Domain Decomposition of the Dirac operator in the QUDA library

Simone Bacchio, Kate Clark, Jacob Finkenrath, Balint Joo, Ferenc Pittler, Jiqun Tu, Mathias Wagner, Evan Weinberg

LATTICE 2024 LIVERPOOL

Problem

We are developing a generic 4-dimensional domain decomposition aiming to support algorithms such as the Red-Black Schwarz Alternating Procedure (SAP), time-slice domains and multilevel algorithms in the QUDA library.

QUDA library

QUDA is a portable, open-source, highly-optimized library for lattice QCD calculations on GPUs, supported by `NVIDIA`. It offers an implementation of most-used Dirac operators, a state-of-the-art multigrid solver, and eigensolvers, as well as recently it fully supports multiple right-hand sides. It is available on GitHub [1] and provides C or C++ APIs.

Basic concepts

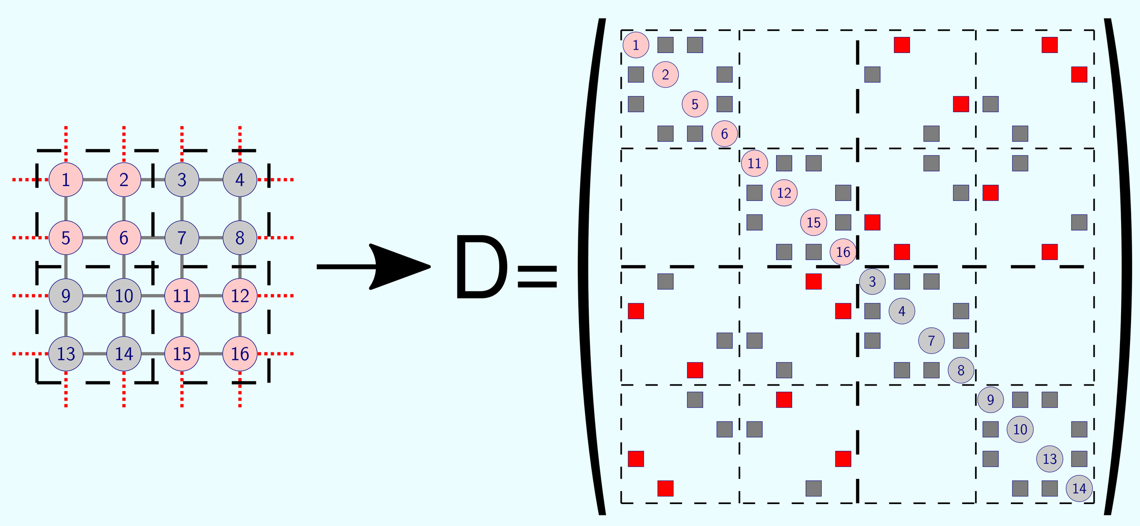

Figure 1: Schematic representation of the red-black reduced operator. The Dirac operator D has been ordered according to the red-black block decomposition of the lattice.

Example of Domain Decomposition:
Under the red-black domain decomposition, the Dirac operator assumes a block structure, e.g.

$$D = \begin{bmatrix} D_{rr} & D_{rb} \\ D_{br} & D_{bb} \end{bmatrix} \quad \text{and} \quad \psi = \begin{bmatrix} \psi_r \\ \psi_b \end{bmatrix},$$

where D_{rr}, D_{bb} has a block diagonal form and D_{rb}, D_{br} connecting the respective subdomains as schematically illustrated in Fig. 1.

Schwarz Alternating Procedure (SAP):
SAP is a well-known approach for solving the linear system $D\psi = b$ that exploits domain decomposition techniques [2].
Starting from an initial guess ψ^0, the solution ψ^n is updated iteratively as

$$\psi_r^n = \psi_r^{n-1} + D_{rr}^{-1}\left(b_r - D_{rr}\psi_r^{n-1} - D_{rb}\psi_b^n\right),$$

where ψ_r and ψ_b are the solutions on the red and black domains, respectively.

Additive SAP: All domains are updated *simultaneously* using the previous iteration for all.

Multiplicative SAP: Domains are updated *sequentially*, using on the right-hand side the newest available solution on the other domains.

Implementation

The goal of this work is to add in QUDA the infrastructure to easily implement *any* domain decompositions of the Dirac operator. The implementation is currently available in PR#1447 [3] and is under review. Hereafter we summarize the key features of the implementation.

Strategy: Although the domain-decomposition (DD) is a property of the Dirac operator, we figured out it was easier to implement it as a property of the color-spinor field (CSF). All CSFs have now an additional parameter structure, `DDparam`, that can be dynamically set to switch on DD features of the field. E.g. the application of D_{rb}, i.e. $y = D_{rb}x = P_r D P_b x$, can be expressed by setting the input vector as "black", i.e. $x_b = P_b x$, and the output vector as "red", i.e. $y_r = P_r y$, as follows:

Pseudocode for $y = D_{rb}x$: `x.dd_red_only(); y.dd_black_only(); applyD(y,x);`

All operators supported: All Dirac operators have been made *DD-aware* such that if the input and/or output CSF have DD enabled, then they act properly. This was achieved via two generic functions which are specialized depending on the DD kind. Namely,

```
constexp bool DDArg::isZero ( const Coord &x ) const
constexp bool DDArg::doHopping ( const Coord &x, int mu, int dir ) const
```

The first tells whether the in/output field is zero at a given coordinate, and the second if the hopping in a given direction should be performed. Coordinates are given globally to support "broken" domains.

Performance optimizations: The performance of the Dirac operator is left unchanged via dedicated compilations for each kind of DD, i.e. a dedicated `DDArg` structure is implemented. In the case of no DD, `DDArg = DDNo`, we have `DDNo::isZero = false` and `DDNo::doHopping = true`.
Additionally, if domains fit in the local lattice and Dirichlet-like boundary conditions are applied, then communications are switched off in all directions where not necessary. This is a key feature of most DDs, which exploit the Dirichlet boundaries to improve the scaling of the hopping term.

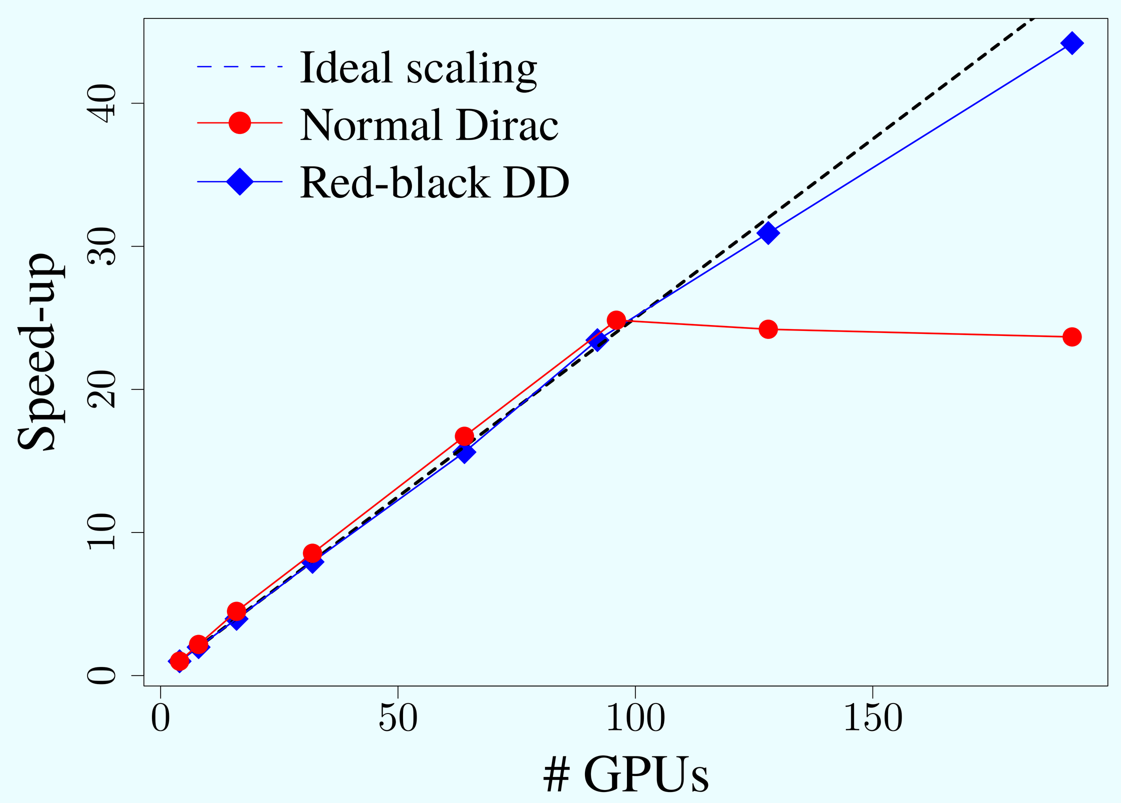

Figure 2: Scaling test of the Wilson Dirac operator using the full operator, red points, and the red-black decomposed operator, blue points, where red-red and black-black blocks are applied simultaneously but without hopping between them. The dashed line illustrates the ideal scaling. The tests were done on Juwels-booster NVIDIA A100 GPUs. The partitioning was chosen such that the red-black blocks would fit the local lattice, thus avoiding communications. The block size is 4^4 and this operator is commonly used in the additive SAP smoother.

Testing: We have also implemented projection functions that set to zero entries of the CSF that do not belong to active domains, `x.projectDD()`. This is used in unit testing for checking that e.g. $D_{rb}x == P_r D P_b x$. Additionally, this is also used as a *temporary* solution for unsupported applications and operators, e.g. such as even-odd (EO) preconditioning + DD.

TODO list: The effort is still a work in progress and we are planning the following activities next.

- Support of DD in BLAS routines, i.e. DD-wise reductions and linear algebra. These are necessary for linear solvers and efficient computation of e.g. $D_{rr}^{-1}b_r$, which is block-wise and local.
- Support of DD in multigrid (MG) operator, i.e. DD-wise restriction, prolongation and coarse Dirac. Required in the first place to use SAP as a smoother for the multigrid solver, and then to compute domain-wise solutions using MG in the case of multilevel (ML) algorithms.
- Support of DD in the application of EO preconditioned Dirac operator, i.e. Schur complement, which requires additional, so-called, *snake* terms [4]. Fig. 3 illustrates some examples of the snake terms. These terms are truncated if Dirichlet boundary conditions are enforced on each of the four hopping terms. But $\lfloor D_{oe}\rfloor\lfloor D_{eo}\rfloor\lfloor D_{oe}\rfloor\lfloor D_{eo}\rfloor \neq \lfloor D_{oe}D_{eo}D_{oe}D_{eo}\rfloor$, where $\lfloor\cdot\rfloor$ indicates the application of Dirichlet boundary conditions on the operator. In Ref [4] is it observed that these terms are fundamental for the convergence of the solver.
- Improvement of performance executing block-wise threads as done e.g. in the application of the prolongation and restriction operators of the multigrid solver.

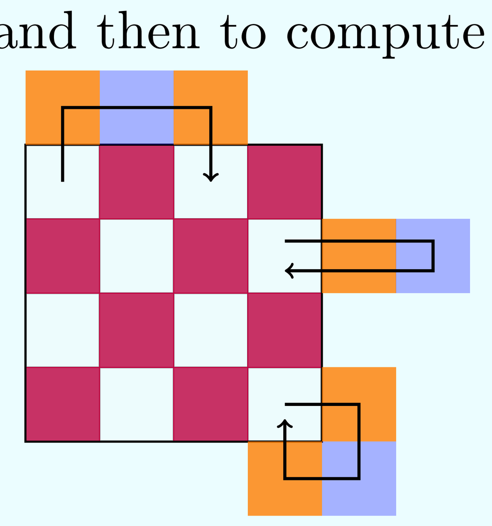

Figure 3: Snake terms of the $\hat{D}^\dagger\hat{D}$ operator. Credit [4].

References

[1] `https://github.com/lattice/quda`
[2] M. Luscher, arXiv:hep-lat/0310048 [hep-lat].
[3] `https://github.com/lattice/quda/pull/1447`
[4] J. Tu, *et al.*arXiv:2104.05615 [hep-lat]

Outlook

With a fully supported implementation of domain decomposition techniques in the QUDA library, the following features will be enabled:

- SAP preconditioning as smoother for the multigrid solver.
- Easy addition of custom DD for various applications.
- Improved scaling via computations on the local domains that increase the throughput while reducing communications.
- DD-wise solution of the Dirac operator, including support for MG solvers, and thus useful in e.g. master-field simulations to compute the solution simultaneously on many large domains of the lattice.
- First step towards multi-level simulations, where domains are updated in parallel for an exponential reduction of the noise, see Fig. 4.

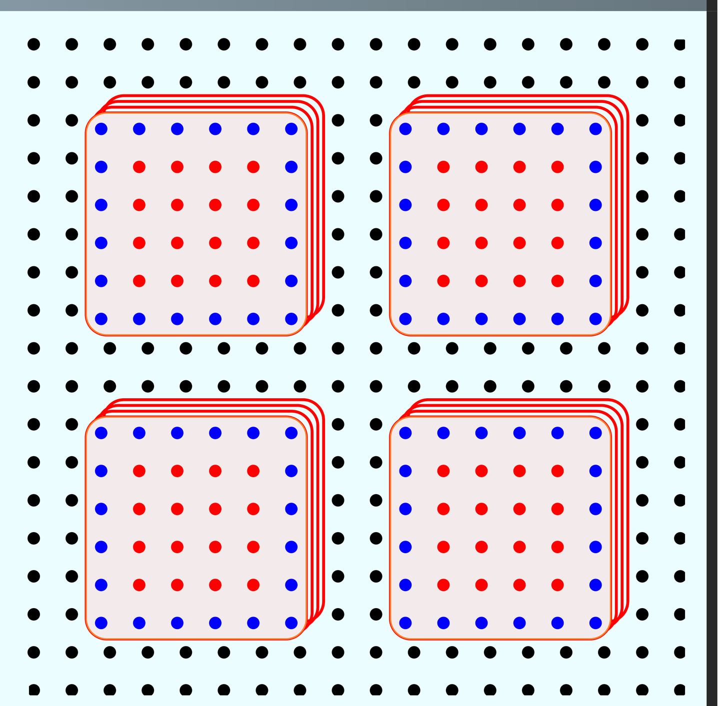

Figure 4: Multilevel DD.

Acknowledgements

This work has received funding through the Inno4scale project, funded by the European High-Performance Computing Joint Undertaking (JU) under Grant Agreement No 101118139. The JU receives support from the European Union's Horizon Europe Programme.