Implementation of Simultaneous Inversion of a Multi-shifted Dirac Matrix for Twisted-Mass Fermions within DDαAMG

Shuhei Yamamoto,[a,1,*] **Simone Bacchio**[a,1] **and Jacob Finkenrath**[a,1]

[a]*Computation-based Science and Technology Research Center, The Cyprus Institute,*
Aglantzia, Nicosia, Cyprus

E-mail: s.yamamoto@cyi.ac.cy

At physical light quark masses, efficient linear solvers are crucial for carrying out the millions of inversions of the Dirac matrix required for obtaining high statistics in quark correlation functions. Adaptive algebraic multi-grid methods have proven to be very efficient in such cases, exhibiting mild critical slowing down towards very light quark masses and outperforming traditional solver methods, such as the conjugate gradient method, at the physical point. We will discuss our implementations of simultaneous inversion of a (degenerate) Dirac matrix for twisted-mass fermions for multiple right-hand-sides (rhs) with multi-shifts and block-Krylov solvers. The implementation is carried out within the community library DDαAMG, which implements aggregation-based Domain Decomposition adaptive algebraic multi-grid methods. The block-Krylov solvers are provided via the Fast Accurate Block Linear krylOv Solver (Fabulous) library and can be used at coarser levels. Our code inverts Dirac matrices with different twisted-mass terms and for multiple rhs simultaneously and is thus also suitable for components within a typical lattice QCD simulation workflow, such as the rational approximation. We show preliminary results on scalability and compare the performance of our implementation when using different Block-Krylov solver techniques.

---

1. Introduction

In lattice QCD, inversion of Dirac matrices for a given right hand side (rhs) is a major computational task, performed repeatedly many times, for example, in measurement of observables and generation of gauge configurations. Due to its importance, a number of algorithms to efficiently solve linear systems are proposed. These algorithms make use of the nature of the matrix to be inverted, i.e., being large and sparse. Typically in lattice QCD, approaches based on Krylov-subspace methods are adopted. These methods are projection-based where the matrix is projected onto a subspace, called Krylov subspace, which is spanned by vectors obtained by repeated application of the matrix onto an initial guess, x_0. They work well for large and sparse matrices such as Dirac matrices. In particular, the conjugate gradient (CG) method has been used extensively. However, the CG method is known to suffer from critical slowing down when the quark mass becomes smaller, resulting in the exponential increase in convergence time.

To alleviate this issue, various methods have been tested, and it turned out that a class of solvers based on multigrid preconditioning is effective in reducing convergence time [1–5]. There are various implementations based on multigrid approaches. For clover-Wilson fermions, for example, Ref. [1] proposes a two-level multigrid approach based on Lüscher's inexact deflation. Alternatively, multigrid approaches are applied to the generalized conjugate residual (MG-GCR) method in [4, 6–8]. Here, we will focus on an aggregation-based domain-decomposition multigrid approach (DDαAMG), first proposed in Ref. [5] , adapted to degenerate twisted-mass fermions in Ref. [9], and generalized to non-degenerate twisted-mass fermions in Ref. [10]. For the sake of completeness, multigrid approaches are adopted also for other fermions. For instance, MG-GCR is extended for the case of domain-wall fermions [11], and DDαAMG for overlap fermions [12]. In what follows, we will discuss added capabilities of simultaneous inversion of multiple rhs and the usage of block-Krylov solvers in the DDαAMG software package.

2. DDαAMG

In this section, we briefly review the basics of DDαAMG. As is mentioned in Section 1, DDαAMG has been proposed to overcome the difficulties encountered by traditional Krylov-subspace methods such as CG. For this purpose, it incorporates two preconditioners, namely smoother and coarse-grid correction. The smoother acts on higher modes of functions on the lattice, and coarse-grid correction works on lower modes with the help of restriction and prolongation operators. The restriction operator extracts low-mode information and project onto a coarse lattice. Then, the prolongation operator interpolates a function on a coarse lattice to the original fine lattice. Various preconditioning procedures can be combined with coarse-grid correction at each coarse level. In this work, we have adopted the red-black Schwarz alternating procedure (SAP) [2] for the smoother, and for the coarse-grid correction, we have chosen to use algebraic multigrid (AMG) [13].

In application of SAP, the given lattice is divided into small blocks in a chessboard manner, and inversions of the local Dirac matrices, which are defined on the blocks, are performed. So SAP removes UV-modes from the error after its application [2].. As such, it suppresses the error in the higher end of the spectrum.

On the other hand, AMG focuses on the low modes of the lattice. It is formulated as follows. First, we define two operators, the prolongation operator, P, and the restriction operator, R. The operators, P and R, are defined algebraically using a few approximate low-mode eigenvectors based on the idea of Lüscher's inexact deflation [1]. Due to local coherence, this construction captures low-mode characteristics of the lattice [1]. Now, with the help of these operators, one can define a coarse lattice and a coarse-grid operator. The solution on the coarse grid is then prolonged back to the original lattice to produce an estimate. This construction can be extended to multiple levels by applying it recursively on the coarse lattices. Application of coarse-grid correction reduces the relative error in low modes. So if we combine the two preconditioners, the SAP and coarse-grid correction, we can suppress the relative error in both high and low end of the eigenspectrum.

A bottleneck of DDαAMG is its strong scalability at fix volume with increasing level of parallelization. Figure 1 depicts the scaling behavior on SuperMUC-NG equipped with Intel Xenon
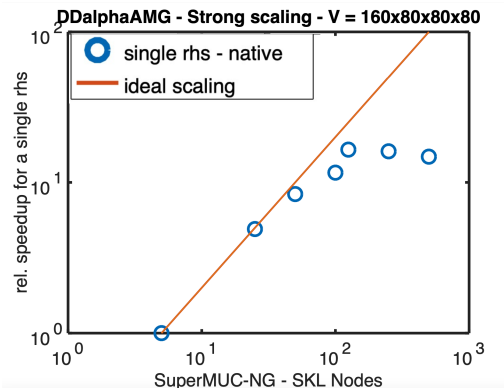

Figure 1: A scaling plot on the ensemble of $N_f = 2 + 1 + 1$ twisted mass clover with $a \sim 0.07$fm and $V = 80^3 \times 160$ at physical point simulated on SuperMUC-NG (Intel Xeon ("Skylake")) at The Leibniz Supercomputing Center. The orange solid line is the plot of ideal scaling, while the blue dots indicate the actual scaling behavior.

"Skylake" at The Leibniz Supercomputing Center. For higher node counts, the speed-up diverges from the ideal scaling, and stagnation of performance beyond 125 Skylake nodes is observed for a three-level MG approach. Given the current hardware trend, core counts per node will increase, and this means that the window for ideal scaling will shrink even further. To overcome this issue, we have implemented simultaneous inversion of multiple rhs, which we will discuss in the next section.

3. Multiple Right Hand Sides

Prior to implementation of multiple rhs, the original code at `https://github.com/sbacchio/DDalphaAMG` inverted each rhs one by one when it is given more than one rhs. To gain benefit from SIMD architecture, loops over lattice objects such as gauge fields and lattice vectors are vectorized explicitly. This vectorization was done by manually rewriting these loops with instruction sets for a specific SIMD extension. However, such explicit vecotrization brings inconvenience as we need to rewrite all vectorized loops for a different SIMD extension with its own instruction sets. Hence, in the version available at `https://github.com/sy3394/DDalphaAMG`, we

decided to use the autovectorization feature of compilers by unrolling loops over multiple rhs. In this way, we let compilers perform optimization analysis and vectorization. This change has improved portability of our code and simplified its maintenance.

In concrete, we used pragmas such as `_Pragma("unroll")`, `_Pragma("vector aligned")`, and `_Pragma("ivdep")` in front of these loops to signal compilers that they need to be vectorized. In particular, these pragmas are applied to a `for`-loop of a pre-determined iteration length: `for( int i=0; i<num_loop; i++)`. Then, loops over rhs are written in terms of this basic loop, and so the number of rhs is required to be a multiple of `num_loop`. To accommodate our code to this loop strategy, we have prepared a new data structure for the bundle of vectors and rewritten all low-level routines involving loops over rhs to respect this new data structure. Figure 2 schematically represents difference between old data structure and new data structure. As shown in the figure,
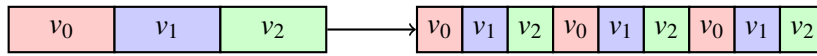

Figure 2: The comparison of how vectors are organized before and after implementation of simultaneous inversion of multiple r.h.s. The left figure show vector ordering in the old implementation, and the right figure in the new implementation.

in the old implementation, rhs are ordered so that they are placed one after the other. In the new implementation, however, a given entry from all vectors are gathered and placed together, and the chunk of the next entry is put consecutively to the chunk associated with the previous entry so that the vector index is now the fastest running index.

Using Intel® Advisor for profiling, we found that the compiler automatically shifted vectorization from 128 bits to 256 bits for our new implementation of multiple rhs, as is shown in Table 1. It also has added benefit of reducing data loading time for the matrix, as the data need to be loaded

| Num. R.H.S. | 1 rhs | | 4 rhs | | 8 rhs | |
Instruction Mix	SP Flops	DP Flops	SP Flops	DP Flops	SP Flops	DP Flops
128-bit	95.26%	86.59%	23.41%	4.99%	24.92%	3.60%
256-bit	2.58%	1.26%	60.68%	78.13%	74.02%	94.76%
Total	97.26%		84.03%		98.81%	

Table 1: Vectorization Reports

only once for all rhs. Moreover, the scaling problem with DDαAMG is mitigated via simultaneous inversion of multiple rhs. Figure 3 shows the scaling behavior of DDαAMG with simultaneous inversion of multiple rhs. As is shown in the figure, the breakdown of strong scaling is pushed away to higher parallelization. It now happens at around 512 nodes, widening the scalability window. The right plot in the figure shows the scaling behavior at the coarsest level. We see that the scaling behavior at the coarsest level is similar to the one at the top, which is based on the total computer time for inversion. This suggests that the stagnation of speedup with the increasing number of cores can be traced to that at the coarse levels.
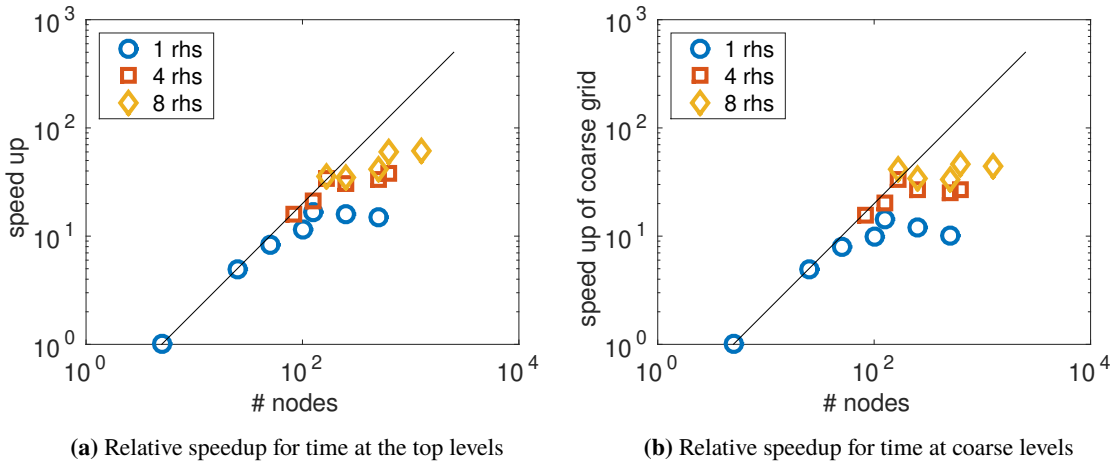
(a) Relative speedup for time at the top levels   **(b)** Relative speedup for time at coarse levels

Figure 3: Comparison of scaling behavior with different number of r.h.s. as a function of the number of nodes used for simulation.

4. Block Solvers

Our implementation of simultaneous inversion of multiple rhs also works well with block solvers, which can accelerate convergence via extension of the search space by combining Krylov spaces for all rhs. To enable various block Krylov solvers, we linked our code to Fast Accurate Block Linear krylOv Solver (Fabulous), developed by Inria (France) [14–16]. Fabulous provides block solvers, namely block GMRES and GCR, with various features such as detection of inexact breakdown, deflated restarting, and incremental QR factorization. This library also comes with two different orthogonalization schemes, Classical Gram-Schmit (CGS) and Modified Gram-Schmit (MGS) as well as their iterative variants each with a choice of blocking in orthogonalization. Details can be found in the documentation at https://gitlab.inria.fr/solverstack/fabulous.

With this library linked to DDαAMG, it now provides block solvers at each level as an option for the solver for the coarse-grid problem in AMG in addition to non-block FGMRES. The task is then to see if the use of block solvers leads to faster overall convergence at the top level. For this, we needed to tune some of the parameters related to Fabulous as well as DDαAMG to identify a parameter region where DDαAMG with block solvers performs better than without the block solvers.

To set up the stage, we selected a three-level AMG with a FGMRES solver at the top level, as it is known to work best from our experience. The goal residual at the top level was set to 10^{-10}. Among the parameters associated with DDαAMG, the values of those parameters not related to coarse-grid inversion are set to be optimal values found in Ref. [9]. Among the various options for block solvers, inexact breakdown will not be considered in this study, as our implementation for multiple rhs always applies the Dirac matrix on at least num_loop many chunk of rhs. As for orthogonalization scheme, it was found that CGS without iterative application generally performs better in our setup. In addition, GCR is the only flexible solver available in Fabulous so that we have considered only GCR and FGMRES as a middle-level solver. Now, the parameters to be tuned

are the choice of solvers at the middle and bottom and their target residuals. In what follows, we first consider the solvers without deflation.

The parameter tuning was conducted on Cyclone (Intel Xeon Gold 6248) at The Cyprus Institute using a lattice of size $48^3 \times 98$ at the physical point [17]. We have used 6 nodes and 32 cores from each node.

4.1 Tuning the residuals

To find optimal parameter values with block solvers, we considered three cases: (Middle Solver, Bottom Solver) = (Block, Block), (Block, Non-Block), and (Non-Block, Block). These cases were compared against the case (Non-Block, Non-Block). For each case, we inverted 4 rhs with different bottom residuals for selected middle residuals. Then, the results are compared with the (Non-Block, Non-Block) case, which is shown in Fig. 4. The slid lines are associated with purely non-block AMG, and other lines with mixed AMG. The result indicates that purely non-block AMG leads to a faster convergence time.
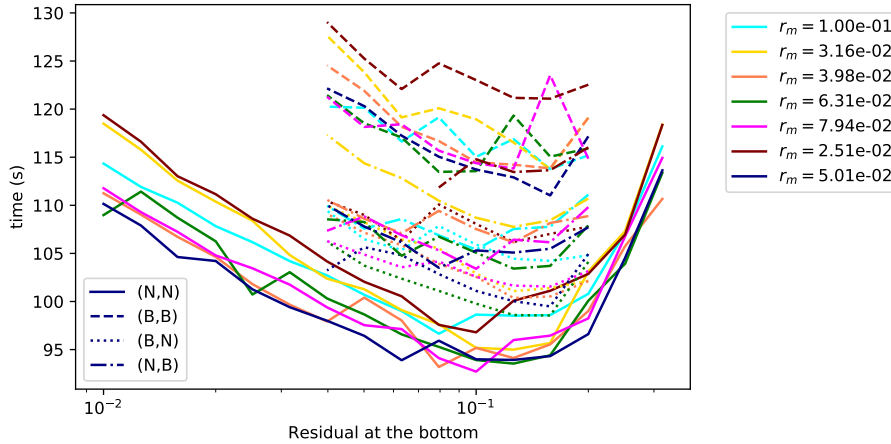

Figure 4: Comparison of convergence time as a function of the bottom residual, r_b, for different middle residuals, r_m between non-block AMG (solid lines) and mixed AMG. The dashed line is associated with (Block, Block), the dotted line with (Block, Non-Block), and the dash-dotted line with (Non-Block, Block), respectively.

To see whether this is due to overhead in each iteration of block solvers or simply an indication of their ineffectiveness, we compared the average iteration counts for convergence at each level over repeated calls to the coarse solver at the given level. Figure 5 shows the comparison between non-block AMG and (Block, Block) AMG. It reveals that the block solver as a middle solver requires more iterations than the non-block solver. This is perhaps due to the small number of iterations required for convergence at the middle level. At this level, the target residual does not need to be as small as the goal residual at the top level. In our case, the goal residual at the top level is set to 1×10^{-10}, but the middle residual of around $0.5 \sim 1 \times 10^{-1}$ is small enough to ensure fast convergence at the top. This is exhibited by Fig. 4 where the total convergence time takes the smallest values at $r_b \approx 1 \times 10^{-1}$ and $r_m \approx 7 \times 10^{-2}$. On the other hand, the right figure in Fig. 5 shows that the average iteration counts with a block solver, indicated by dashed lines, is smaller

than using a block solver below $r_b = 10^{-1}$. Thus, block solvers can be effective when used as the bottom solver in the three-level multigrid. However, even in this case, AMG with block solvers
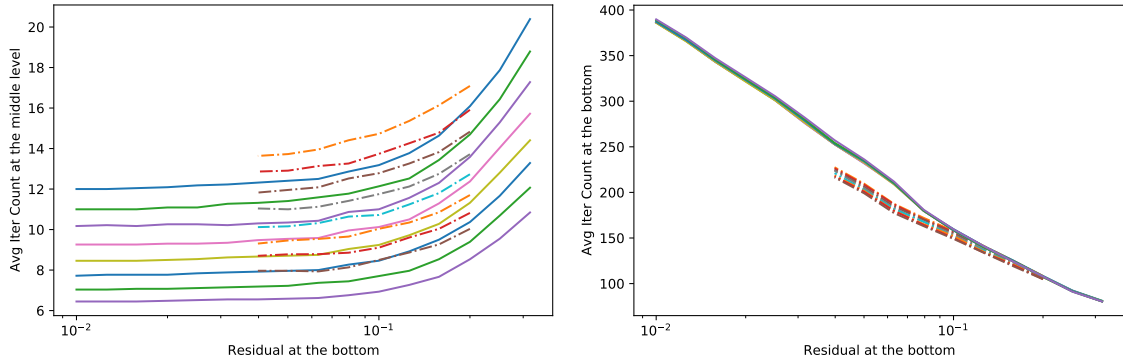

Figure 5: Comparison of average iteration counts between AMG with (Non-Block, Non-Block) in solid line and AMG with (Block, Block) in dashed line.

requires more time to converge compared with AMG without block solvers. To understand what constitutes the bottlenecks, we have used Score-P to trace and analyze performance of our code. It turned out that the overhead originates from two major sources, namely reordering of vector layout, accounting for about 25% of the total overhead, and global communication, which takes up about 75%. The first source comes from reordering of vector layout before and after each operator call within Fabulous. This is necessary, as Fabulous uses a more conventional layout of multiple rhs where vectors are ordered one after the other, while in our code, the vector index is the fastest running index. Also, Fabulous uses a user-provided matrix-vector-multiplication routine, and this routine is written for a new vector-running-fastest layout. The other source is that the block-solvers increase the number of MPI-reduction calls significantly, by an order of magnitude. This is due to the enlarged search space, requiring more inner products. This can be optimized by using non-blocking MPI-routines and pipelining. If available, they will lead to reduction of inversion time, to the extent that the block-solver becomes faster than the non-block version. Note that due to the increase in computational workload in application of the coarsest operator to multiple rhs, some communication can be hidden in the piplelined version, which will directly minimize the overheads.

Lastly, we investigated effectiveness of deflation for the block Krylov solvers, which is available for BGRO-DR in the newest version of Fabulous, where DR stands for deflated restarting. The results are summarized in Fig. 6. This shows a comparison of total iteration counts at the bottom with non-block bottom solver and block solvers with and without deflation. As can be seen, Fabulous solvers at the bottom with deflation are effective in reducing the total iteration counts. They become more effective with the increasing number of rhs on account of a larger size of block Krylov spaces. With 12 rhs, the number of the total iteration counts for Block BGCR-DR with $\delta\mu = 1$ is around half of the iteration count in case of the non-block GMRES, but due to the overheads, the total inversion time is roughly the same as for the non-block variant.

5. Outlook

Multigrid methods, such as DDαAMG, can significantly speed up inversion of linear systems including the one for the twisted-mass operator at physical quark masses. While this has circum-
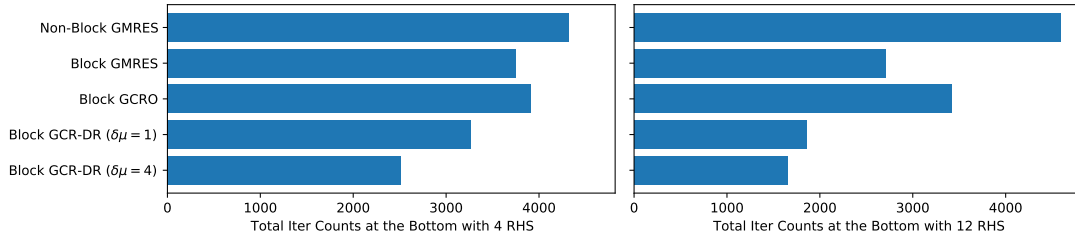
Figure 6: Comparison of total iteration count at the bottom with non-block FGMRES as the middle solver at $r_m = 0.1$ and $r_b = 0.1$. Here, $\delta\mu$ is a multiplicative factor in the definition of the Dirac matrix for twisted-mass fermions: $D_{TM} = D_W + i\delta\mu\mu\gamma_5$, provided as an parameter.

vented the problem of critical slowing down, the strong scalability of multigrid methods is limited by the coarse grid size.

We show that simultaneous inversion of multiple right-hand sides can overcome this issue and widens the scalability window while reducing data loading time and improving portability and maintainability of the code at the same time. Further improvements are possible by using block Krylov solvers at the coarsest level. We tested various block Kylov solvers with DDαAMG by linking it to an external library, Fabulous. This involved tuning some parameters, such as the residual on the coarse grids. In general, we found that block Krylov solvers can reduce iteration count by up to a factor 2 in comparison to the optimized native GMRES method, especially if we use BGCRO-DR. Note that BGCRO-DR can be used without an additional shift parameter on the coarsest grid in case of twisted mass fermions.

At the moment, the use of Fabulous comes with an additional overhead due to reordering of vectors in each DDαAMG kernel call and due to blocking of global communication functions. This overhead can be mitigated by using pipelined block-Krylov solvers and modifying inner-product routines, which could lead to improvements in inversion time by up to a factor 2.

6. Acknowledgments

References

[1] M. Luscher, *Local coherence and deflation of the low quark modes in lattice QCD*, *JHEP* **07** (2007) 081 [`0706.2298`].

[2] M. Luscher, *Deflation acceleration of lattice QCD simulations*, *JHEP* **12** (2007) 011 [0710.5417].

[3] J. Osborn, R. Babich, J. Brannick, R. Brower, M. Clark, S. Cohen et al., *Multigrid solver for clover fermions*, *PoS* **LATTICE2010** (2010) 037 [1011.2775].

[4] R. Babich, J. Brannick, R. Brower, M. Clark, T. Manteuffel, S. McCormick et al., *Adaptive multigrid algorithm for the lattice Wilson-Dirac operator*, *Phys. Rev. Lett.* **105** (2010) 201602 [1005.3043].

[5] A. Frommer, K. Kahl, S. Krieg, B. Leder and M. Rottmann, *An adaptive aggregation based domain decomposition multilevel method for the lattice wilson dirac operator: multilevel results*, 1307.6101.

[6] J. Brannick, R. Brower, M. Clark, J. Osborn and C. Rebbi, *Adaptive multigrid algorithm for lattice qcd*, *Phys. Rev. Lett.* **100** (2008) 041601 [0707.4018].

[7] M. Clark, J. Brannick, R. Brower, S. McCormick, T. Manteuffel, J. Osborn et al., *The Removal of critical slowing down*, *PoS* **LATTICE2008** (2008) 035 [0811.4331].

[8] R. Babich, J. Brannick, R. C. Brower, M. A. Clark, S. D. Cohen, J. C. Osborn et al., *The Role of multigrid algorithms for LQCD*, *PoS* **LAT2009** (2009) 031 [0912.2186].

[9] C. Alexandrou, S. Bacchio, J. Finkenrath, A. Frommer, K. Kahl and M. Rottmann, *Adaptive Aggregation-based Domain Decomposition Multigrid for Twisted Mass Fermions*, *Phys. Rev. D* **94** (2016) 114509 [1610.02370].

[10] C. Alexandrou, S. Bacchio and J. Finkenrath, *Multigrid approach in shifted linear systems for the non-degenerated twisted mass operator*, *Comput. Phys. Commun.* **236** (2019) 51 [1805.09584].

[11] S. D. Cohen, R. Brower, M. Clark and J. Osborn, *Multigrid Algorithms for Domain-Wall Fermions*, *PoS* **LATTICE2011** (2011) 030 [1205.2933].

[12] J. Brannick, A. Frommer, K. Kahl, B. Leder, M. Rottmann and A. Strebel, *Multigrid Preconditioning for the Overlap Operator in Lattice QCD*, *Numer. Math.* **132** (2016) 463 [1410.7170].

[13] P. Wesseling, *Introduction to multigrid methods*, tech. rep., Institute for Computer Applications in Science and Engineering (ICASE), 1995.

[14] M. Robbé and M. Sadkane, *Exact and inexact breakdowns in the block gmres method*, *Linear Algebra and its Applications* **419** (2006) 265 .

[15] R. B. Morgan, *Restarted block-gmres with deflation of eigenvalues*, *Applied Numerical Mathematics* **54** (2005) 222 .

PoS(LATTICE2021)536

[16] E. Agullo, L. Giraud and Y.-F. Jing, *Block gmres method with inexact breakdowns and deflated restarting*, *SIAM Journal on Matrix Analysis and Applications* **35** (2014) 1625 [https://doi.org/10.1137/140961912].

[17] ETM collaboration, A. Abdel-Rehim et al., *First physics results at the physical pion mass from $N_f = 2$ Wilson twisted mass fermions at maximal twist*, *Phys. Rev. D* **95** (2017) 094515 [1507.05068].

PoS(LATTICE2021)536