to free the resource used by our arrays and finalize PLASMA. To deallocate the C array, a simple call to `free` is needed whereas in Fortran, PLASMA provides the routine `PLASMA_DEALLOC_HANDLE` to do so. `PLASMA_Finalize` call will finalize PLASMA.

```

in C:
    /* Deallocate L and IPIV */
    free(L); free(IPIV);
    /* Plasma Finalize */
    INFO = PLASMA_Finalize();
in Fortran:
! -- Deallocate L and IPIV
    CALL PLASMA_DEALLOC_HANDLE( HL, INFO )
    CALL PLASMA_DEALLOC_HANDLE( HPIV, INFO )
!-- Finalize Plasma
    CALL PLASMA_FINALIZE( INFO )

```

CHAPTER 4

Performance of PLASMA

4.1 A library for multi-core architectures

To achieve high performance on multi-core architectures, PLASMA relies on tile algorithms, which provide fine granularity parallelism. The standard linear algebra algorithms can then be represented as Directed Acyclic Graphs (DAG) where nodes represent tasks and edges represent dependencies among them. Our programming model enforces asynchronous, out of order scheduling of operations. This concept is used as the basis for a scalable yet highly efficient software framework for computational linear algebra applications.

In LAPACK, parallelism is obtained through the use of multithreaded *Basic Linear Algebra Subprograms* (BLAS). In PLASMA, parallelism is no longer hidden inside the BLAS but is brought to the fore to yield much better performance.

PLASMA performance strongly depends on tunable execution parameters trading off utilization of different system resources. The outer block size (NB) trades off parallelization granularity and scheduling flexibility with single core utilization, while the inner block size (IB) trades off memory load with extra-flops.

PLASMA is currently scheduled statically with a trade off between load balancing and data reuse.

4.2 Comparison to other libraries

We present here the performance of the three following one sided factorizations: Cholesky, QR, and LU. We compare PLASMA against the two established linear algebra packages LAPACK and ScaLAPACK. The experiments were conducted on two different multi-core architectures based on Intel Xeon EMT64 and IBM Power6.

PLASMA, LAPACK and ScaLAPACK are all linked with the optimized vendor BLAS available on the system provided within Intel MKL 10.1 and IBM ESSL 4.3 on the Intel64 and Power6 architectures, respectively. The first architecture is a quad-socket quad-core machine based on an Intel Xeon EMT64 E7340 processor operating at 2.39 GHz. Its theoretical peak is equal to 9.6 Gflop/s/ per core or 153.2 Gflop/s for the whole node (16 cores). The second architecture is a SMP node composed of 16 dual-core Power6 processors. Each dual-core Power6 processor runs at 4.7 GHz, leading to a theoretical peak of 18.8 Gflop/s per core and 601.6 Gflop/s per node (32 cores).

Factorizations are performed in double precision. PLASMA is tuned with the pruned search method as described in [1]. For ScaLAPACK we have tuned the data distribution parameters (p,q,nb) as functions of the number of cores and the matrix size through an exhaustive search. For reference LAPACK, we have been using the default block size (no tuning).

ScaLAPACK and PLASMA interfaces allow the user to provide data distributed on the cores. In our shared-memory multi-core environment, because we do not flush the caches, these libraries have thus the advantage to start the factorization with part of the data distributed on the caches. This is not negligible. For instance, a 8000×8000 double precision matrix can be held distributed on the L3 caches of the 32 cores of a Power6 node.

Figures 4.1, 4.2, 4.3 presents performance for the Cholesky, QR and LU factorizations, respectively.

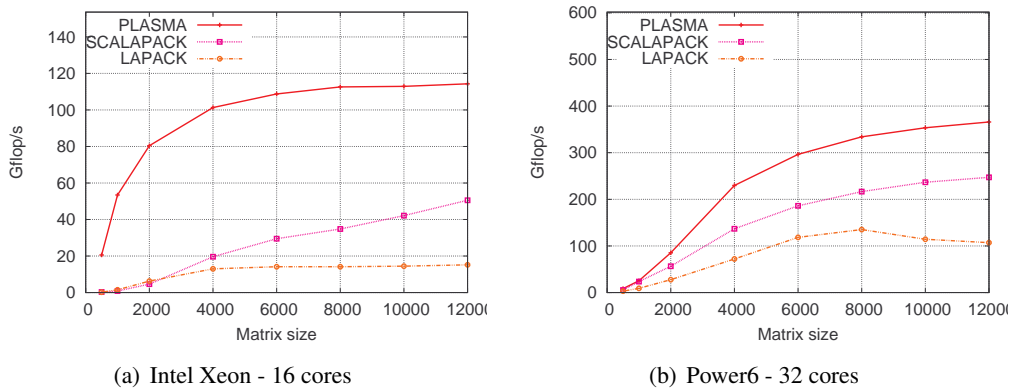


Figure 4.1: Performance of the Cholesky factorization (Gflop/s).

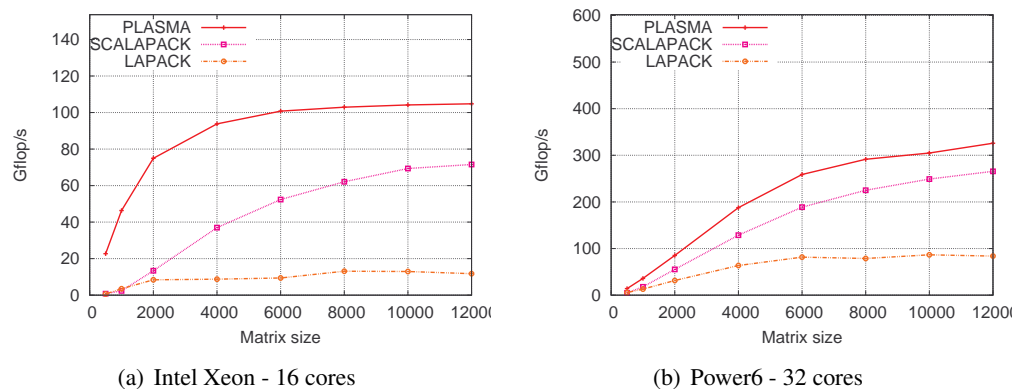


Figure 4.2: Performance of the QR factorization (Gflop/s).

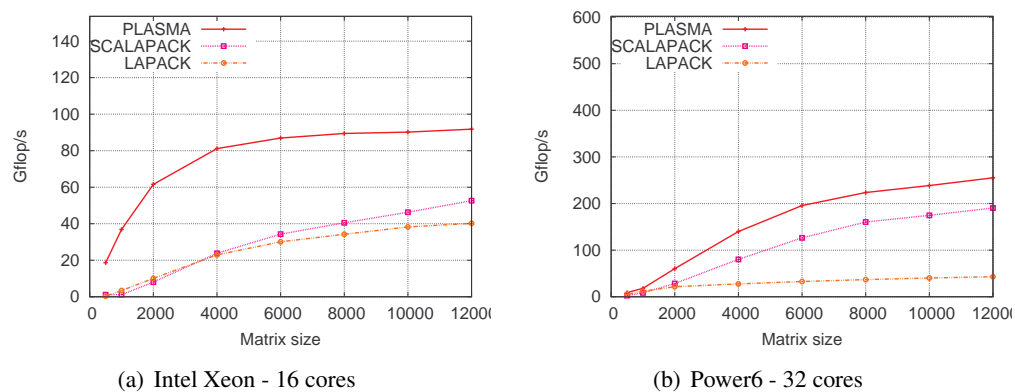


Figure 4.3: Performance of the LU factorization (Gflop/s).

4.3 Tuning - Howto

Users willing to obtain good performance from PLASMA need to tune PLASMA parameters. The example code below illustrates how to change the NB and IB parameters. A pruned search method to obtain “good” parameters is described in [1]. We note that autotuning is part of the PLASMA’s roadmap; unfortunately, as of 2.0.0, the PLASMA software does not have its autotuning component available for release.

```
...
/* Plasma Initialize */
PLASMA_Init(cores);
/* Plasma Tune */
PLASMA_Disable(PLASMA_AUTOTUNING);
PLASMA_Set(PLASMA_TILE_SIZE, NB);
PLASMA_Set(PLASMA_INNER_BLOCK_SIZE, IB);
...
```

CHAPTER 5

Accuracy and Stability

We present backward stability results of the algorithms used in PLASMA. Backward stability is a concept developed by Wilkinson in [8, 9]. For more general discussion on accuracy and stability of numerical algorithms, we refer the users to Higham [7]. We follow Higham's notation and methodology here; indeed we mainly present his results.

5.1 Notations

- We assume the same floating-point system as Higham [7]. His assumptions are fulfilled by the IEEE standard. The unit round-off is u which is about 1.11×10^{-16} for double.
- The absolute value notation $|\cdot|$ is extended from scalar to matrices where matrix $|A|$ denotes the matrix with (i, j) entry $|a_{ij}|$.
- Inequalities between matrices hold componentwise. If we write $A < B$, this means that (A and B are of the same size and) for any i and any j , $a_{ij} < b_{ij}$.
- If $nu < 1$, we define $\gamma_n \equiv \frac{nu}{1-nu}$.
- Computed quantities are represented with an overline or with the $fl(\cdot)$ notation.
- Inequality with $|\cdot|$ can be converted to norms easily. (See [7, Lem.6.6].)

5.2 Peculiarity of the error analysis of the tile algorithms

Numerical linear algebra algorithms relies at their roots on inner products. A widely used result of error analysis of the inner product is given in Theorem 5.2.1

Theorem 5.2.1 (Higham, [7, Eq.(3.5)]) *Given x and y , two vectors of size n , if $x^T y$ is evaluated in floating-point arithmetic, then, no matter what the order of evaluation, we have*

$$|x^T y - fl(x^T y)| \leq \gamma_n |x|^T |y|.$$

While there exists a variety of implementations and interesting research that aim to reduce errors in inner products (see for example [7, chapter 3 and 4]), we note that Theorem 5.2.1 is given independently of the order of evaluation in the inner products. The motivation for being independent of the order of evaluation is that inner products are performed by optimized libraries which uses associativity of the addition for grouping the operations in order to obtain parallelism and data locality in matrix-matrix multiplications.

Theorem 5.2.2 presents a remark from Higham. Higham notes that one can significantly reduce the error bound of an inner product by accumulating it in pieces. (Which is indeed what an optimized BLAS library would do to obtain performance.)

Theorem 5.2.2 (Higham, [7, §3.1]) *Given x and y , two vectors of size n , if $x^T y$ is evaluated in floating-point arithmetic by accumulating the inner product in k pieces of size n/k , then, we have*

$$|x^T y - fl(x^T y)| \leq \gamma_{n/k+k-1} |x|^T |y|.$$

Theorem 5.2.2 has been used by Castaldo, Whaley, and Chronopoulos [5] to improve the error bound in matrix-matrix multiplications.

The peculiarity of the tile algorithm is that they explicitly work on small pieces of data and therefore benefits in general from a *better* error bounds than their LAPACK counterparts.

5.3 Tile Cholesky factorization

The Cholesky factorization algorithm in PLASMA performs the same operations than any version of the Cholesky. The organization of the operations in the inner products might be different from one algorithm to the other. The error analysis is however essentially the same for all the algorithms.

Theorem 5.3.1 presents Higham's result for the Cholesky factorization.

Theorem 5.3.1 (Higham, [7, Th.10.3]) *If Cholesky factorization is applied to the symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$ runs to completion then the computed factors \bar{R} satisfies*

$$\bar{R}^T \bar{R} = A + \Delta A, \quad \text{where } |\Delta A| \leq \gamma_{n+1} |\bar{R}^T| |\bar{R}|.$$

Following Higham's proof, we note that Higham's Equation (10.4) can be tighten in our case since we know that the inner product are accumulated within tiles of size b . The γ_i term becomes $\gamma_{i/b+b-1}$ and we obtain the improved error bound given in Theorem 5.3.2.

Theorem 5.3.2 *If Tile Cholesky factorization is applied to the symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$ runs to completion then the computed factors \bar{R} satisfies*

$$\bar{R}^T \bar{R} = A + \Delta A, \quad \text{where } |\Delta A| \leq \gamma_{n/b+b} |\bar{R}^T| |\bar{R}|.$$

We note that the error bound could be even made smaller by taking into account the inner blocking used in the Cholesky factorization of each tile.

Higham explains how to relate the backward error of the factorization with the backward error of the solution of a symmetric positive definite linear system of equations.

5.4 Tile Householder QR factorization

The *Tile Householder QR* is backward stable since it is obtained through a product of backward stable transformation. One can obtain a tighter error bound with the tile algorithm than the standard

Higham explains how to relate the backward error of the factorization with the backward error of the solution of a linear least squares.

5.5 Tile LU factorization

Theorem 5.5.1 (Bientinesi and van de Geijn, [2, Th.6.5]) *Given $A \in \mathbb{R}^{n \times n}$, assume that the blocked right-looking algorithm in Fig. 6.1 completes. Then the computed factors \bar{L} and \bar{U} are such that*

$$\bar{L}\bar{U} = A + \Delta A, \quad \text{where } |\Delta A| \leq \gamma_{n/b+b} (|A| + |\bar{L}||\bar{U}|).$$

We note that Bientinesi and van de Geijn do not consider permutations.

Higham explains how to relate the backward error of the factorization with the backward error of the solution of a linear system of equations.

Words of cautions: It is important to note that Theorem 5.5.1 does not state that the algorithm is stable. For the algorithm to be stable, we need to have

$$|\overline{L}||\overline{U}| \sim |A|.$$

Whether this is the case or not in the general case is an ongoing research topic. Therefore we recommend users to use `PLASMA_dgesv` with care. In case of doubt, use `PLASMA_dgels`.

CHAPTER 6

Installing PLASMA

In order to install the PLASMA library, you require a C compiler, a Fortran Compiler, a BLAS library and the availability of the pthread library. Before PLASMA can be built, or the tested, you must define all machine-specific parameters for the architecture to which you are installing PLASMA. All machine-specific parameters are contained in the file `make.inc`. We provide you an installer to ease the installation process. Users are strongly encouraged to use it.

6.1 Getting the PLASMA Installer

The PLASMA installer is a set of python scripts developed to ease the installation of the PLASMA library. It can automatically download, configure and compile the PLASMA library including its BLAS dependency.

It is available on PLASMA download page:

<http://icl.cs.utk.edu/plasma/software/>

6.2 PLASMA Installer flags

Here's a list of the flags that can be used to provide the installer information about the system like, for example, the C and Fortran compilers, the BLAS library that is already present on the system and need not be downloaded.

`./setup.py`

<code>-h</code> or <code>--help</code>	: prints this message
<code>--prefix</code>	: path where to create the libraries, build and log of the installer
<code>--cc=[CMD]</code>	: the C compiler.
<code>--fc=[CMD]</code>	: the Fortran compiler.
<code>--ccflags=[FLAGS]</code>	: the flags for the C compiler (default <code>-O3</code>)
<code>--fcflags=[FLAGS]</code>	: the flags for the Fortran compiler (default <code>-O3</code>)
<code>--ldflags_c=[flags]</code>	: loader flags when main program is in C. Some compilers (e.g. PGI) require different options when linking C main programs to Fortran subroutines and vice-versa
<code>--ldflags_fc=[flags]</code>	: loader flags when main program is in Fortran. Some compilers (e.g. PGI) require different options when linking Fortran main programs to C subroutines and vice-versa
<code>--makecmd=[CMD]</code>	: the make command (make by default)
<code>--blaslib=[LIB]</code>	: a BLAS library (path should be absolute if <code>--prefix</code> is used)
<code>--downblas</code>	: if you do not want to provide a BLAS we can download and install it for you

```

--nbcores          : The number of cores to be used by the testing.
                    (2 by default)

--notesting         : disables the PLASMA testing. The BLAS library is not
                    required in this case.

--clean            : cleans up the installer directory.

```

The installer will set

```

OMP_NUM_THREADS=1
GOTO_NUM_THREADS=1
MKL_NUM_THREADS=1

```

to disable the multithreading within BLAS. Do not forget to set those environment variables for any further PLASMA testing you want to run.

6.3 PLASMA Installer Usage

For an installation with **gcc, gfortran and Reference BLAS**

```
./setup.py --cc gcc --fc gfortran --downblas
```

For an installation with **ifort, icc and MKL** (em64t architecture)

```
./setup.py --cc icc --fc ifort --blaslib="-lmkl_em64t -lguide"
```

For an installation with **gcc, gfortran and VecLib** (Mac OS/X)

```
./setup.py --cc gcc --fc gfortran --blaslib="-framework vecLib"
```

For an installation with **gcc, gfortran, ATLAS**

```
./setup.py --cc gcc --fc gfortran --blaslib="-lf77blas -lcblas -latlas"
```

For an installation with **gcc, gfortran, goto BLAS and 4 cores**

```
./setup.py --cc gcc --fc gfortran --blaslib="-lgoto" --nbcores=4
```

For an installation with **xlc, xlf, essl and 8 cores**

```
./setup.py --cc xlc --fc xlf --blaslib="-lessl" --nbcores=8
```

6.4 PLASMA Installer Support

Please note that this is an alpha version and, even if it has been tested on a wide set of systems, may not work. In case you encounter a problem, your feedback would be greatly appreciated and would be very useful to improve the quality of this installer. Please submit your complaints and suggestions to the PLASMA forum:

<http://icl.cs.utk.edu/plasma/forum/>

6.5 Tips and Tricks

6.5.1 Testings are slow

Please note that, in the case where the installer is asked to automatically download and install the BLAS library, the reference BLAS is installed and, thus, very low performance is to be expected. It is strongly recommended that you use an highly optimized BLAS library (like ATLAS, MKL, GotoBLAS, ESSL etc.) and provide a path to its location through the `-blaslib` flag.

6.5.2 For Mac

Veclib is accessible if you install the Xcode/developer package provides with your Mac OS installation CD. On MAC OS/C you may be required to add

```
--ccflags=' -I/usr/include/sys/'
```

6.5.3 Processors with Hyper-threading

The plasma installer cannot detect if you have hyper-threading or not on your machine. We advice always to limit the number of core for the first testing of the PLASMA library to half the numbers of cores available if you do not know your exact architecture. Using hyper-threading will cause the PLASMA testings to hangs.

6.5.4 Problem with download

If the required packages cannot be automatically downloaded (for example, because no network connection is available on the system), you can take them any way you like from

the following URLs and place them in the build directory (if the directory does not exist, create it):

BLAS: <http://netlib.org/blas/blas.tgz>

PLASMA: <http://icl.cs.utk.edu/projectsfiles/plasma/pubs/plasma.tar.gz>

6.6 PLASMA under Windows

The port to Windows has not be done yet, but should be coming shortly...

CHAPTER 7

Troubleshooting

This chapter will be filled along with nonsolved troubleshootings when they occur.

Bibliography

- [1] E. Agullo, B. Hadri, H. Ltaief and J. Dongarra. Comparative study of one-sided factorizations with multiple software packages on multi-core hardware. In the *Proceedings of SC'09: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2009
- [2] P. Bientinesi and R. A. van de Geijn. The science of deriving stability analyses. FLAME Working Note #33. 2009.
- [3] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency Computat.: Pract. Exper.*, 20(13):1573–1590, 2008. <http://dx.doi.org/10.1002/cpe.1301>.
- [4] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35:38–53, 2009. <http://dx.doi.org/10.1016/j.parco.2008.10.002>.
- [5] A. M. Castaldo, R. C. Whaley, and A. T. Chronopoulos. Reducing floating point error in dot product using the superblock family of algorithms. *SIAM J. Sci. Comput.*, 31(2):1156–1174, 2008. <http://dx.doi.org/10.1137/070679946>.
- [6] J. W. Demmel, N. J. Higham, and R. S. Schreiber. Stability of block *LU* factorization. *Numerical Linear Algebra with Applications*, 2(2):173–190, 1995. <http://dx.doi.org/10.1002/nla.1680020208>.
- [7] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*, Second Edition Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0 (hardback).

-
- [8] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32. Her Majesty's Stationery Office, London, 1963. vi+161 pp. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994. ISBN 0-486-67999-3.
- [9] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965. xviii+662 pp. ISBN 0-19-853403-5 (hardback), ISBN 0-19-853418-3 (paperback).