NLAFET
Parallel Numerical Linear Algebra for Future Extreme Scale Systems

H2020–FETHPC–2014: GA 671633

# D5.2
# Software integration

April 2018

## Document information

| | |
|---|---|
| Scheduled delivery | 2018-04-30 |
| Actual delivery | 2018-04-27 |
| Version | 1.1 |
| Responsible partner | INRIA |

## Dissemination level

CO — Confidential (until June 15; after that Public)

## Revision history

| Date | Editor | Status | Ver. | Changes |
|---|---|---|---|---|
| 2018-04-15 | INRIA & UNIMAN members | Draft | 1.0 | Initial version of document produced |
| 2018-04-25 | INRIA, UNIMAN, RAL members | Final Version | 1.1 | Second version of document produced |

## Author(s)

Simplice Donfack, Laura Grigori, Jan Papez, Olivier Tissot, INRIA
Maksims Abalenkovs, Jack Dongarra, Mawussi Zounon, UNIMAN
Iain Duff, Stojce Nakov, RAL

## Internal reviewers

Carl Christian Kjelgaard Mikkelsen, UMU
Sébastien Cayrols, RAL
Florent Lopez, RAL

## Copyright

## 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 Figures

## List of Tables

# 1 Introduction

The *Description of Action* document states for Deliverable 4.2:

"*Integration of the NLAFET library into the respective application environments.*"

This deliverable is in the context of Workpackage 5, and describes our efforts on testing and integrating the NLAFET library into several challenging real-world applications.

1. Task-based Shared Memory Parallelism into 2DRMP. Collaboration with Professor Stan Scott. NLAFET contact Jack Dongarra, UNIMAN.
2. Load flow based calculations in large-scale power systems and PowerFactory code. Collaboration with DIgSILENT GmbH, Germany. NLAFET Contact, Iain Duff, RAL.
3. Communication avoiding iterative methods and their efficiency for solving linear elasticity problems and linear systems arising from Code Saturne (collaboration with EDF). NLAFET contact Laura Grigori, Inria.
4. Data analysis in astrophysics and Midapack. Collaboration with University Paris 7, France. NLAFET contact Laura Grigori, Inria.

We discuss them in detail in Sections 2 to 5 and we present our conclusions in Section 6.

The communication avoiding iterative methods tested in (3) and (4) above are currently available within preAlps library (Inria) via the github of NLAFET. PreAlps was developed by Inria in the context of Workpackage 4. The code has a reverse communication interface such that it can be easily integrated into other scientific application and is scalable up to 16,000 cores.

In the Description of Action document, we have planned to collaborate with Thomas Schulthess of ETH Zurich, Switzerland to test the dense solvers/eigensolvers developed in Workpackage 2 in materials science and chemistry. However this collaboration was not pursued due to a lack of time from ETH Zurich.

# 2 Introduction of Task-based Shared Memory Parallelism into 2DRMP

This report presents extension of the `2DRMP` package to task-based shared memory parallelism via `PLASMA` [2]. The most computationally intensive parts containing linear algebra operations can be performed with `PLASMA` routines. The new routines introduced include: LU-factorisation based linear systems solution `dgetrs`, eigenvalue decomposition `dsyevd` and matrix–matrix multiplication `dgemm`. Preliminary performance results of kernel functions are presented.

## 2.1 Description of `2DRMP`

Developed in the 1970s, the R-matrix propagation method became a popular choice for solving close-coupling equations describing the electron and photon collisions with atoms,

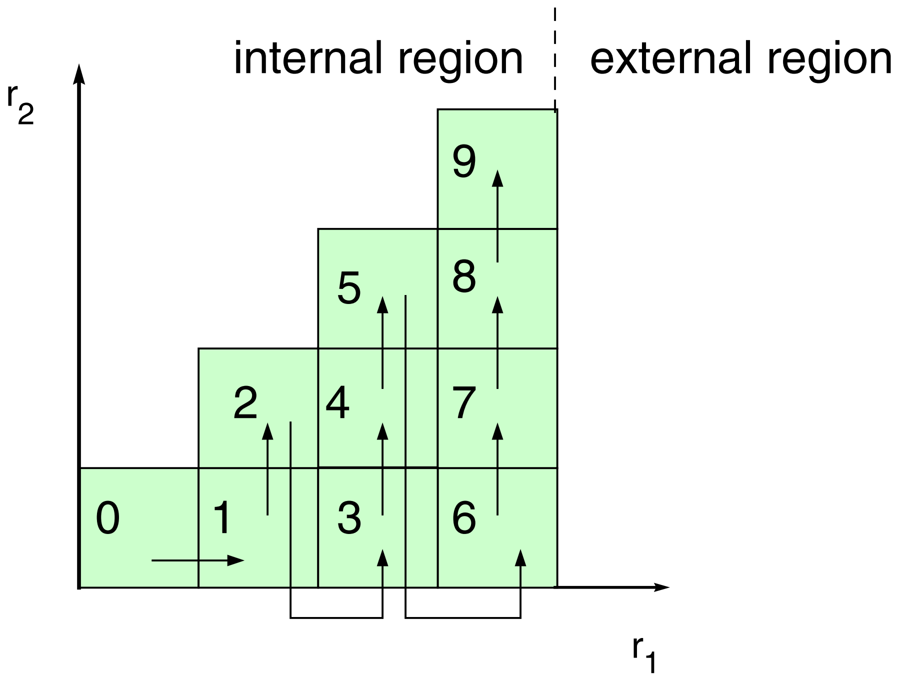

Figure 1: Propagation of global R-matrix through inner region

ions and molecules. The `2DRMP` presents a collection of R-matrix propagation programs aimed at simulation of electron scattering from Hydrogen-like atoms and ions at intermediate energies [17]. The main idea behind the efficiency of `2DRMP` is the subdivision of the internal propagation region into a number of subregions. Local R-matrices are propagated within each subregion contributing to the global R-matrix propagation. Figure 1 describes the order of the global R-matrix propagation via the subregions.

`2DRMP` code consists of seven subprograms:

**bp** constructs atomic basis functions and long-range potential coefficients

**rint2** computes radial integrals

**newrd** constructs Hamiltonian matrices in each subregion

**diag** diagonalises the subregion Hamiltonian matrices

**amps** uses matrix eigenvalues to calculate surface amplitudes on subregion edges

**prop** propagates global R-matrix across subregions of inner region (See Figure 1 for details)

**farm** solves the final set of equations

In the initial state the `2DRMP` package has been highly parallel. The majority of computationally intensive linear algebra operations have been performed in parallel with `LAPACK` or `ScaLAPACK` routines [3, 5]. Two levels of distributed memory parallelism have been present in the package: (i) `MPI` parallelisation according to subregions of the global R-matrix and (ii) `MPI` parallelisation according to the scattering energies [1].

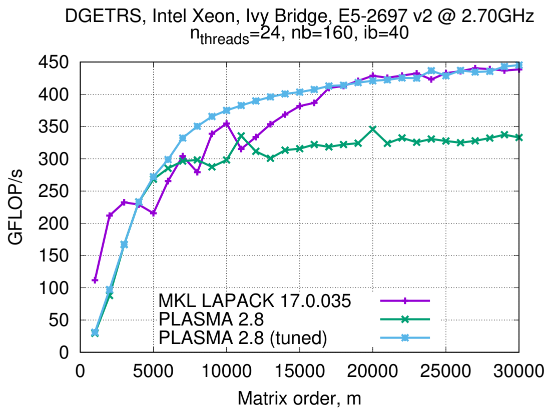

Figure 2: Performance of linear system solution based on LU-factorisation

## 2.2 Introduction of Tiled Asynchronous Routines

Alternative way to calculate computationally expensive parts of the code has been introduced with PLASMA. Tiled asynchronous PLASMA routines were incorporated into 2DRMP subprograms. The routine `diagonalise_matrix` in the `diag` part is now based on `dsyevd_tile_async`, performing eigenvalue decomposition on symmetric matrices. The propagation stage `prop` applies two PLASMA routines `dgetrs_tile_async` for linear system solutions based on LU-factorisation in the `system_solver` routine and `dgemm_tile_async` for matrix–matrix products.

## 2.3 Numerical Results

We compiled PLASMA using ICC 17.0.035 and linked to MKL 17.0.035 for its single-thread BLAS functions. For the sake of comparison, we also report the performance of MKL's corresponding routines. Figures 2 and 3 present performance results of `dgetrs` and `dsyevd` kernel routines run on the 24 core Ivy Bridge node (part of the Wonder, Phase 2 system at the STFC Daresbury Laboratory). PLASMA 2.8 is tested against the Intel MKL 17.0.035. Tuned PLASMA used variable tile sizes `nb`, that resulted in the highest performance for given matrix orders. The third curve, denoted "PLASMA", illustrates performance of untuned PLASMA with constant tile and inner block sizes `nb` and `ib`. Tuned PLASMA outperforms Intel MKL for matrix orders between 6000 and 17000.

Similar performance signature is obtained for the matrix diagonalisation kernel. In this case tuned PLASMA outperforms the MKL for a larger range of matrix orders: $m \in [6000, 20000]$. Even the untuned PLASMA persistently outperformed MKL for $m > 6000$. It was not particularly beneficial to tune PLASMA in this case, since the performance difference between the tuned and untuned versions was only minor.

## 2.4 Conclusion and Future Work

Preliminary experiments show that the use of tiled asynchronous routines for computationally expensive linear algebra operations in a suite of 2DRMP software is beneficial for certain matrix orders with $m > 6000$.

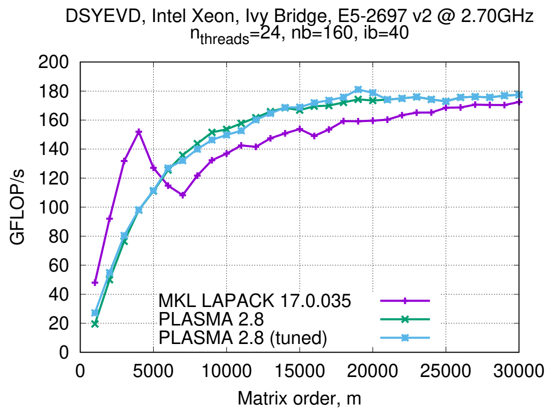

Figure 3: Performance of eigenvalue decomposition for symmetric matrices

Future work on this research front includes: (i) utilisation of the new `PLASMA` routines in `2DRMP`, (ii) potential application of matrix addition routines and (iii) thorough correctness testing and performance comparison against the original `LAPACK`-based version of `2DRMP`.

## 3 Load flow in large scale power systems

The main NLAFET contact is Iain Duff and the main applications contact is Bernd Klöss of DIgSILENT GmbH, Germany.

In this section we present performance results on test cases provided by the Power Systems application of DigSILENT GmbH (see *Deliverable 5.1*) using the ParSHUM library. The ParSHUM library is a parallel multithreaded library for factorization of highly unsymmetric matrices. For a given sparse matrix $A$, our solver decomposes the matrix into the form

$$PAQ = LU,$$

where $P$ and $Q$ are row and column permutation matrices, respectively and $L$ and $U$ are sparse lower and upper triangular matrices respectively. For further information about this library, please refer to *Deliverables 3.4* and *3.5*.

### 3.1 Performance results

The tests presented in this section were performed on a system called Kebnekaise, which is located in the High Performance Computing Center North (HPC2N) at Umeå University[1]. Each compute node contains 28 Intel Xeon E5-2690v4 cores organised into 2 NUMA islands with 14 cores in each. The nodes are connected with a FDR Infiniband Network. Each CPU core has 32 KB L1 data cache, 32 KB L1 instruction cache and 256 KB L2 cache. Moreover, for every NUMA island there is 35 MB of shared L3 cache. The total amount of RAM per compute node is 128 GB. In our experiment, we are using only just one NUMA node, so all the tests presented below are executed on fourteen cores.

[1] See https://www.hpc2n.umu.se/resources/hardware/kebnekaise.

We define the symmetry index $si$ by the expression

$$si(A) = \frac{number_{i \neq j}\{a_{ij} * a_{ji} \neq 0\}}{nnz\{A\}},$$

where $nnz\{A\}$ is the number of off-diagonal entries in the matrix $A$. A symmetric matrix will thus have a symmetry index of 1.0. We define $0/0$ to have the value 1.0 so that a diagonal matrix will be symmetric. A triangular matrix will have symmetry index zero. Our experiments suggest that matrices with symmetry indices of less than 0.9 can be considered highly unsymmetric. Additionally, we define the fill-in factor as the number of entries in the $L$ and $U$ factors divided by the number of entries in $A$ :

$$\frac{nnz\{L\} + nnz\{U\}}{nnz\{A\}}.$$

The main characteristics for the matrices used in this study are presented in Table 1.

| Matrix | | $n$ | $nnz$ | $si$ |
|---|---|---|---|---|
| InnerLoop1 | Balanced load flow | 197K | 745K | 0.44 |
| InnerLoop2 | Balanced load flow | 197K | 806K | 0.46 |
| InnerLoop3 | Balanced load flow | 197K | 806K | 0.46 |
| InnerLoop4 | Balanced load flow | 197K | 806K | 0.46 |
| Jacobian_unbalancedLdf | Unbalanced load flow | 203K | 2.41M | 0.80 |
| Newton_Iteration1 | Balanced optimal power flow | 427K | 2.38M | 0.14 |

Table 1: Statistics for the matrices that are used in this study. The size ($n$), number of nonzero entries ($nnz$) and the symmetry index ($si$) is given for each matrix.

We compared the performance of our library with two state-of-the-art solvers for unsymmetric matrices: MA48 [7] from the HSL library and UMFPACK [**?**] from the SuiteSparse library.

The execution time and the fill-in factor for the three solvers are presented in Figures 4 and 5 respectively. The ParSHUM solver outperforms the MA48 and UMFPACK solver for all the test problems except the Jacobian_unbalancedLdf matrix, for which the UMFPACK solver yields the lowest execution time. The main reason for this is that the ParSHUM solver has a much larger fill-in factor for this matrix, 3.9 for UMFPACK and 12.4 for the ParSHUM solver. Although the ParSHUM solver produces more fill-in for every matrix, it manages to compensate with its parallel execution. We will continue to improve our solver to cope better with these problems and further on this in *Deliverable 5.3.*

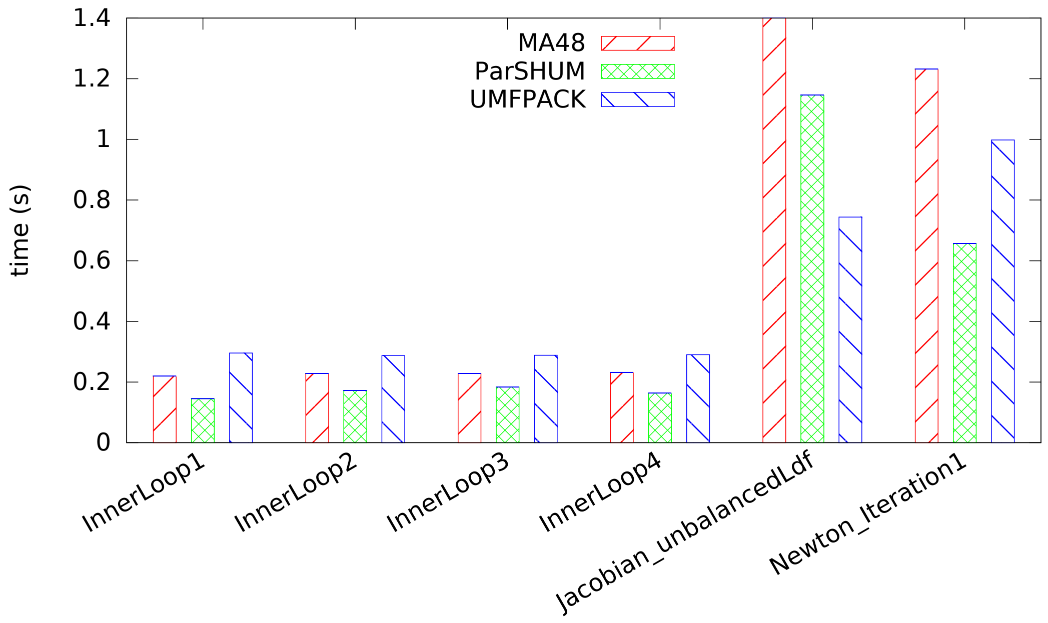

Figure 4: The execution time for the MA48, ParSHUM and UMFPACK solvers for the matrices from Table 1. The execution time for the Jacobian_unbalancedLdf matrix for MA48 is not presented since its value is too large (4.2 s).

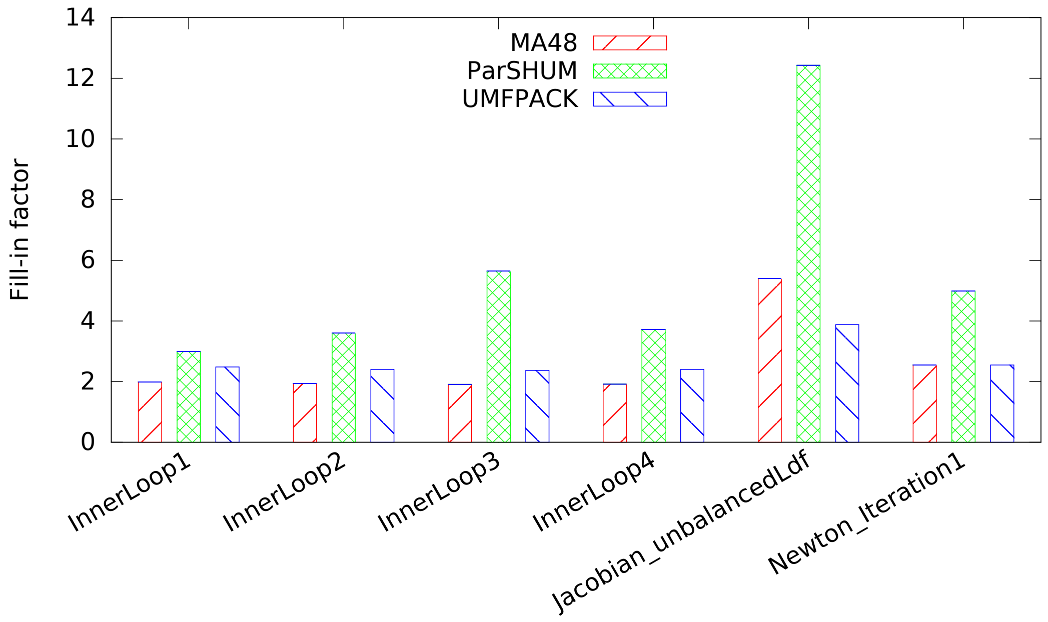

Figure 5: The fill-in factor for the MA48, ParSHUM and UMFPACK solvers for the matrices from Table 1.

# 4 Communication avoiding iterative methods and their efficiency for solving linear elasticity problems and linear systems arising from Code Saturne (collaboration with EDF)

In this section, we describe the numerical experiments we performed in order to assess the parallel efficiency of the communication avoiding iterative methods developed in Workpackage 4 for two different applications: linear systems arising from solving linear elasticity problems and linear systems arising from Code Saturne (collaboration with EDF). We focus on Enlarged Conjugate Gradient (ECG). We do not recall the definition of the method, neither the details of the implementation, and refer the reader to *Deliverable 4.2, 4.3 & 4.4* for more details on those aspects.

## 4.1 Description of the parallel environment

In the experiments we use a block Jacobi preconditioner, associating with each block a MPI process. Before calling ECG, each MPI process factorizes the diagonal block of $A$ corresponding to the local row panel that it owns. At each iteration of ECG, each MPI process performs a backward and forward solve locally in order to apply the preconditioner. Hence the application of the preconditioner does not need any communication. It is likely that there exist better preconditoners than block Jacobi for our test cases, however we are interested in the iterative method rather than in the preconditioner. In particular, we do not want to target specific applications and aim at being as generic as possible. Although in theory it is possible to apply any preconditioner within this implementation, in practice it is essential that applying this preconditioner to several vectors at the same time is not too costly, *e.g*, a sublinear complexity with respect to the number of vectors.

The following experiments are performed on a machine located at Umeå University as part of High Performance Computing Center North (HPC2N), called Kebnekaise. In our experiments, we use the so-called compute nodes, which are formed by Intel Xeon E5-2690v4 (Broadwell) with 2x14 cores. For a detailed description of the machine, we refer to the online documentation[2].

We compile the code (and its dependencies) using Intel toolchain installed on the machine: `mpiicc` (based on `icc` version 18.0.1 20171018) and MKL [20] version 2018.1.163. We use PETSc [4] in order to compare ECG implementation to PETSc PCG implementation. In particular, PETSc is configured to use MKL-PARDISO as exact solver for sparse matrices in the block Jacobi preconditioner. For partitioning the matrix we are using the METIS library downloaded and installed by PETSc.

## 4.2 Test cases

For our first test cases we also use the CFD solver *Code_Saturne* that solves the Navier-Stokes equations for incompressible flows. This code is developed in-house by EDF and is distributed under the GPL open source license (available at http://www.code-saturne.org). *Code_Saturne* has been included in the Unified European Applications Benchmark Suite of

[2]https://www.hpc2n.umu.se/resources/hardware/kebnekaise

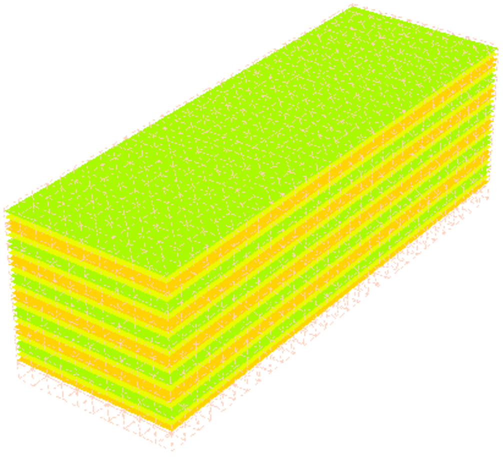

Figure 6: Heterogeneity pattern of the Young's modulus and Poisson's ratio for elasticity matrices.

the PRACE project (see http://www.praceri.eu/ueabs). It is used for a wide range of applications, many of which are related to nuclear engineering, but increasingly with applications related to renewables. We focus on simulations such as the computation of fluid flow in tube bundles, either cross-flow as in steam generators, or tangential as in Pressurized Water Reactor fuel assemblies, as these applications are numerically quite representative of a broader range of applications, and large-scale meshes for benchmarking are available. We have collaborated with Yvan Fournier from EDF to test ECG on a matrix corresponding to the BUNDLE test case.

The Ela matrices arise from the linear elasticity problem with Dirichlet and Neumann boundary conditions defined as follows

$$\operatorname{div}(\sigma(u)) + f = 0 \qquad \text{on } \Omega \tag{4.1}$$
$$u = 0 \qquad \text{on } \partial\Omega_D \tag{4.2}$$
$$\sigma(u)\cdot n = 0 \qquad \text{on } \partial\Omega_N \tag{4.3}$$

where $\Omega$ is a unit cube. The matrices Ela_$N$ correspond to this equation discretized with FreeFem++ [12] using a triangular mesh with $1600 \times N \times N$ points on the corresponding vertices and P1 finite elements scheme. The Dirichlet boundary is denoted $\partial\Omega_D$, $\partial\Omega_N$ is the Neumann boundary, $f$ is some body force, $u$ is the unknown displacement field. $\sigma(.)$ is the Cauchy stress tensor given by Hooke's law: it can be expressed in terms of Young's Modulus $E$ and Poisson's ratio $\nu$. For a more detailed description of the problem see [9, 14, 18]. We consider discontinuous $E$ and $\nu$, $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ and $(E_2, \nu_2) = (10^7, 0.45)$ (Figure 6). This test case is known to be difficult because the matrix is ill conditioned. In particular, the classical one-level preconditioners are not expected to be very effective [6]. Numerical properties of the test matrices are summarized in Table 2.

In all the experiments the tolerance is set as the default tolerance of PETSc, *i.e.*, $10^{-5}$ and the maximum number of iterations is set to 5000. The right-hand side is chosen uniformly random, and then normalized. The initial guess is set to 0. We do not use any kind of threading and use 28 MPI processes per node. Unless otherwise stated, we use one OpenMP thread per MPI process – we also perform numerical experiments to observe the effect of

| Name | Size | Nonzeros | Problem |
|---|---|---|---|
| BUNDLE | 13,044,996 | 347,890,620 | CFD |
| Ela_20 | 2,118,123 | 74,735,397 | Linear elasticity |
| Ela_30 | 4,615,683 | 165,388,197 | Linear elasticity |

Table 2: Test matrices.

threading in the last section.

## 4.3 Impact of the enlarging factor

First we study the impact of the enlarging factor $t$ on the methods. We fix the number of processors to 112 and we vary the value of $t$ for the 4 methods: Orthodir (Odir), Orthodir with dynamic reduction of the search directions (D-Odir), Orthomin (Omin) and Breakdown-Free Orthomin (BF-Omin). The results obtained are summarized in Table 3

| | t | Odir | D-Odir | Omin | BF-Omin |
|---|---|---|---|---|---|
| BUNDLE | 1 | 20.3 | 20.4 | 20.0 | 20.2 |
| | 2 | 23.1 | 23.0 | 23.9 | 23.1 |
| | 4 | 28.7 | 28.8 | 28.8 | 28.8 |
| | 8 | 36.8 | 36.4 | 36.7 | 36.4 |
| Ela_20 | 1 | ++ | ++ | ++ | ++ |
| | 4 | 97.6 | ++ | - | ++ |
| | 8 | 72.8 | 55.0 | - | ++ |
| | 12 | 56.8 | 51.5 | - | ++ |
| | 16 | 53.6 | 47.5 | - | ++ |
| | 20 | 56.3 | 47.2 | - | ++ |
| | 24 | 57.8 | 46.6 | - | ++ |
| | 28 | 59.9 | 47.5 | - | ++ |

Table 3: Runtime results for BUNDLE and Ela_20 ($P = 112$). The ++ means that the maximum number of iterations (5000) was reached and the - means that a breakdown occurred.

For BUNDLE matrix the reduction of the number of iteration is not balancing the increase in flops and the runtime is slightly increasing when $t$ increases. In fact, the number of iterations when $t = 1$ is already very low in this case (85) and increasing $t$ does not allow to reduce it significantly. For instance, when $t = 8$ the number of iterations is 69, a decrease of only 20%. Using the dynamic reduction of the search directions is not very effective for this matrix because it generally occurs when the method is about to converge.

For Ela_20 test case we remark that a breakdown occurs with Orthomin for all the values of $t$ that we tested. This behavior of elasticity matrices had also been reported in [11]. Using BF-Omin effectively cures the breakdowns but does not allow the method to converge

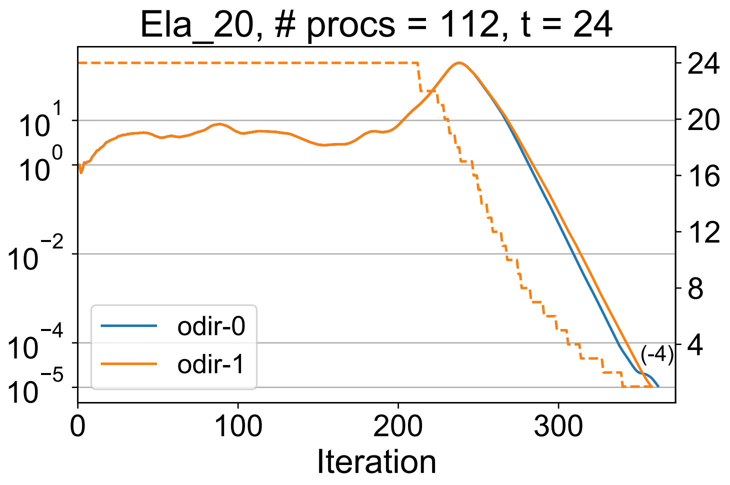

Figure 7: Convergence of D-Odir (odir-1) compared to Odir (odir-0). The dash line represents the number of search directions for D-Odir. In parenthesis the difference of iteration count to reach convergence between D-Odir and Odir (+ means that D-Odir took more iterations to converge).

within the prescribed maximum number of iterations. Similarly, D-Odir does not converge when $t = 4$, but performs very well when $t$ is larger. On the contrary, Odir is very stable and converges for all the values of $t$ tested. However, we also notice that using the dynamic Orthodir variant (D-Odir) allows to reduce significantly the runtime when $t$ is large: D-Odir is around 20% faster than Odir. Overall, for this matrix, the best method is D-Odir with $t = 24$.

In order to better understand how the dynamic reduction of the search directions affects the convergence we plot the normalized residual and the block size (dash line) as a function of the iteration count for Ela_20 (Figure 7). We note that the Ela_20 test case is very favorable: the block size is reduced even a bit before the convergence and the number of iterations is lower than when the search reductions are not reduced. On other test cases, we notice that the convergence is not really affected by the reduction of the search directions because the number of iterations remains almost the same. However the block size is effectively reduced as soon as the method starts to converge.

In conclusion, D-Odir is the best method over the different variants of ECG that we tested: it is a good compromise between the stability of Odir and the efficiency of the classical CG. Nevertheless, there exist matrices such as BUNDLE for which the reduction of the number of iterations does not compensate the extra cost of ECG compared to the classical CG, even when using the dynamic reduction of the search directions. These results support the theoretical convergence study that has been done but not presented in this document. Indeed ECG(t) is acting as if the $t$ smallest eigenvalues of the matrix were deflated. Finally, we notice that values of $t$ between 4 to 24 are good default parameters. Indeed, such values allow to effectively reduce the number of iterations while maintaining an affordable cost per iteration.

### 4.4 Strong scaling study

Following the parameter study, we perform a strong scaling study on Ela_30.

For BUNDLE test case, we compare PETSc PCG and Omin with $t = 4$. We do not use the dynamic reduction of the search directions as it is not very effective for this matrix, and we use Omin because it is a bit cheaper that Odir and does not breakdown for this matrix. The results are summarized in Table 4. We observe that both PETSc PCG and ECG are scaling very well. Nevertheless, in this case enlarging the Krylov subspaces does not allow to reduce significantly the number of iterations and ECG remains almost 2 times slower than PETSc PCG.

| | Omin(4) | | | CG | | |
|---|---|---|---|---|---|---|
| # MPI | # iter | res | time (s) | # iter | res | time (s) |
| 252 | 78 | 1.3E-4 | 12.1 | 89 | 1.3E-4 | 7.4 |
| 504 | 88 | 1.8E-4 | 5.9 | 98 | 1.9E-4 | 3.4 |
| 1,008 | 95 | 2.6E-4 | 2.9 | 105 | 2.7E-4 | 1.5 |

Table 4: Strong scaling results for BUNDLE.

For Ela_30 we compare PETSc PCG and D-Odir with $t = 24$, the best choice over the parameters we tested. The resulting runtimes are summarized in Table 5. Enlarging the Krylov subspaces allows us to reduce drastically the number of iterations: D-Odir(24) performs around 25 times less iterations than CG. As a consequence, D-Odir(24) is more than 5 times faster than PETSc PCG at small scale and around 2.5 times faster at large scale. We believe that this relatively poor scaling of D-Odir(24) compared to PETSc PCG at large scale is due to the implementation that is not as optimized as PETSc which has been developed over many years.

| | D-Odir(24) | | | CG | | |
|---|---|---|---|---|---|---|
| # MPI | # iter | res | time (s) | # iter | res | time (s) |
| 252 | 513 | 1.3E-4 | 77.9 | 13,626 | 1.3E-4 | 406.9 |
| 504 | 531 | 1.9E-4 | 45.5 | 15,819 | 1.9E-4 | 258.9 |
| 1,008 | 606 | 2.6E-4 | 23.7 | 17,023 | 2.7E-4 | 94.7 |
| 2,016 | 696 | 2.6E-4 | 14.5 | 19,047 | 2.7E-4 | 34.5 |

Table 5: Strong scaling results for Ela_30.

## 4.5 Impact of threads on performance

One motivation for enlarging the Krylov subspaces is to increase the arithmetic intensity of the resulting methods. The use of the so-called many-core architectures such as Nvidia GPUS, Intel Xeon Phi, or Sunway SW26010 used in the Sunway TaihuLight supercomputer, is particularly interesting in this context. As the implementation relies on the MKL library which is multi-threaded [20], it is straightforward to assess its efficiency on the Xeon Phi processors.

In order to do so, we perform the following experiments on NERSC's supercomputer Cori. It consists in two partitions, one with Intel Haswell processors and another one with the last generation of Intel Xeon Phi processors: Knights Landing (KNL). More precisely, the second partition consists in 9,688 single-socket Intel Xeon Phi 7250 (KNL) processors with 68 cores each. For a detailed description of the machine, we refer to the online documentation[3]. We compile the code (and its dependencies) using the default compilers and libraries installed on the machine: `icc` version 18.0.1, `cray-mpich` version 7.6.2, MKL version 2018.1.163 and METIS version 5.1.0.

We consider Ela_30 test case and we study the impact of threads on the strong scaling of Odir(24). We do not use the dynamic reduction of the search directions in order to keep the cost of one iteration constant during the solve to better understand the effect of threading. More precisely, we increase the number of threads assigned to each MPI process from 1 to 8. Then we vary the number of MPI processes from 64 to 2048. This means that we are using at most $2,048 \times 8 = 16,384$ cores. In practice, we use so-called core specialization in order to reserve 4 cores per node for handling system services that is why our total number of cores is not a multiple of 68.

The results obtained are summarized in Figure 8. First of all, we notice that there is a trade-off between using threads or MPI processes because the number of MPI processes dictates the preconditioner. Indeed, there is as many blocks in the block Jacobi preconditioner as the number of MPI processes, thus increasing the number of MPI processes deteriorates the quality of the preconditioner but also reduces its application cost. For instance, using 512 MPI processes takes 123s, and using 64 MPI processes with 8 threads each takes 179s: it is an increase of 50% compared to 512 MPI processes. Nevertheless, we observe that using more than 2 threads, and up to 8, has always a significant effect on the speed-up, even when the number of MPI processes is high. For instance, as shown in Table 6, increasing the number of threads from 1 to 8 with a fixed number of $2,048$ MPI processes leads to an decrease in runtime of nearly 3. Of course, we are not close to full efficiency when using multiple threads, but we are still taking advantage of the multithreaded BLAS 3 routines. This is illustrated by the Table 6 where we compare the speed-up obtained by using threads for Omin(1), which corresponds to the classical PCG, and Odir(24). We observe that using more than 2 threads is not effective at all with Omin(1) whereas it always significantly increases the speed-up with Odir(24).

| | Omin(1) | | Odir(24) | |
|---|---|---|---|---|
| # omp | time (s) | speed-up | time (s) | speed-up |
| 1 | 89 | - | 44 | - |
| 2 | 74 | 1.2 | 29 | 1.5 |
| 4 | 80 | 1.1 | 21 | 2.1 |
| 8 | 79 | 1.1 | 16 | 2.8 |

Table 6: Comparison between Omin(1) and Odir(24) with 2048 MPI processes. We indicate the speed-up when increasing the number of threads (# omp) for each method.

Finally, we are able to obtain an overall speed-up of 50 when using $16,384$ cores with respect to 64 cores. Compared to an ideal speed-up of 256, it may seem that this result is

[3]http://www.nersc.gov/users/computational-systems/cori/configuration/

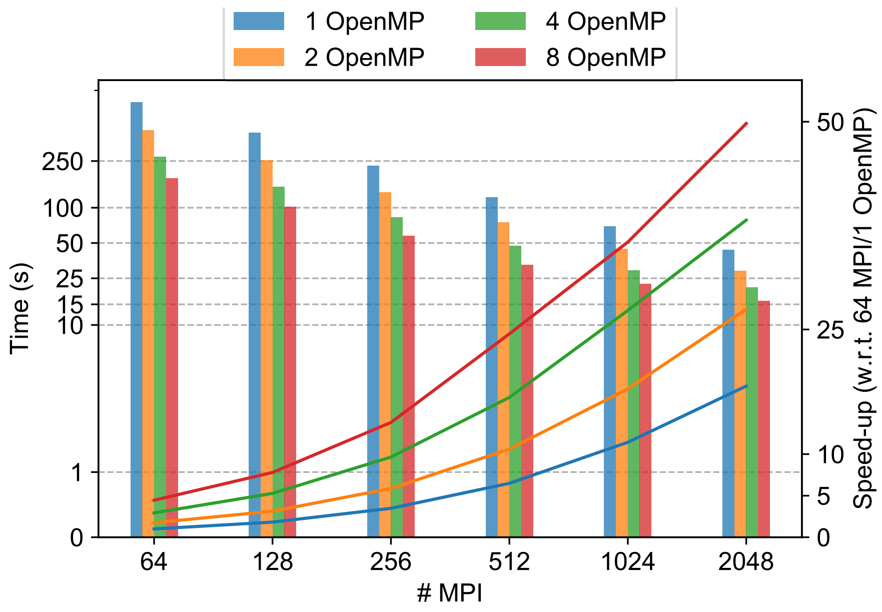

Figure 8: Strong scaling study on Ela_30 test case. The bars represent the runtime (left) and the lines represent the corresponding speed-up with respect to 64 MPI with 1 thread each (right).

not very good (around 20% of efficiency), however it is well-known that Krylov methods are not very efficient at very high scale[4]. Furthermore, it is important to notice that the matrices tested are relatively small, but they allow us to simulate extreme scale situation: with 16,384 cores the average number of unknowns per core is around 280. We have shown that in such cases, using enlarged Krylov subspaces allows to increase arithmetic intensity while decreasing the communication by drastically decreasing the overall iterations. Thus it takes advantage of the current trend in hardware architecture for reaching exascale.

## 5 Astrophysics : CMB data analysis

The main NLAFET contact is Laura Grigori and the principal application contacts are Radek Stompor from APC/CNRS, France, and Carlo Baccigalupi of SISSA Italy.

Studies of the Cosmic Microwave Background (CMB) anisotropies have been driving the progress in our understanding of the Universe for nearly quarter of the century. CMB data analysis commonly involves solutions of large, structured linear systems of equations. Two typical and important examples of such systems are map-making and Wiener Filter systems of equations. These systems are solved either as a stand-alone task or as part of more involved process, such as a power spectrum estimation, which commonly require multiple solutions of such systems.

In both the examples mentioned above, the system matrices are in practice not available and one can only perform their application to a vector. Because of this, the Krylov methods (in particular, as the matrices are symmetric positive definite, preconditioned con-

[4]This is well illustrated by HPCG benchmark: http://www.hpcg-benchmark.org/custom/index.html?lid=155&slid=293

jugate gradients) are the methods of choice. A standard preconditioner used in the field is a simple block-diagonal preconditioner, which is easy to construct and apply. Recently (see, e.g., [10,19]), this preconditioner was improved using the deflation of few vectors approximating the eigenvectors associated with the smallest eigenvalues of the system matrix, which often harm the convergence of the PCG solver. The resulting, two-level preconditioner provides some speed-up (see also [16]), however, there is still a need for faster and well-scalable solvers because of the anticipated sizes of the forthcoming datasets.

In our collaboration, we have for the moment installed the preAlps library at NERSC supercomputing center, Lawrence Berkeley National Laboratory, and we currently interact with Paris 7 researchers to interface our code with their map making code. The interfacing phase will be finalized soon and we will perform numerical experiments in the following weeks. In the meantime, we have also collaborated on studying a new linear systems solving approach, first introduced in [8] for Wiener Filter application and in [13] for map-making. This approach is based on so-called *messenger-field* algorithm with *cooling* and it became quickly popular in the field. However, as we have shown in [15], the messenger field corresponds to fixed-point iterations of an appropriately preconditioned initial system. We then argue that a conjugate gradient solver applied to the same preconditioned system will in general ensure at least a comparable and typically better performance in terms of the number of iterations to convergence and time-to-solution. This is illustrated in Figure 9, where we compare the convergence of PCG and messenger-field in two test cases in Wiener filter application.

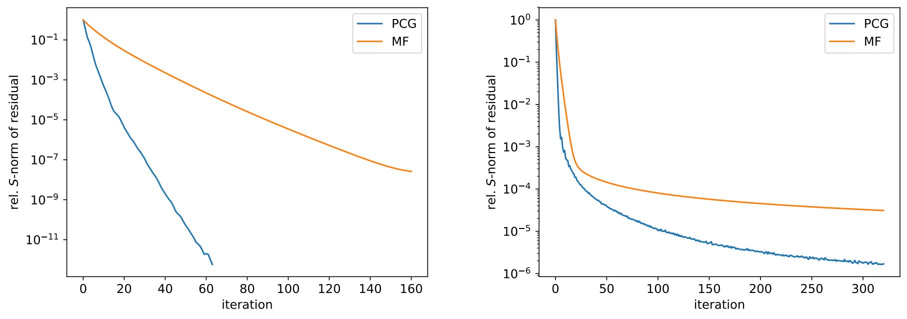

Figure 9: Convergence of PCG and messenger-field methods in two test cases in Wiener filter application.

The relative comparison of the performance of the messenger-field method *with* cooling and that of the PCG solver is still unclear and the freedom in defining the cooling scheme makes the mathematical analysis of this method and in particular its potential advantages over others difficult. However, we have observed in our numerical experiments (see Figure 10) that PCG was superior. We regard this important for setting the direction in which our future work should proceed. The results motivate us to look for better preconditioning techniques as the most promising way to address the problem of solving large systems in the CMB data analysis.

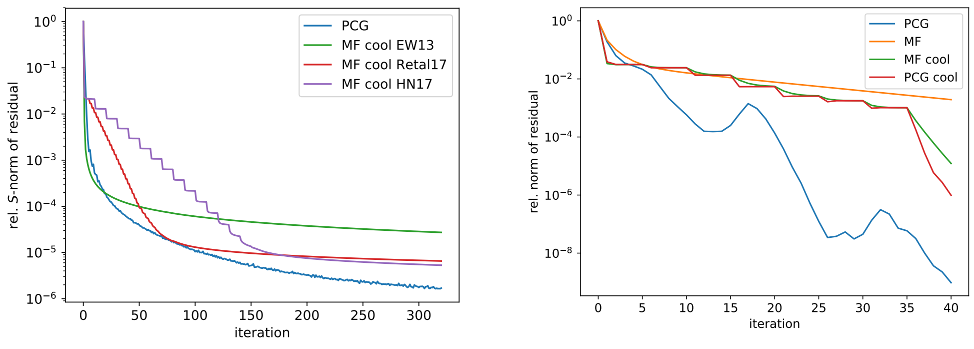

Figure 10: Convergence of PCG and messenger-field with various cooling schemes in Wiener filter (left) and map-making (right) application.

## 6 Conclusion

In this deliverable we have summarized our efforts on testing and integrating our software in several challenging applications.

In the context of communication avoiding iterative methods developed in Workpackage 4 (Inria), our results focused mainly on the elasticity 2D and 3D problems which are widely used in structural analysis. The structure is formed by several layers of materials whose resistance is measured when applying an external force or under a load. The matrix associated with this problem is symmetric positive definite, but getting the solution is very challenging for traditional iterative methods. Our experiments show that our research approaches solve efficiently the elasticity 2D and 3D problem. ECG consists in increasing the Krylov subspace in order to reduce the global number of iterations of the solver, while LORASC preconditioner consists in approximating the inverse of the Schur-complement to build a robust and scalable preconditioner which decrease considerably the number of iterations of an iterative solver. ECG can be used whenever classic CG is required and can be combined with other preconditioner such as Block Jacobi, and LORASC preconditioner can be used with any other iterative solver such as PETSC. This makes both approaches easy to integrate into existing scientific application. We have also found that ECG does not outperform existing approaches for the matrices provided to us by EDF. In that case we found that classic iterative methods converge already fast. We are currently working on interfacing ECG with the astrophysics application, and experiments will be performed in the following weeks.

## 7 Acknowledgments

This project is funded from the European Union's Horizon 2020 research and innovation programme under the NLAFET grant agreement No 671633. We thank the High Performance Computing Center North (HPC2N) at Umeå University for providing computational resources and valuable support.

## References

[1] A Message-Passing Interface standard (MPI). Accessed on Apr 8, 2018.

[2] Parallel Linear Algebra Software for Multicore Architectures (PLASMA). Accessed on Apr 8, 2018.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' guide*, volume 9. SIAM, 1999.

[4] Satish Balay, Shrirang Abhyankar, M Adams, Peter Brune, Kris Buschelman, L Dalcin, W Gropp, Barry Smith, D Karpeyev, Dinesh Kaushik, et al. Petsc users manual revision 3.7. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2016.

[5] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, et al. *ScaLAPACK users' guide*, volume 4. SIAM, 1997.

[6] V. Dolean, P. Jolivet, and F. Nataf. *An introduction to domain decomposition methods.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015. Algorithms, theory, and parallel implementation.

[7] I. S. Duff and J. K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Software*, 22(2):187–226, 1996.

[8] F. Elsner and B. D. Wandelt. Efficient wiener filtering without preconditioning. *A&A*, 549:A111, 2013.

[9] L. Grigori, F. Nataf, and S. Youssef. Robust algebraic schur complement based on low rank correction. Technical report, ALPINES-INRIA, Paris-Rocquencourt, 6 2014.

[10] L. Grigori, R. Stompor, and M. Szydlarski. A parallel two-level preconditioner for cosmic microwave background map-making. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 91:1–91:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[11] L. Grigori and O. Tissot. Reducing the communication and computational costs of enlarged Krylov subspaces conjugate gradient. NLAFET Working Note WN 13. Also as Research Report RR-9023, Feb 2017.

[12] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[13] K. M. Huffenberger and S. K. Næss. Cosmic Microwave Background Mapmaking with a Messenger Field. *ApJ*, 852:92, January 2018.

[14] F. Nataf, F. Hecht, P. Jolivet, and C. Prud'Homme. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. *SC13, (Denver, Colorado, United States)*, 2013.

[15] J. Papež, L. Grigori, and R. Stompor. Solving linear equations with messenger-field and conjugate gradients techniques - an application to CMB data analysis. NLAFET Working Note WN 19, Mar. 2018.

[16] G. Puglisi, D. Poletti, G. Fabbian, C. Baccigalupi, L. Heltai, and R. Stompor. Iterative map-making with two-level preconditioning for polarized Cosmic Microwave Background data sets. *ArXiv e-prints*, January 2018.

[17] N Stanley Scott, M Penny Scott, Phil G Burke, Timothy Stitt, Virginia Faro-Maza, Christophe Denis, and A Maniopoulou. 2DRMP: A suite of two-dimensional R-matrix propagation codes. *Computer Physics Communications*, 180(12):2424–2449, 2009.

[18] N. Spillane. An adaptive multipreconditioned conjugate gradient algorithm. 38:A1896–A1918, 01 2016.

[19] M. Szydlarski, L. Grigori, and R. Stompor. Accelerating the cosmic microwave background map-making procedure through preconditioning. *A&A*, 572:A39, December 2014.

[20] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi*, pages 167–188. Springer, 2014.