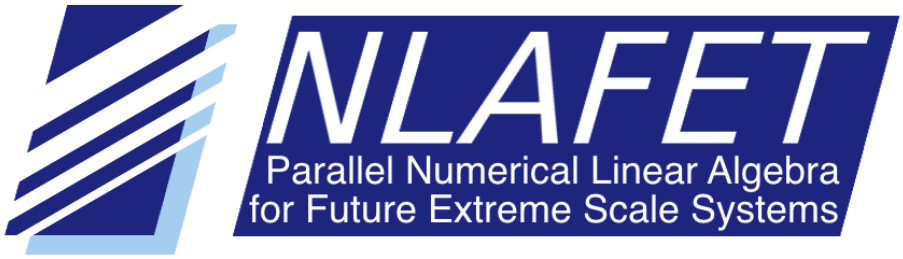

H2020–FETHPC–2014: GA 671633

D7.5

Beta release of the NLAFET library

Prototype software — Part 1

April 2017

Document information

Scheduled delivery	2017-04-30
Actual delivery	2017-04-26
Version	2.0
Responsible partner	UMU

Dissemination level

PU — Public

Revision history

Date	Editor	Status	Ver.	Changes
14/03/2017	Bo Kågström	Draft	0.1	First layout of structure
02/04/2017	Bo Kågström	Draft	1.1	Version for internal reviews
18/04/2017	Bo Kågström	Draft	1.2	Revisions and new material from partners
20/04/2017	Jack Dongarra	Draft	1.3	Revisions for the dense factor and cross-cutting
25/04/2017	Bo Kågström	Final	2.0	Revised for submission

Author(s)

- Bo Kågström, Lars Karlsson, and Carl Christian Kjelgaard Mikkelsen, UMU.
- Iain Duff and Florent Lopez, STFC.
- Samuel Ralton and Jack Dongarra, UNIMAN.
- Simplice Donfack, Alan Ayala, and Laura Grigori, INRIA.

Internal reviewers

- Nick Higham, UNIMAN.
- Authors from the list above have also reviewed parts of the different drafts.

Contributors

Besides the authors of this report, the following members of the NLAFET Team have made important contributions to the software presented and available at GitHub/NLAFET:

- Mirko Myllykoski, Björn Adlerborn, Angelika Schwarz, and Mahmoud Eljammaly, UMU.
- Jonathan Hogg, Stojce Novak, and Vedran Novakovic, STFC.

- Pedro Valero-Lara, Mawussi Zounon, and Negin Bagherpour, UNIMAN.
- Olivier Tissot, INRIA.

We also recognize important contributions by the following collaboration partners.

- The Innovative Computing Laboratory (ICL), the University of Tennessee.
- Jim Demmel at the University of California Berkeley.

Acknowledgements

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under the grant agreement number 671633.

Table of Contents

List of Tables

1 Introduction

The *Description of Action* (DoA) document states for deliverable D7.5:

> "*D7.5 Beta release of the NLAFET library*
>
> Beta release of parts of the NLAFET library and User's Guide."

This deliverable is in the context of Task 7.3 (Open source activities).

1.1 NLAFET repositories on GitHub

The codes developed and distributed in the context of the NLAFET project are hosted on the GitHub platform and can be found using the link `https://github.com/NLAFET/`. GitHub is a web-based application for managing source codes that relies on the version control system Git (`https://git-scm.com/`). In addition to code management, GitHub also provides many important features for software development and code distribution such as the possibility of associating Wikis with every repository. It also includes a set of bug tracking tools helping developers to get feedback from users and to maintain the codes. Although it is possible to freely create an unlimited number of repositories for open-source projects, GitHub also offers the possibility for academic users to create an unlimited number of private repositories without any fees. Therefore in our NLAFET project we can create a repository for each piece of software that we are developing and can manage the access rights with great flexibility. Many of the repositories include several pieces of software that together form a package of routines for a subset of fundamental linear algebra operations considered in the NLAFET project. Typically, such a package is organized in directories or even subdirectories to reflect different types of functionality, tools and solvers.

For a given repository, the main three levels of access rights that we are interested in are:

- Read/Write access for a particular team within the project and no access to any other users.
- Read/Write access for a particular team plus a Read or Read/Write access for another team in the context of a collaboration. By default, members of the projects have Read access to any repository.
- Read/Write access for a particular team or teams and a public Read access when the code is ready to be released.

These examples are the three main options used in the project but these rules might be modified in many ways to allow further collaboration at various testing stages or for potential reviewers.

The two main administrators for the GitHub NLAFET project are Samuel Relton at UNIMAN and Florent Lopez at RAL but all NLAFET partners contribute in keeping the GitHub up to date by managing the repositories they are responsible for. In addition, every team involved in the project can create a repository and manage their own settings. In particular, we will make repositories public whenever the codes are robust enough to be released. In addition, new repositories will be added and occasionally some existing repositories will be merged. Public releases will occur at several times during the project.

We have organized the description of the current repositories according to the following topics:

- Dense matrix factorizations and solvers (Section 2).
- Solvers and tools for standard and generalized dense eigenvalue problems (Section 3).
- Sparse direct factorizations and solvers (Section 4).
- Communication optimal algorithms for iterative methods (Section 5).
- Cross-cutting tools (Section 6).

As of this date, we have 12 repositories in the GitHub NLAFET project. Note that the internal file organization for every repository depends on the team that is managing the repository. However, for most of the repositories the following subdirectories (or obvious equivalents) can be found:

- `src` containing the source code files.
- `test` containing test drivers.
- `data` with a set of inputs for testing the code.
- `results` containing the expected results for the test drivers when using the input provided in the `data` directory.

Also each repository contains a Makefile and a README file (or User Guide) with a description of the software and any special usage and install instructions. After installation, the tests based on the given data can be executed and compared with the expected results.

Finally, in Section 7 we give a brief status report on the ongoing collaboration with our application partners, and in Section 8 we summarize the repositories with current access levels in a table and give an account of our coming software contributions to the NLAFET library.

2 Dense factorizations and solvers

As of this date, the repositories for Batched BLAS and dense factorizations and solvers are:

`BBLAS-ref`: Reference implementation of Batched BLAS routines.

- General matrix multiply, triangular matrix multiply, triangular solve, symmetric rank-k update, symmetric rank-2k update, symmetric matrix multiple and their complex counter parts (xgemm, xtrsm, xtrmm, xsyrk, xsyr2k, xherk, xher2k, xhemm, xsymm).

`plasma`: Snapshot of the PLASMA library for dense linear algebra.

- the one-sided factorizations and solves (LU, Cholesky, QR, LQ, and symmetric indefinite)
- inversion of matrices based on general, symmetric positive definite, symmetric indefinite matrices, and triangular.

- mixed precision LU and Cholesky factorizations and solves for iterative refinement,
- banded LU and triangular factorization and solves,
- bidiagonal factorization and Singular Value Decomposition (SVD),
- utility routines such as computing norms etc.

In the NLAFET/BBLAS-ref repository, we have provided a complete reference implementation of the Batched Level-3 BLAS as part of Deliverable D7.3 (Draft Specification for Hybrid BLAS). In an aim to utilize highly parallel computing resources more efficiently, there is a current trend towards splitting large linear algebra problems into many smaller, perhaps, thousands of smaller subproblems, which can be solved concurrently. The solutions of these smaller problems are then combined to give the solution to the original (large) problem. Examples of this is are in dense linear algebra and deep learning methods which have small independent matrix multiple operations. The solution to these potential inefficiency is to develop a new standard set of routines for carrying out linear algebra operations on batches of small matrices, building on the well-known Basic Linear Algebra Subproblems (BLAS) standard. The idea behind the Batched BLAS (BBLAS) is to perform multiple BLAS operations in parallel on many small matrices, making more efficient use of hardware than a simple for loop would allow.

The Batched BLAS (BBLAS) aims to make more efficient use of hardware resources when multiple small BLAS calls are made simultaneously. Taking matrix multiplication as an example we wish to solve problems of the form

$$C_i = \alpha A_i B_i + \beta C_i, \quad i = 1\colon N.$$

This repository was intended as a starting point for further discussions with vendors such as Intel, ARM, Cray, and NVIDIA to clarify a community-wide standard for Batched BLAS operations. After two workshops and continuing discussions on the subject (see the NLAFET Working Notes 4 and 12 for workshop reports [12, 11]), we will shortly be updating this proposed standard. We plan to have a finalisation report with community support for the NLAFET Deliverable D2.4 (due in month 36).

Currently, we are planning to move to the group-based user interface which appears in current versions of `cblas_dgemm_batch` in Intel MKL. Ongoing discussions are debating the type of error handling that will be supported by the community standard. We are also looking to standardise other memory formats to store the batch of matrices, such as the strided and interleaved memory formats; details in NLAFET Working Note 5 *A Comparison of Potential Interfaces for Batched BLAS Computations* [16]. Our experiments have shown that these alternative memory formats can increase performance more than 10× for very small matrices. A repository exploring these alternative memory formats can be found on Github[1].

For dense factorizations and solvers, we have released an implementation of most of the existing BLAS and LAPACK routines using the OpenMP task-based programming paradigm. This appears as a new version of the PLASMA project for dense linear algebra, which is a collaboration between The University of Tennessee and The University of Manchester. A current snapshot of this project can be found in the NLAFET github directory NLAFET/plasma whilst the main repository is at `https://bitbucket.org/icl/plasma`.

[1] `https://github.com/sdrelton/bblas_interleaved`

Though not all of these functions are entirely relevant for the NLAFET project, the team at Manchester was responsible for (amongst other parts) the QR, LU, LL^T and LDL^T factorizations and the BLAS operations within. These routines are competitive with and often beat, implementations from the existing LAPACK collection and Intel's MKL library. In particular, these four factorizations form the backbone of Deliverable D2.1.

The excellent performance we see is due to the high levels of parallelism expressed in the task-based framework along with an autotuning framework based on Lua. More details on the runtime systems and task-based programming can be found in Deliverable D2.1. Further detail about the autotuning involved can be found in section 6 of this document and in Deliverable D6.4.

From this OpenMP version we were able to generate StarPU versions of the routines by utlising the KStar source-to-source compiler[2]. By comparing these against an older implementation in a runtime called Quark (developed at the University of Tennessee), plus current versions of MKL and LAPACK, we give a comprehensive summary of differences between the various runtimes that are suitable for the NLAFET project in Deliverable D2.1.

The software has been fully tested on Haswell and Broadwell NUMA nodes, along with the Intel Xeon Phi (codenamed Knights Landing).

3 Dense eigenvalue problems—tools and solvers

As of this date, the repositories for the tools and solvers of dense standard and generalized eigenvalue problems are:

- `SEVP-PDHSEQR-Alg953`: Nonsymmetric standard eigenvalue problems—package of routines for computing a real standard Schur form $S = Q^T AQ$.
- `GEVP-PDHGEQZ`: Nonsymmetric generalized eigenvalue problems—package of routines for computing a real generalised Schur form $(S, T) = Q^T(A, B)Z$.
- `Eigenvectors`: Eigenvector computations.
- `Eigen-reorder`: Eigenvalue reordering in Schur forms.

The parallel distributed algorithms and software contributions for computing Schur forms of standard and generalized eigenvalue problems, and available in the first two listed repositories above, represent the front line of research today. Within the NLAFET project, we make new contributions that both improve on and extend the functionalities to be able to efficiently compute eigenvectors corresponding to an arbitrary user selection of eigenvalues. Moreover, in our effort of addressing extreme scale systems we are developing task-based counter parts of the functionalities already available in the `SEVP-PDHSEQR-Alg953` and `GEVP-PDHGEQZ` repositories. Some of these results are available in the respositories `Eigenvectors` and `Eigen-reorder`. More is to come during the project.

`SEVP-PDHSEQR-Alg953`: The repository contains a clone of ALGORITHM 953 of the *Collected Agorithms of ACM* authored by Robert Granat, Bo Kågström, Daniel Kressner and Meiyue Shao, which is a *state-of-the-art package of library routines* for computing a

[2] `http://kstar.gforge.inria.fr`

standard Schur decomposition $S = Q^T AQ$, where A is a general square matrix with real entries and Q is an orthogonal transformation matrix. The associated scientific paper is published by ACM Transactions of Mathematical Software [9].

The main routine PDHGEQR is based on our parallel multishift QR algorithm with aggressive early deflation for computing the standard real Schur decomposition S. The real and complex conjugate pairs of eigenvalues appear as 1×1 and 2×2 blocks, respectively, along the diagonal of S and can be reordered in any order. Typically, this functionality is used to compute an orthogonal basis for an invariant subspace corresponding to a selected set of eigenvalues. The parallel algorithms and software are developed by the Umeå team in collaboration with EPFL. The software is MPI based, written in Fortran 90 for double precision real arithmetic, and targets distributed memory HPC systems.

The software is complemented by a PDHGEQR User's Guide, available in the repository, which includes sections on installation, compilation and usage of the parallel library routines. The Guide also contains instructions and scripts for tuning of parameters.

`GEVP-PDHGEQZ`: The repository contains a *state-of-the-art package of library routines* for computing a generalized Schur decomposition $(S, T) = Q^T(A, B)Z$, where A and B are general square matrices with real entries and Q and Z are orthogonal transformation matrices. The real and complex conjugate pairs of eigenvalues appear as 1×1 and 2×2 blocks, respectively, along the diagonals of (S, T) and can be reordered in any order. Typically, this functionality is used to compute orthogonal bases for a pair of deflating subspaces corresponding to a selected set of eigenvalues. The parallel algorithms and software are developed by the Umeå team in collaboration with EPFL (Daniel Kressner) and based on our scientific work published by SIAM Journal on Scientific Computing [1]. The software is MPI based, written in Fortran 90 for double precision real arithmetic, and targets distributed memory HPC systems.

The main routine is PDHGEQZ, which is based on our parallel multishift QZ algorithm with aggressive early deflation for computing the generalized real Schur form of a matrix pair (A, B) in Hessenberg-triangular (HT) form [1]. A novel parallel algorithm for identifying and deflating infinite eigenvalues such that they do not inflict damage to other eigenvalues, due to round-off, is part of the software.

The software is complemented by a User's Guide, namely NLAFET Working Note 2 *PDGEQZ User Guide* [2]. The calling sequences for the main driver routines with input and output parameters are described in detail. In addition, a set of tunable parameters and their usage with default values are discussed. During the build process, internal tests are performed to ensure the software works as intended. Both sequential and parallel tests are performed, with validation of the computed results.

The package also includes our MPI-based implementation, PDGGHRD, for the reduction of a real general matrix pair (A, B) to Hessenberg-triangular form (H, T), which is the first main step in the computation of a generalized Schur form (S, T) of (A, B). PDGGHRD is a one-stage distributed algorithm with wave-front scheduling. The static scheduler addresses the problem of underutilized processes caused by two-sided updates of matrix pairs based on sequences of rotations. For more details see the NLAFET Working Note 1 *Distributed One-Stage Hessenberg-Triangular Reduction with Wavefront Scheduling* [3].

`Eigenvectors`: We have developed a new parallel distributed memory algorithm for simultaneous computation of a set of eigenvectors X (columns of $X($ associated with a user selection of eigenvalues for matrices in Schur form $S = U^H AU$. Optionally, the algorithm

computes the eigenvectors of A by back-transformation, i.e. $X \leftarrow UX$. The algorithm has been implemented for the standard eigenvalue problem $Ax = \lambda x$ for double precision complex floating point arithmetic. The new parallel implementation is MPI based. A summary of the new algorithm, its implementation and performance can be found in the deliverable report *D2.5: Eigenvalue problem solvers*.

The goal remains to develop a task based algorithm for the problem of computing eigenvectors and the work is in progress. As there was no existing software for the parallel computation of eigenvectors, we first addressed the problem within the familiar setting of MPI programming. In its own right our MPI software is an important contribution that extends the functionality of the current state of the art.
The fundamental problems associated with the efficient computation of eigenvectors are:

- The computation of any eigenvector can overflow.
- The computation of a single eigenvector is memory bound.

The potential overflow problem is likely to manifest itself when the corresponding eigenvalue is part of a cluster of nearby eigenvalues. In LAPACK, there exits a sequential algorithm for addressing overflow in the computation of a single eigenvector. Unfortunately, this algorithm can not be parallelized efficiently as global synchronization is required at the end of every single iteration. This problem has now been solved! We now understand how scalings can be applied in a parallel context to fight overflow. These technical details are discussed in the NLAFET Working Note 9 *Robust solution of triangular systems* [14].

The computation of a single eigenvector is memory bound. This is an inescapable consequence of modern computer architectures with deep memory hierarchies. However, it is possible to interleave the computation of several eigenvectors and increase the arithmetic intensity. We now understand how this can be done! The arithmetic intensity can be increased even more by merging the computation of eigenvectors of the Schur matrix S with the back transformation to eigenvectors of the original matrix A. These technical details are discussed in the NLAFET Working Note 10 *Towards Highly Parallel and Compute-Bound Computation of Eigenvectors of Matrices in Schur Form* [4].

`Eigen-reorder`: We have developed a new parallel algorithm for eigenvalue reordering in Schur forms. The algorithm has been implemented for the standard eigenvalue problem for matrices in real Schur form, double precision arithmetic. The new implementation is task based and can be executed using StarPU. A summary of the new algorithm, its implementation and performance can be found in the deliverable report *D2.5: Eigenvalue problem solvers*.

The previous state of the art was established by an MPI based algorithm, implemented in ScaLAPACK as PBDTRSEN, a parallel blocked code for reordering eigenvalues in real Schur forms (an earlier contribution by the Umeå group and is also a part of the library within `SEVP-PDHSEQR-Alg953`). Many of its key features, such as multiple moving windows for increased concurrency and accumulation of local transformations for increased arithmetic intensity have been retained in the new algorithm. The main obstacle has been to redesign and develop the algorithm into a task based algorithm. For this problem we have been successful, as demonstrated by performance results in the deliverable report D2.5. The measured results are from shared memory NUMA nodes on the Abisko and Kebnekaise systems at HPC2N, Umeå University.

The lessons learned are directly applicable for the standard eigenvalue problem for complex matrices A, as well as for both real and complex flavours of the generalized

eigenvalue problem $Ax = \lambda Bx$. In addition, our understanding on how to cope with the non-trivial data dependencies caused by the two-sided transformations in eigenvalue reordering will facilitate the development of algorithms and software for the eigenvalue problems to be addressed.

Much more technical information can be found in the NLAFET Working Note 11 *Task-Based Parallel Algorithms for Eigenvalue Reordering in Real Schur Forms* [15]. We are currently in the process of reconfiguring the task based code to run on a distributed memory machine using the runtime system StarPU[3], a research project with its own development team. We have been very pleased with their responses to our comments, proposals and questions.

4 Sparse direct factorizations and solvers

As of this date, the repositories for the sparse direct solvers are:

- `Highly-Unsymmetric`: Sparse solver for highly unsymmetric systems.
- `SpLLT`: Sparse Cholesky solver for symmetric, positive-definite systems.
- `SpLDLT`: Sparse $LDLT$ solver for symmetric, indefinite systems.

We are developing two different solvers for sparse symmetric systems that are kept in two separate repositories. First we have `SpLLT`, a Cholesky solver for positive definite systems and secondly `SpLDLT` for solving symmetric indefinite problems. Table 1 gives a summary of the parallel codes for symmetric systems involved in Task–3.2 of our project.

In the context of *positive definite systems*, we have developed `SpLLT` using a runtime system approach for enabling portability and maintainability of the code. In addition, the runtime system provides a high-level API allowing us to express the parallel code in a simple manner. The runtime system is then responsible for handling the dependency management and data coherency across the architecture. In this context, `HSL_MA87` corresponds to our reference solver and relies on a classical design where the solver itself handles the task dependencies and scheduling. As a result of this approach, the cost of handling the task-graph is kept extremely low compared to a generic runtime system such as StarPU but `HSL_MA87` suffers from a lack of portability and maintainability. We implemented the algorithms in `SpLLT` using several runtime systems and programming models and we reported our results in the NLAFET Working Note 7 *Experiments with sparse Cholesky using a sequential task-flow implementation* [7]. The goal of this study was to ensure that our approach would lead to an implementation that is competitive in terms of performance compared to a state-of-the-art solver. The compilation process for `SpLLT` is handled by the CMake tools (`https://cmake.org/`) and the different variants of our solver that are OpenMP, StarPU or Parsec can be obtain by setting the variable `RUNTIME` when configuring the compilation Makefile. The software package includes a test driver called `spllt_test` located in the `test` directory for testing the performance and correctness of the solver and uses the runtime system that has been chosen during the configuration of the compilation. Note that `SpLLT` also contain the test driver `run_ma87.f90` for testing `HSL_MA87` which is useful when conducting performance comparison against our solver.

[3] `http://starpu.gforge.inria.fr/`

For now, our solver has been proven to be efficient on multicore architectures and although it is capable of running on a GPU system thanks to the portability provided by the StarPU runtime system, we still need to improve the GPU-version to make it efficient on heterogeneous CPU-GPU platforms.

In the context of *indefinite system*s, we are developing the SpLDLT solver which uses the a posteriori threshold pivoting (APTP) algorithm designed by Jonathan Hogg and implemented in `SSIDS V2` mainly in the context of the NLAFET project (see NLAFET Working Note 6 *A new sparse LDL^T solver using a posteriori threshold pivoting*[13]). The idea is to extend the SSIDS solver and move to a fully task parallel approach as in `SpLLT` using a runtime system. We plan to compare our solver against the two state-of-the-art solvers `PARDISO` and `HSL_MA97` which implement two different strategies for solving indefinite systems. `HSL_MA97` implements a numerically stable algorithm which limits the parallelism whereas `PARDISO` offers as much parallelism as possible but may lead to inaccurate results on numerically challenging problems.

In the context of Task–3.3, we are developing a solver for *unsymmetric systems* that are both structurally and numerically unsymmetric. This code implements a Markowitz/threshold scheme for the LU factorization and is specifically designed to exploit parallelism. The first part of our study in this work consists in comparing the serial version of our code to the state-of-the-art solver `HSL_MA48` which is a pure serial code and once we match the performance given by our reference solver we will then extend our approach by implementing the parallel version of our code.

5 Communication optimal algorithms for iterative methods

As of this date, the repositories for communication avoiding (CA) algorithms and tools for iterative methods are:

- `preAlps`: Kernels for preconditioned iterative methods.
- `enlargedKrylov`: Enlarged Krylov methods.

The INRIA team has been involved in the development of the packages `PreAlps` and `enlargedKrylov` available through the NLAFET Github repository. The main purpose of the `PreAlps` library is to provide robust algebraic preconditioners that can be used to solve linear systems arising from challenging applications. The first release of `preAlps` contains highly optimized kernels that are the building blocks of the iterative methods and preconditioners that we develop in this project, and that can also be used on their own in other applications. It is described in *Deliverable D4.1: Computational Kernels for Preconditioned Iterative Methods, prototype software, phase 1.* The package `enlargedKrylov` corresponds to the software in preparation about enlarged Krylov methods that are presented in *Deliverable D4.2: Analysis and algorithm design*, and will be released as *Deliverable D4.3: Prototype software, phase 2* in M24.
In summary, the current version of the library `PreAlps` released as *Deliverable D4.1* provides a routine for computing the sparse matrix–matrix product and routines for computing a low rank approximation of a sparse matrix based on communication avoiding sparse QR and CUR factorization kernels. The library also provides several sequential and parallel routines that can be used to perform basic linear algebra operations on sparse matrices. Our experiments show that our kernels lead to significant speedups on large

Table 1: Parallel codes for symmetric systems.

Symmetric positive definite		
MA87	HSL	Task-based Supernodal multicore OpenMP (2.0)
SpLLT	NLAFET	Task-based Supernodal. Multicore OpenMP (4.0, Task-based). Very efficient. Multicore Parsec (no data distribution). Performance issues. Multicore StarPU. Runs on heterogeneous CPU-GPU machines using MAGMA and CUBLAS kernels (potrf, trsm, syrk, gemm) plus a scatter/assemble kernel. Still not efficient. No distributed memory. Plan to port the StarPU code on distributed memory. We need to work on the data distribution.
Symmetric indefinite		
MA97	HSL	Task-based Multifrontal. OpenMP (2.0) Bit-compatible
MA86	HSL	Task-based Supernodal. OpenMP (2.0)
SSIDS V1	SPRAL	Batched Multifrontal. Runs entirely on GPU using CUDA with a CPU driver.
SSIDS V2	SPRAL	Task-based Multifrontal on multicore and batched Multifrontal factorization of subtrees on GPU (SSIDS V1). Work on CPU and GPU quite separated.
SpLDLT	NLAFET	Task-based Multifrontal. StarPU multicore still under development. Plan to implement a CPU-GPU version with StarPU code. Current routine has pdef option but quite primitive in both supernodal and multifrontal mode.

problems.

One of the main kernels developed in `PreAlps` is the `spMSV` (sparse matrix-set of vectors product) routine which is used to compute the product $C = A \cdot B$ of two sparse matrices A and B, where A is sparse and B is formed by a set of vectors. `spMSV` will be used by the enlarged Krylov subspace solvers that we develop in NLAFET, reported in M18 and released in M24. It is important to note that `spMSV` can also be used to compute efficiently a parallel sparse matrix-vector multiplication or a parallel sparse matrix-dense matrix multiplication, as those two operations are instances of our algorithm when the matrix B is formed by only one vector or a set of dense vectors, respectively. We evaluate the performance of `spMSV` on a set of matrices from the University of Florida sparse matrix collection [5] (those matrices arise from various real applications), and we compare the performance with respect to MKL. As MKL supports only shared memory, we tested two routines of MKL on a single node in order to determine how they linearly scale compared to `spMSV`. The first routine, `MKL(sparse x sparse)`, performs the product of two sparse matrices stored using CSR format. The second routine, `MKL(sparse x dense)`, performs the product of two matrices where the first is sparse and stored using CSR format, and the second is dense. Our experiments in *Deliverable D4.1* show that `spMSV` is on average 28 times faster than `MKL(sparse x dense)` with a maximum speedup of 90, and on average 12 times faster than `MKL(sparse x sparse)` with a maximum speedup of 25.

The library `preAlps` also includes two routines to perform a low-rank approximation of a sparse matrix based on QR and CUR factorizations. They both call the routine `preAlps_tournamentPivoting` which manages the parallelization using a reduction operation with a local (intra-processors) flat tree and a global (inter-processors) binary tree. This reduction technique is called tournament pivoting and has been used to construct communication avoiding algorithms, e.g., see [6] and [10]. Given a sparse matrix A of size $m \times n$, the routine `preAlps_tournamentPivotingQR` finds a rank-k QR approximation, i.e. $A\Pi \approx QR$ where Π is a permutation matrix obtained by tournament pivoting, Q is an orthogonal matrix implicitly defined using Householder reflections and R is a upper triangular matrix. For some cases, the diagonal elements of R can be taken as an approximation of the singular values of the matrix [6].
Interpretability of a low-rank approximation is very important in diverse applications, for that we propose the routine `preAlps_tournamentPivotingCUR` which allows to find a rank-k approximation of a sparse matrix using its rows and columns, i.e. $\Pi_r A \Pi_c \approx CUR$ where Π_r and Π_c are permutation matrices obtained using tournament pivoting, C is a $m \times k$ matrix formed by k selected columns of A, U is a $k \times k$ dense matrix and R is a $k \times n$ matrix formed by k selected rows of A.
The experiments presented in *Deliverable D4.1* show that the low-rank approximation routines have a good scalability and can be used for treating problems from diverse applications.

The subdirectories in `preAlps` repository follow the structure suggested in the introduction of the document, and include instructions to install and run the program. The `src` directory contains the source code of our main kernels `spMSV`, `TournamentPivotingQR`, `TournamentPivotingCUR`, and sequential and parallel routines developed in the library such as loading, distributing, and partitioning a matrix in parallel. The `test` directory contains code for testing. By using the routine `test_spMSV`, `test_prototypeQR` and `test_prototypeCUR`, the end-user can test these routines to perform a matrix-matrix

product, and compute low-rank communication avoiding approximations based on QR and CUR factorizations of sparse matrices.

We have started the development of two new approaches for solving sparse linear systems of equations $Ax = b$ involving symmetric and positive definite matrices arising from the discretization of partial differential equations on unstructured two-dimensional (2D) and three-dimensional (3D) grids. The first approach based on enlarged Krylov methods, and the second is a robust algebraic Schur complement preconditioner based on low rank corrections. Both approaches are described in details in *Deliverable D4.2: Analysis and algorithm design*, and reported in M18, and the software will be available for use as *Deliverable D4.3: Prototype software, phase 2* in M24. As of this date, a prototype of the enlarged Krylov methods is available as `enlargedKrylov` through the NLAFET Github repository. We have also started to test our methods on large matrices provided by our EDF partner (see *Deliverable D5.1: Requirements analysis, M18*).

6 Cross-cutting tools

As of this date, the repositories for cross cutting tools and software are:

- `pcp-runtime`: Parallelizing the critical path.

The UNIMAN team has been involved in cross-cutting issues from two different angles. First, we have explored many of the available runtime systems used for task-based programming and determined their suitability for the NLAFET project. Since there are a large number of different runtime systems we first performed a subjective analysis of their various features, focusing on heterogeneity, support for distributed memory, ease of use, and the possibility to use different task schedulers. From this analysis we narrowed the search down to just a few runtime systems whose performance is explored in more detail within Deliverable D2.1. To summarise, we believe that StarPU is the most suited runtime system for our project.

Second, we have worked on producing a framework for off-line autotuning within the NLAFET project. Specifically we have focused on tuning PLASMA at this stage, though our approach is applicable to the other pieces of software from the NLAFET consortium. The PLASMA software offers rich autotuning opportunities. To facilitate autotuning and ease of switching between different CPU architectures (Intel, ARM, POWER), PLASMA incorporates the Lua scripting language (see http://www.lua.org). Tuning mechanics can be defines in a Lua file, allowing for maximum flexibility and power of expression. Settings can be changed without recompiling the PLASMA library and files can be easily swapped for different target hardware/software environments. The full power of the Lua language can be use to define the tuning mechanics, switch statements, lookup tables, etc. In more detail: we have defined a Lua interface to our code for selecting the optimal tuning parameters at runtime.

Lua is a lightweight, portable, embeddable, and open-source scripting language that is ideal for making small extensions to larger software projects. Its only dependency is the availability of a C compiler so it does not hamper the portability of the overall software. In this scenario, we use Lua to return an "optimal" tile size for matrices with sizes that we haven't tested by interpolating the values seen during our autotuning runs. Also, if we can fit a smooth curve (or surface) to our optimal parameters (which will be explored

later) then this curve can be coded into Lua to provide parameters for matrices with various sizes.

This is more flexible than a standard configuration file since arbitrary functions can be implemented within Lua to, for example, interpolate optimal parameters for known matrix sizes to get good parameters for previously unseen matrix sizes. In Deliverable D6.4 we show that, for some routines, the optimal parameters in PLASMA can be approximated by fitting a logarithmic curve to our tuning data; this curve can then be coded up directly in Lua.

Once this Lua interface is defined, we need to perform some optimization to find the optimal parameters for a variety of matrix sizes. We need this data to perform the curve fitting explained above. The standard method of performing such an optimization is to use a grid-swep: iterating over all possible combinations of parameters and selecting the best one. However, some kernels coming from the NLAFET consortium will require optimizing over many parameters with various types (integers, floating point values for tolerances etc.) which are not well-suited to grid-sweeps due to the high cost and, for floating point parameters, the impossibility of testing all possible values.

We suggest that, instead of using a grid-sweep, optimization software such as OpenTuner[4] can be used to speed-up the optimization step. Since OpenTuner primarily uses various genetic algorithms, it is well-suited to solving problems with different input types and can even optimize compiler flags. A full description of our proposed autotuning framework can be found in Deliverable D6.4, where we apply our methodology to routines from the dense linear algebra project PLASMA.

The ongoing work by the UMU team on the evaluation of auto-tuning techniques will be reported in future deliverables. Partial results on a tunability case study are presented in NLAFET Working Note 8 *Evaluation of the Tunability of a New NUMA-Aware Hessenberg Reduction Algorithm* [8]. Software tools that emerge from the auto-tuning studies will also appear on GitHub/NLAFET.

As a part of Deliverable D6.1, the UMU team has developed a prototype runtime system (called `pcp`) to evaluate a specific idea for potentially enhancing the strong scalability of task-based codes. The purpose of the runtime system is to evaluate the idea—not replace or compete with existing runtime systems. If the idea turns out to work sufficiently well, then we will consider implementing optimized support for it in mature runtime systems such as StarPU.

The idea is to *parallelize the critical path* (hence the name) across a few cores *in addition to* optimizing the tile size in tiled task-based codes globally. The reasoning is that doing so would allow for a coarser optimal task granularity which would lead to more efficient task executions and overall less scheduling overhead. The end result is a faster parallel execution time using optimal parameter settings.

At this stage, we provide two examples to validate the implementation and tentatively evaluate the idea. The `dtrsm` (triangular solver) example solves a lower triangular linear system with multiple right-hand sides using a tiled task-based algorithm. The `dpotrf` (Cholesky factorization) example computes a Cholesky factorization of a dense symmetric positive definite matrix using a tiled task-based algorithm.

The runtime system supports homogeneous shared-memory machines and currently expects a Linux-based operating system with an x86-64 processor architecture. Porting to other operating systems and processor architectures is expected to be straightforward.

[4] `http://opentuner.org/`

The source code for the runtime system, two toy examples used to validate the implementation, test drivers, and test results can be found in the `pcp-runtime` repository under the NLAFET organization on GitHub. More specifically, the repository contains the directories:

- `src/` Source code for the runtime system and the two examples.
- `src/runtime/` Source code for the runtime system library.
- `src/examples/dtrsm/` Source code for the `dtrsm` example.
- `src/examples/dpotrf/` Source code for the `dpotrf` example.
- `test/` Drivers for testing the examples and analyzing the results.
- `data/` Input for the test drivers.
- `results/` Expected results (on Kebnekaise) for the examples.

7 Challenging applications

The outcome of the requirements analysis for the selected applications is described in the report for Deliverable 5.1. The integration of NLAFET software in the respective application software libraries will be described in forthcoming deliverables D5.2 and D5.3.

8 Summary and future contributions

In Table 2, we list the current repositories on GitHub NLAFET with their current level of access. As mentioned in the introduction, we will make repositories public (instead of private) whenever the codes are robust enough to be released. In addition, new repositories will be added and occasionally some existing repositories will be merged. Public releases occur at several times during the project and will be announced on the NLAFET website.

Table 2: GitHub NLAFET repositories.

Repository name	Access	WP	Brief description
`BBLAS-ref`	Public	WP2	Batched BLAS reference implementation
`plasma`	Public	WP2	Snapshot of PLASMA library
`SEVP-PDHSEQR-Alg953`	Public	WP2	Standard eigenvalue problem solvers
`GEVP-PDHGEQZ`	Public	WP2	Generalized eigenvalue problem solvers
`Eigen-reorder`	Private	WP2	Eigenvalue reordering in Schur forms
`Eigenvectors`	Private	WP2	Eigenvector computation
`Highly-Unsymmetric`	Private	WP3	Sparse solver for highly unsymmetric
`SpLLT`	Private	WP3	Sparse LL^T solver for $A = A^T$, pos. definite
`SpLDLT`	Private	WP3	Sparse LDL^T solver for $A = A^T$, indefinite
`preAlps`	Private	WP4	Kernels for preconditioned iterative methods
`enlargedKrylov`	Private	WP4	Enlarged Krylov methods
`PCP-runtime`	Private	WP6	Parallelizing the critical path

References

[1] B. Adlerborn, B. Kågström, and D. Kressner. A Parallel QZ Algorithm for Distributed Memory HPC Systems. *SIAM J. Sci. Comput.*, 36(5):C480–C503, 2014.

[2] Björn Adlerborn, Bo Kågström, and Daniel Kressner. PDHGEQZ User Guide. *NLAFET Working Note* WN-2, May, 2016. Also as Report UMINF 15.12, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[3] Björn Adlerborn, Lars Karlsson, and Bo Kågström. Distributed One-Stage Hessenberg-Triangular Reduction with Wavefront Scheduling. *NLAFET Working Note* WN-1, May, 2016. Also as Report UMINF 16.10, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[4] Björn Adlerborn, Carl Christian Kjelgaard Mikkelsen, Lars Karlsson, and Bo Kågström. Towards Highly Parallel and Compute-Bound Computation of Eigenvectors fo Matrices in Schur Form. *NLAFET Working Note* WN-10, April, 2017. Also as Report UMINF 17.10, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[5] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[6] James W. Demmel, Laura Grigori, Ming Gu, and Hua Xiang. Communication avoiding rank revealing QR factorization with column pivoting. *SIAM Journal on Matrix Analysis and Applications*, 36(1):55–89, 2015.

[7] Iain Duff, Jonathan Hogg, and Florent Lopez. Experiments with sparse Cholesky using a sequential task-flow implementation. *NLAFET Working Note* WN-7, November, 2016. Also as Technical Report RAL-TR-2016-016, Science & Technology Facilities Council, UK.

[8] Mahmoud Eljammaly, Lars Karlsson, and Bo Kågström. Evaluation of the Tunability of a New NUMA-Aware Hessenberg Reduction Algorithm. *NLAFET Working Note* WN-8, December, 2016. Also as Report UMINF 16.22, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[9] R. Granat, B. Kågström, D. Kressner, and M. Shao. ALGORITHM 953: Parallel Library Software for the Multishift QR Algorithm with Aggressive Early Deflation. *ACM Trans. Math. Software*, 41(4):Article 29:1–23, 2015.

[10] Laura Grigori, Sebastien Cayrols, and James W. Demmel. Low rank approximation of a sparse matrix based on LU factorization with column and row tournament pivoting. *NLAFET Working Note* WN-3, May, 2016. Also as INRIA Research Report 8910, Project Team Alpines, France.

[11] Sven Hammarling. Second Workshop on Batched, Reproducible, and Reduced Precision BLAS. *NLAFET Working Note* WN-12, February, 2017. Also as MIMS EPrint 2017.14, Manchester Institute for Mathematical Sciences School of Mathematics, The University of Manchester, UK.

[12] Sven Hammarling. Workshop on Batched, Reproducible, and Reduced Precision BLAS. *NLAFET Working Note* WN-4, July, 2016. Also as MIMS EPrint 2016.41, Manchester Institute for Mathematical Sciences School of Mathematics, The University of Manchester, UK.

[13] Jonathan Hogg. A new sparse LDL^T solver using a posteriori threshold pivoting. *NLAFET Working Note* WN-6, November, 2016. Also as Technical Report RAL-TR-2016-017, Science & Technology Facilities Council, UK.

[14] Carl Christian Kjelgaard Mikkelsen and Lars Karlsson. Robust solution of triangular linear systems. *NLAFET Working Note* WN-9, March, 2017. Also as Report UMINF 17.9, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[15] Mirko Myllykoski, Carl Christian Kjelgaard Mikkelsen, Lars Karlsson, and Bo Kågström. Task-Based Parallel Algorithms for Reordering of Matrices in Real Schur Form. *NLAFET Working Note* WN-11, April, 2017. Also as Report UMINF 17.11, Dept. of Computing Science, Umeå University, SE-901 87 Umeå, Sweden.

[16] Samuel D. Relton, Pedro Valero-Lara, and Mawussi Zounon. A Comparison of Potential Interfaces for Batched BLAS Computations. *NLAFET Working Note* WN-5, August, 2016. Also as MIMS EPrint 2016.xx, Manchester Institute for Mathematical Sciences School of Mathematics, The University of Manchester, UK.