Solving Domain Wall Dirac Equation Using Multisplitting Preconditioned Conjugate Gradient

Jiqun Tu[1]

[1]**Department of Physics, Columbia University**

The 36th International Symposium on Lattice Field Theory, July 23, 2018 @ 16:10

Talk based on: Duo Guo, Robert D. Mawhinney, and Jiqun Tu, [arXiv:1804.08593].

Special thanks to Norman Christ, Chulwoo Jung, and Christopher Kelly.

The RBC & UKQCD collaborations

BNL and BNL/RBRC

Yasumichi Aoki (KEK)
Mattia Bruno
Taku Izubuchi
Yong-Chull Jang
Chulwoo Jung
Christoph Lehner
Meifeng Lin
Aaron Meyer
Hiroshi Ohki
Shigemi Ohta (KEK)
Amarjit Soni

UC Boulder

Oliver Witzel

Columbia University

Ziyuan Bai
Norman Christ
Duo Guo
Christopher Kelly
Bob Mawhinney
Masaaki Tomii
Jiqun Tu
Bigeng Wang
Tianle Wang
Evan Wickenden
Yidi Zhao

University of Connecticut

Tom Blum
Dan Hoying (BNL)
Luchang Jin (RBRC)
Cheng Tu

Edinburgh University

Peter Boyle
Guido Cossu
Luigi Del Debbio
Tadeusz Janowski
Richard Kenway
Julia Kettle
Fionn O'haigan
Brian Pendleton
Antonin Portelli
Tobias Tsang
Azusa Yamaguchi

KEK

Julien Frison

University of Liverpool

Nicolas Garron

MIT

David Murphy

Peking University

Xu Feng

University of Southampton

Jonathan Flynn
Vera Guelpers
James Harrison
Andreas Juettner
James Richings
Chris Sachrajda

Stony Brook University

Jun-Sik Yoo
Sergey Syritsyn (RBRC)

York University (Toronto)

Renwick Hudspith

SUMMIT at ORNL

Move Over, China: U.S. Is Again Home to World's Speediest Supercomputer

June 8, 2018

Figure 1: The New York Times's comment on SUMMIT becoming world's most powerful supercomputer.

Scaling on SUMMIT at ORNL

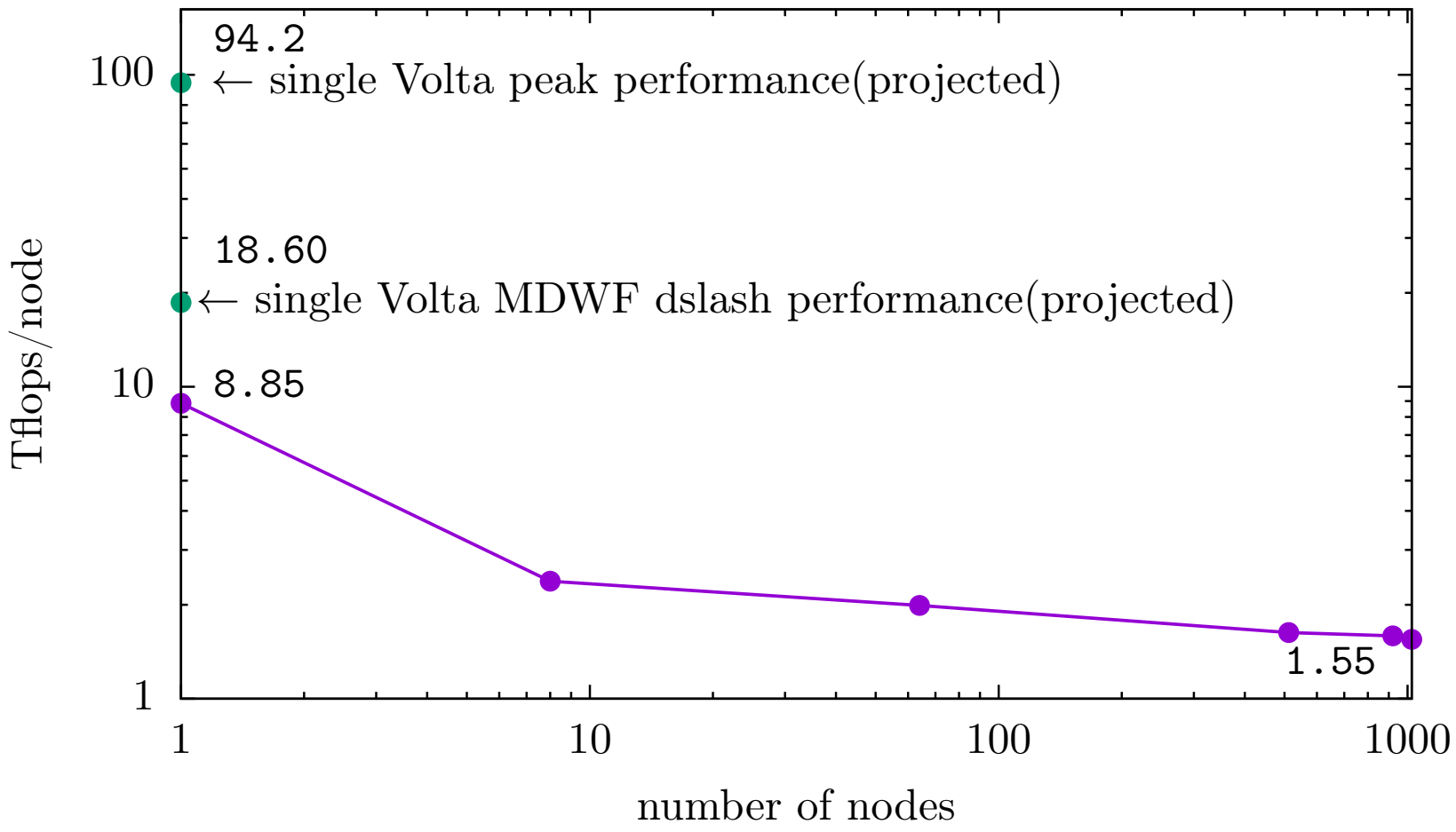


Figure 2: Half precision Möbius domain wall fermion CG weak scaling with local volume of $16 \times 12^3 \times 12$. 6 NVIDIA Volta GPUs on each compute node. Numbers provided by Chulwoo Jung.

Motivation

- Inter-processor communication is the bottleneck for Dirac equation solving.
- For measurement there are many approaches available to improve the situation: Lanczos, EigCG, split-grid, multigrid, etc.
- Not the case for evolution.
- Need an (better) algorithm to reduce the communication overhead and exploit the fascinating local GPU flops.
- Do more work *locally*!

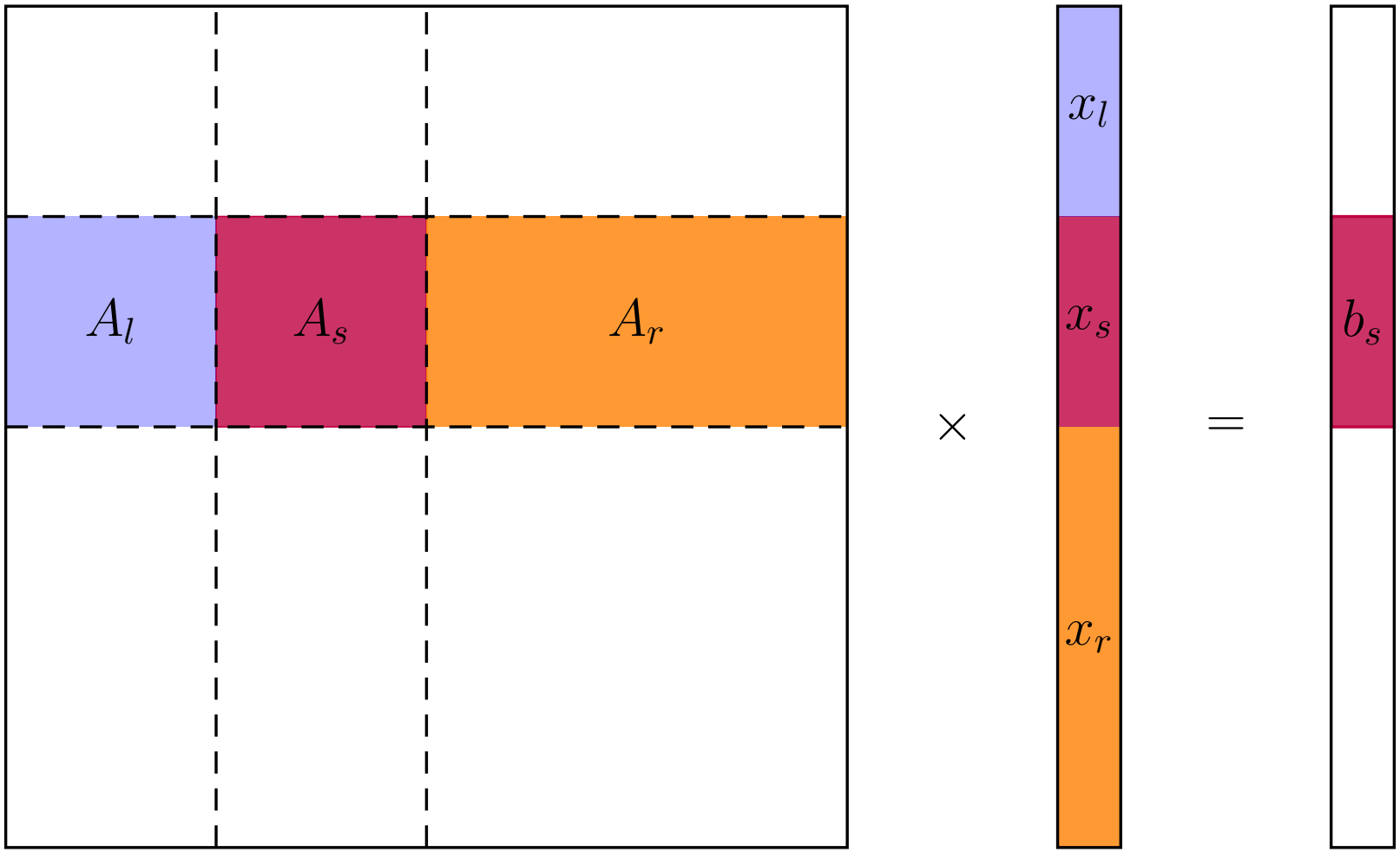
Previous Work

- Domain Decomposition/Multiplicative Schwarz[**M. Lüscher 2004**].
- Addtive Schwarz[**Y. Osaki 2000**] and [**R. Babich 2011**].

Multisplitting Algorithm

For reference see [**D. O'leary 1985**].

$$Ax = b : A_l x_l + A_s x_s + A_r x_r = b_s$$



Multisplitting Algorithm

Solve

$$A_l x_l + A_s x_s + A_r x_r = b_s,$$

Rearrange into an iterative form

$$\begin{aligned} A_s x_s^{(k+1)} &= b_s - A_l x_l^{(k)} - A_r x_r^{(k)} \\ &= b_s - \left(A x^{(k)} - A_s x_s^{(k)}\right) \\ &= r^{(k)} + A_s x_s^{(k)} \equiv \hat{b}_s^{(k)} \end{aligned}$$

For each cycle,

- use communication to calculate the right-hand-side $\hat{b}_s$.
- solve $A_s x_s^{(k+1)} = \hat{b}_s^{(k)}$ locally.
- the updated solution $x_s^{(k+1)}$ will be used to ready the next cycle.

Get A_s for each node by chopping off all off-block-diagonal terms: applying zero Dirichlet boundary condition.

Möbius Domain Wall Fermion

Even-odd preconditioning:

$$\begin{pmatrix} M_5 & -\kappa_b M^4_{eo} \\ -\kappa_b M^4_{oe} & M_5 \end{pmatrix} \begin{pmatrix} \psi_e \\ \psi_o \end{pmatrix} = \begin{pmatrix} \phi_e \\ \phi_o \end{pmatrix},$$

then instead we solve

$$D_{PC}\psi_e = \hat{\phi}_e, \; D_{PC} \equiv M_5 - \kappa_b^2 M^4_{eo} M_5^{-1} M^4_{oe},$$

$$M^4_{oe/eo} = D^w_{x,y}(b_5\delta_{s,t} + c_5 D^5)$$

$$D^w_{x,y} = \sum_\mu \left[(1+\gamma_\mu)U^\dagger_{x-\hat{\mu},\mu}\delta_{x-\hat{\mu},y} + (1-\gamma_\mu)U^\dagger_{x,\mu}\delta_{x+\hat{\mu},y}\right].$$

Using CG:

$$D^\dagger_{PC} D_{PC}\psi_e = D^\dagger_{PC}\hat{\phi}_e$$

The Normal Operator

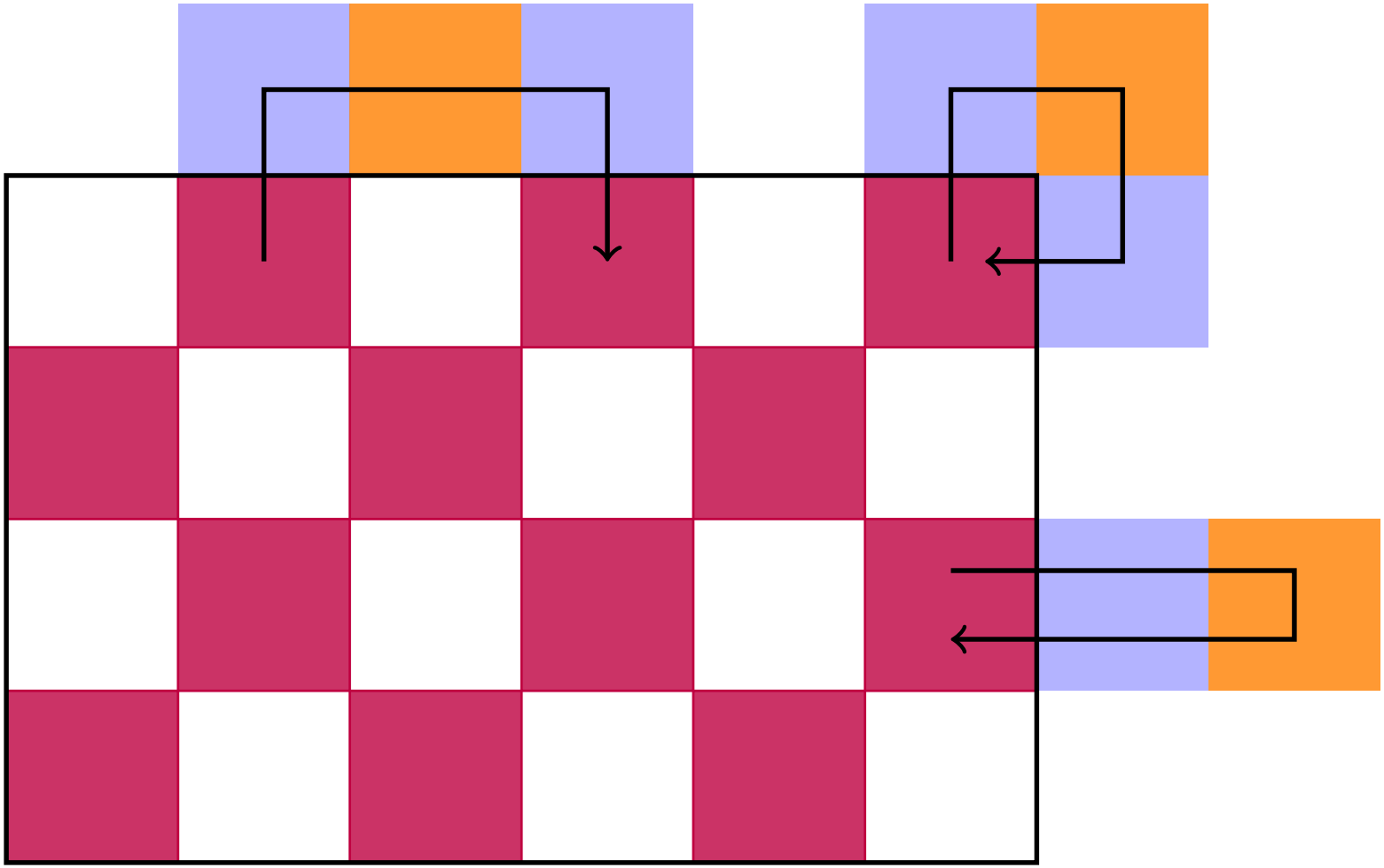
- 4 hopping terms in the normal operator:

$$
\begin{aligned}
A &= D_{PC}^{\dagger} D_{PC} \\
&= (M_5 - \kappa_b^2 M_{eo}^4 M_5^{-1} M_{oe}^4)^{\dagger} (M_5 - \kappa_b^2 M_{eo}^4 M_5^{-1} M_{oe}^4)
\end{aligned}
$$

- This means we need to enforce Dirichlet boundary condition on $D_{PC}^{\dagger} D_{PC}$ instead of the individual hopping terms $M_{eo/oe}^4(D_{x,y}^w)$.
- Need to include the *snake* terms: terms that hop out of the boundary and hop back.
- Seems obvious but not trivial to implement.

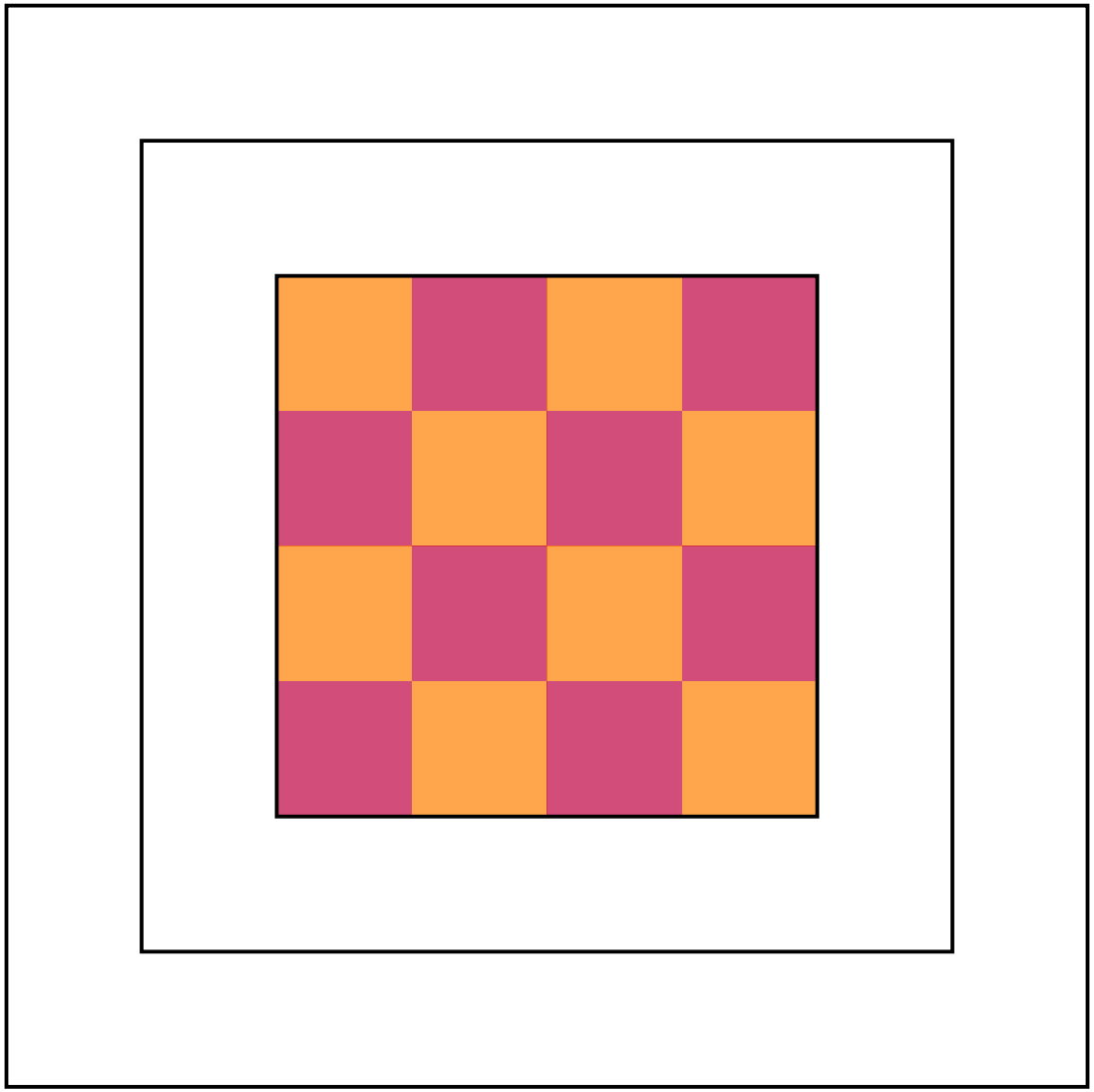
The Normal Operator

The snake terms:



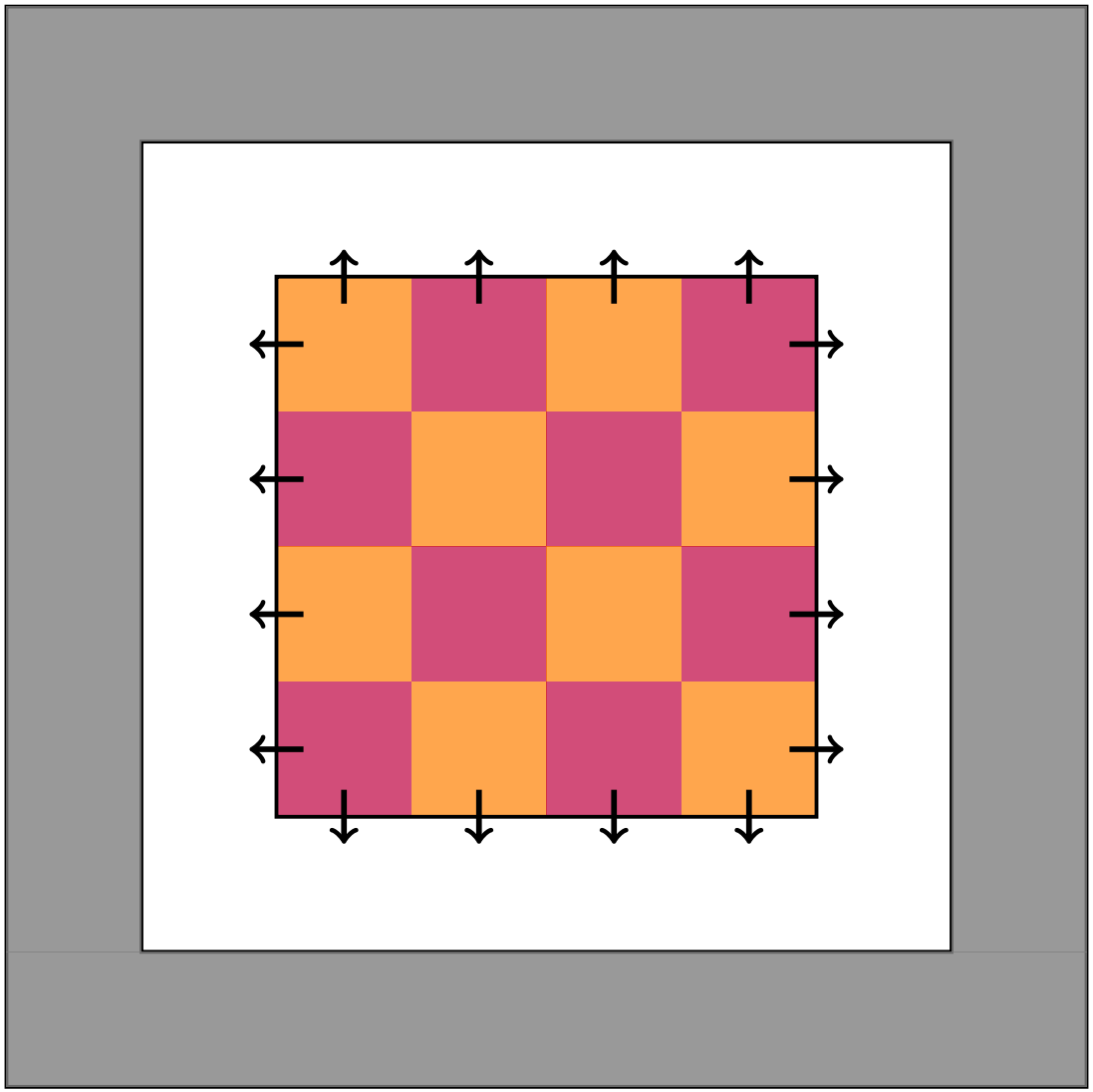
Dslash Implementation

before 1st hopping term



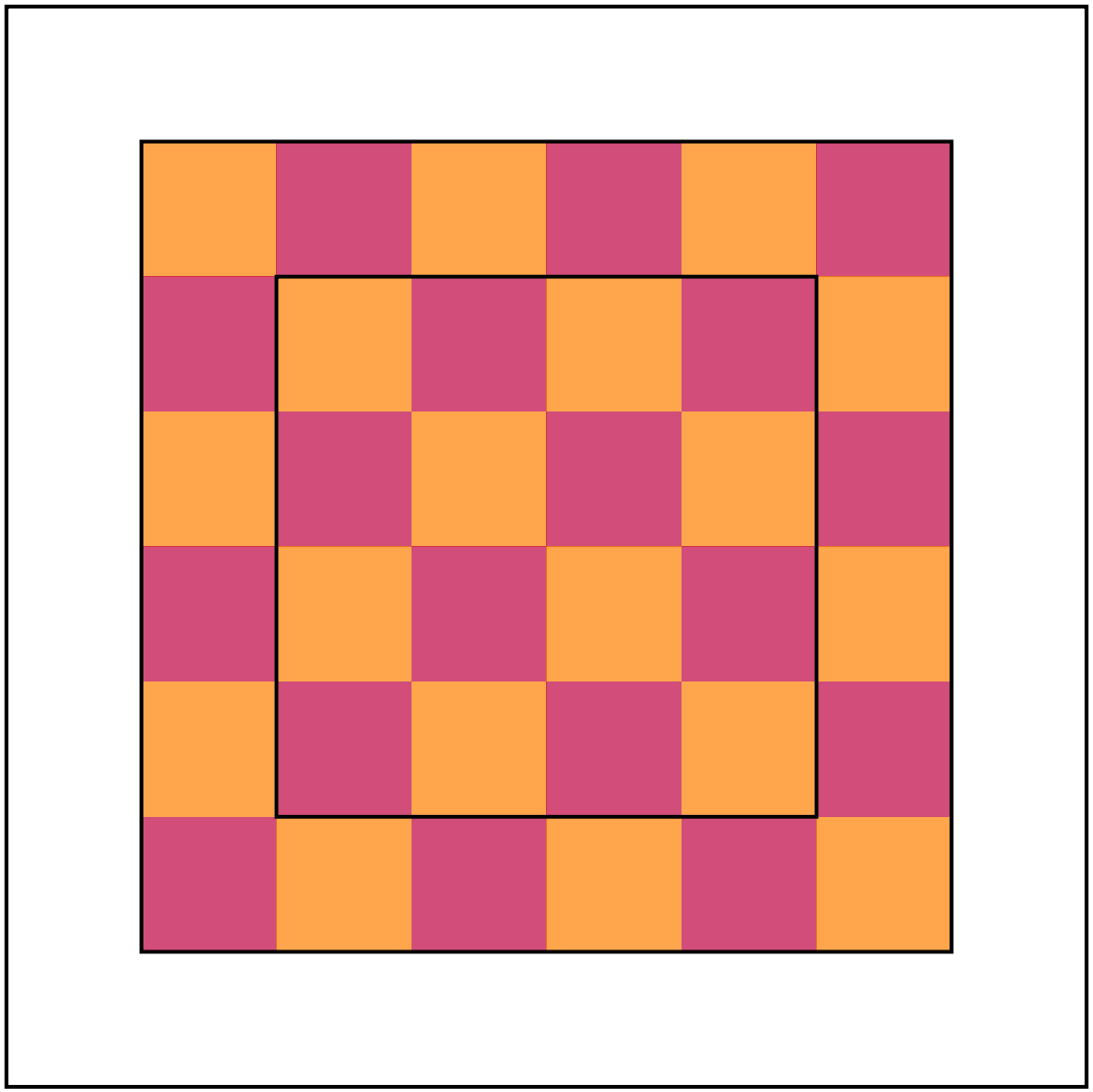
Dslash Implementation

before 1st hopping term



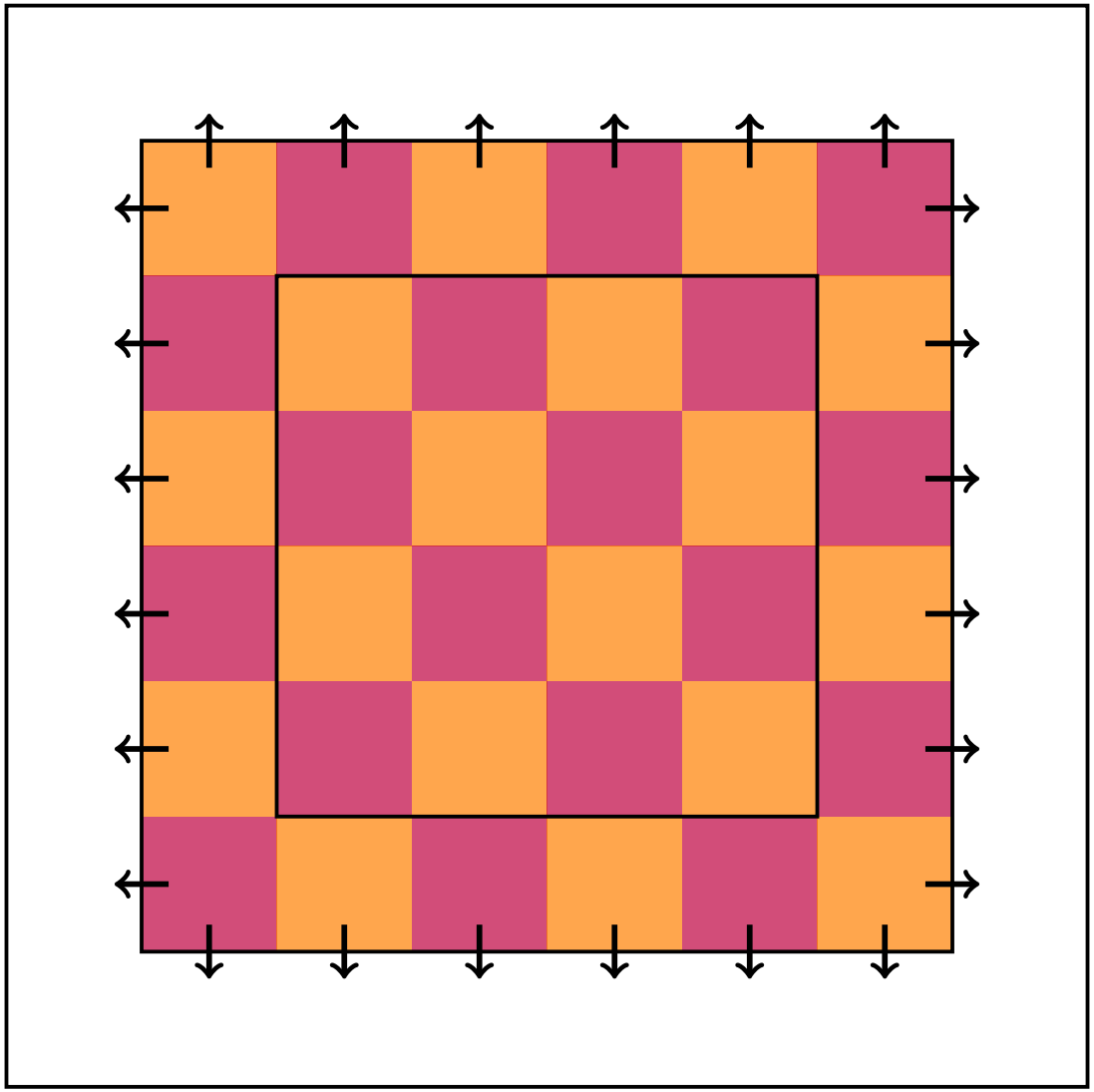
Dslash Implementation

after 1st hopping term



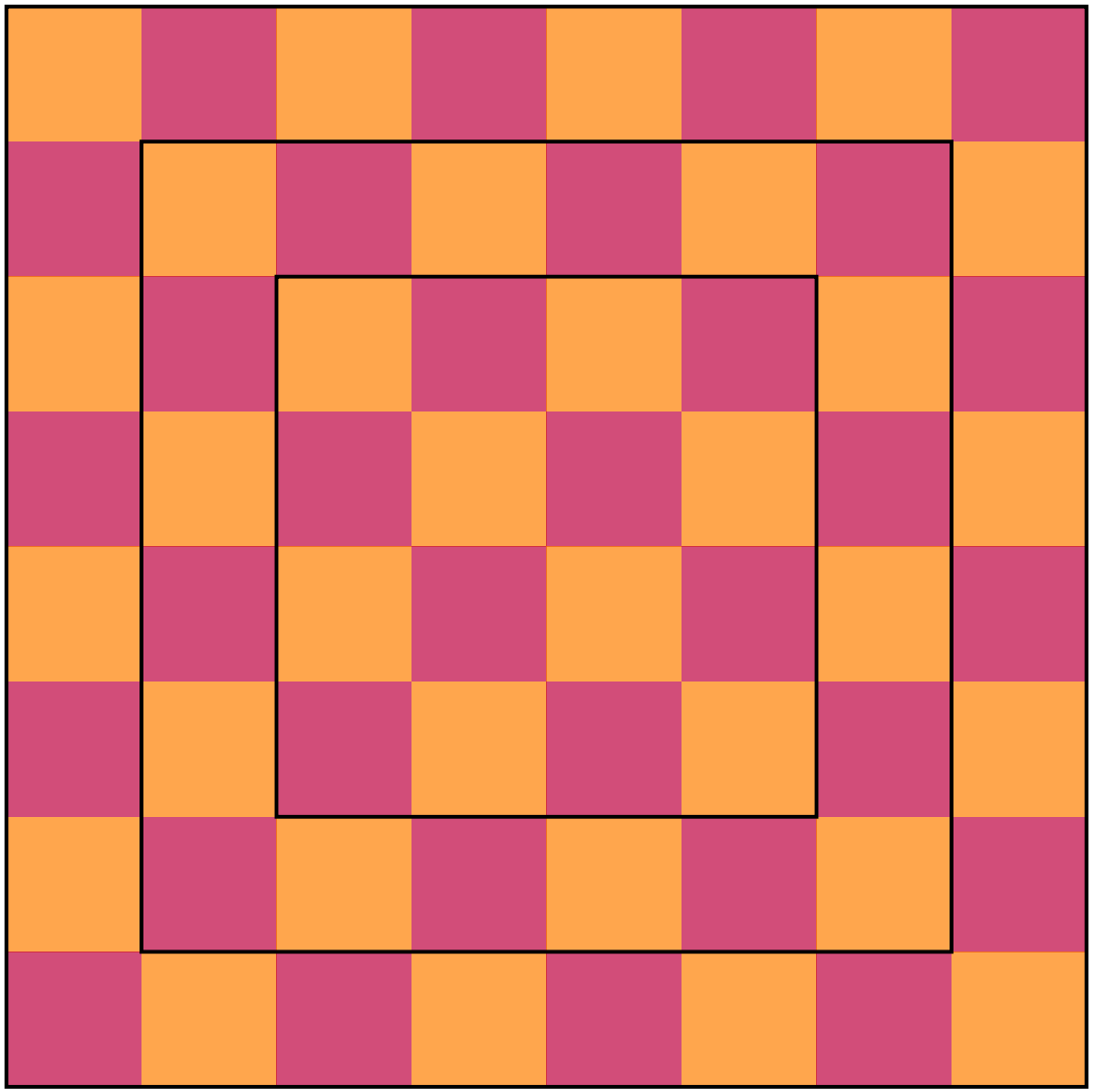
Dslash Implementation

before 2ed hopping term



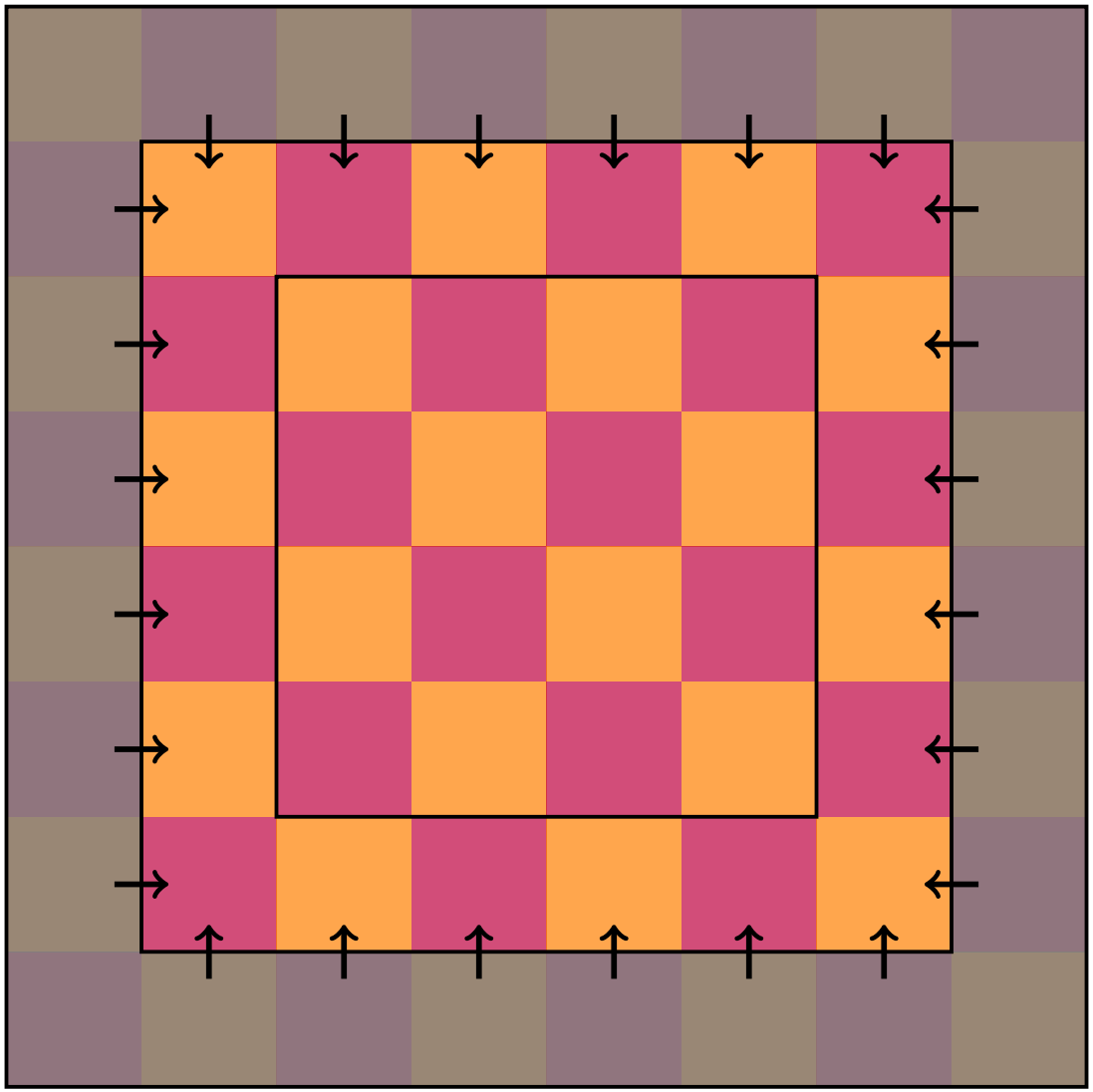
Dslash Implementation

after 2ed hopping term



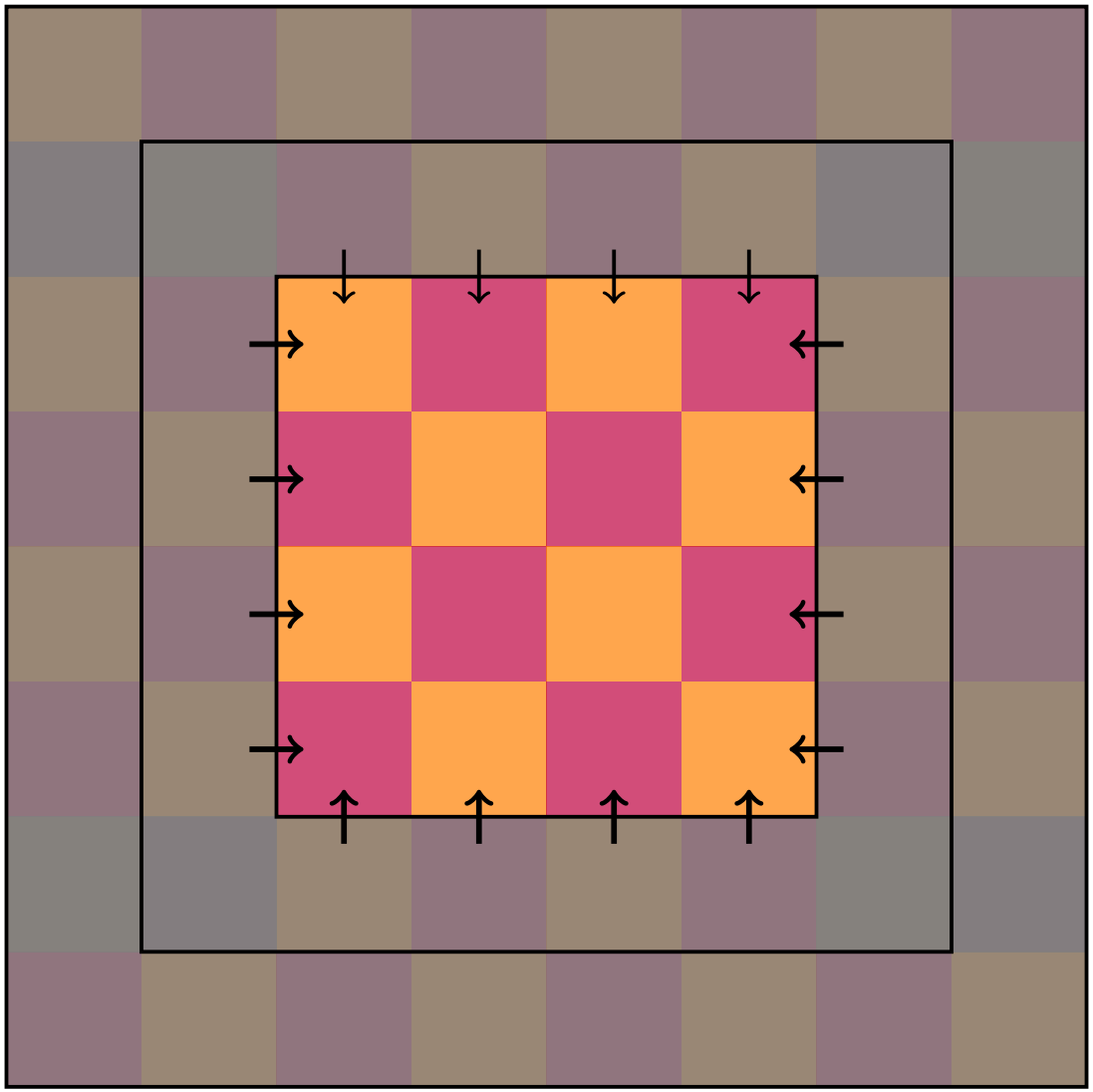
Dslash Implementation

before 3rd hopping term



Dslash Implementation

before 4th hopping term



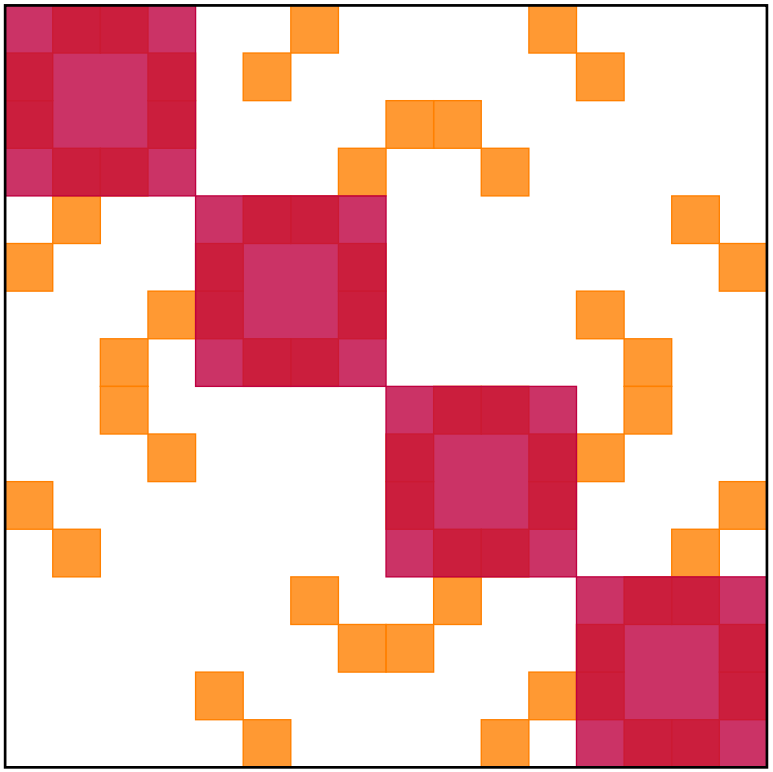
Multisplitting Algorithm

- The algorithm converges with inclusion of the snake terms.
- The convergence rate is slow.
- Similar to [**M. Lüscher 2004**] we use its first cycle with zero initial guess as a preconditioner for CG.
- We use plain CG for the preconditioner solve. Instead of setting a precision stopping condition we iterate for a fixed number of times(*the inner iteration count*).

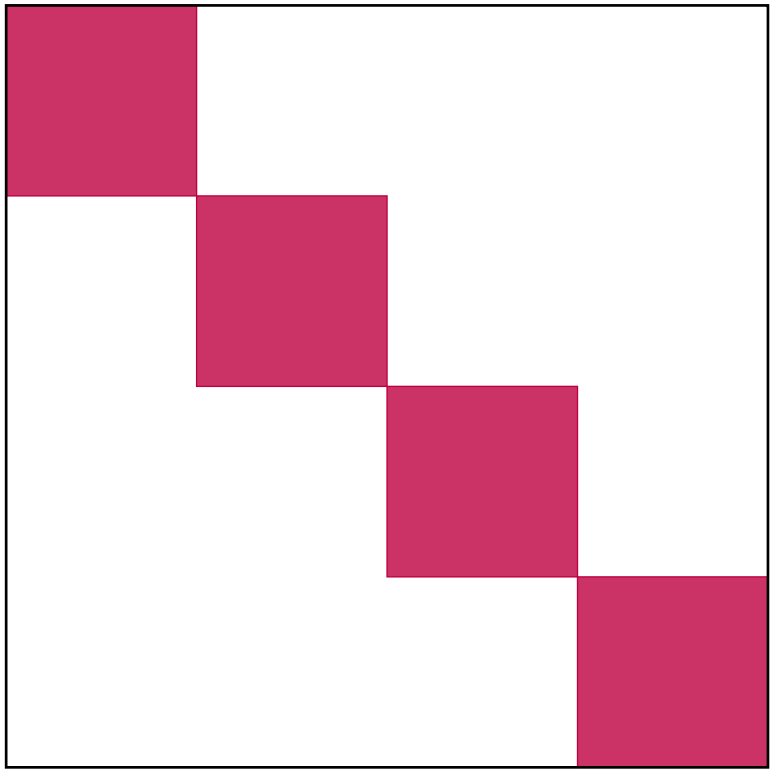
As a Preconditioner

$r_0 = b - Ax_0$
$z_0 = M^{-1} r_0$
$p_0 = z_0$
$k = 0$
while have not converged **do**
  $\alpha_k = \langle r_k, z_k \rangle / \langle p_k, Ap_k \rangle$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  $z_{k+1} = M^{-1} r_{k+1} \leftarrow A_s x_s^{(k+1)} = r^{(k)} + A_s x_s^{(k)}$
  only first cycle, zero initial guess, iterate a fixed number of times
  $\beta_k = \langle z_{k+1}, r_{k+1} \rangle / \langle z_k, r_k \rangle$
  $p_{k+1} = z_{k+1} + \beta_k p_k$
  $k = k + 1$
end while

As a Preconditioner



A



$M = \bigoplus_s A_s$

As a Preconditioner

- Although starting from a different origin, this is now effectively the same with addtive Schwarz *if* one treats the Dirichlet boundary condition correctly.
- Inclusion of the snake terms is crucial.
- Naming issue: [*A Unified Representation and Theory of Algebraic Additive Schwarz and Multisplitting Methods*, **A. Frommer 1997**].
- Multisplitting Preconditioned CG(MSPCG).

Result: $32^3 \times 64$

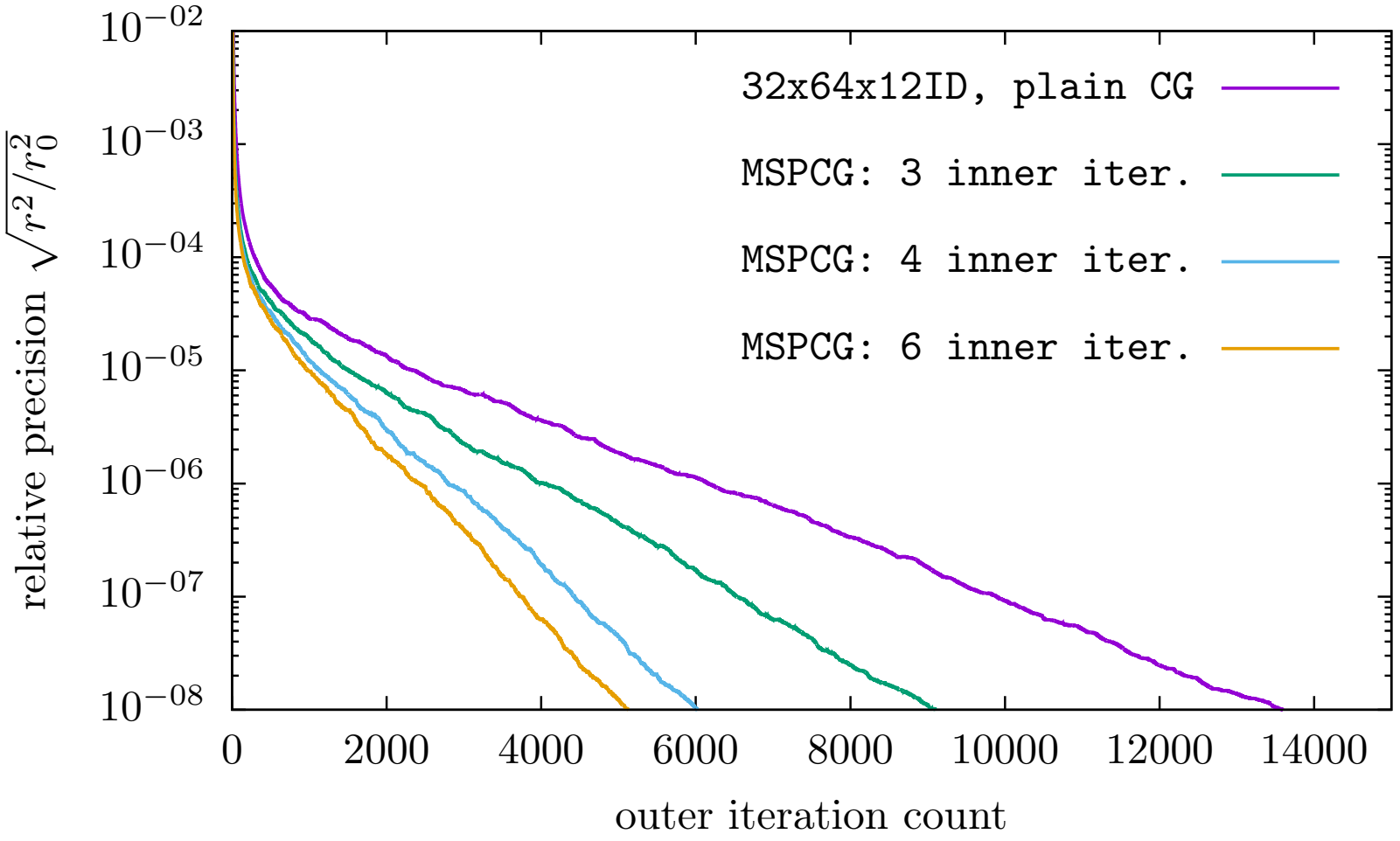


Figure 3: MSPCG solve on a $32^3 \times 64$ lattice ($a^{-1} = 1.37$ GeV) with physical pion mass. Test performed on CORI at NERSC on 128 KNL nodes.

Result: $32^3 \times 64$

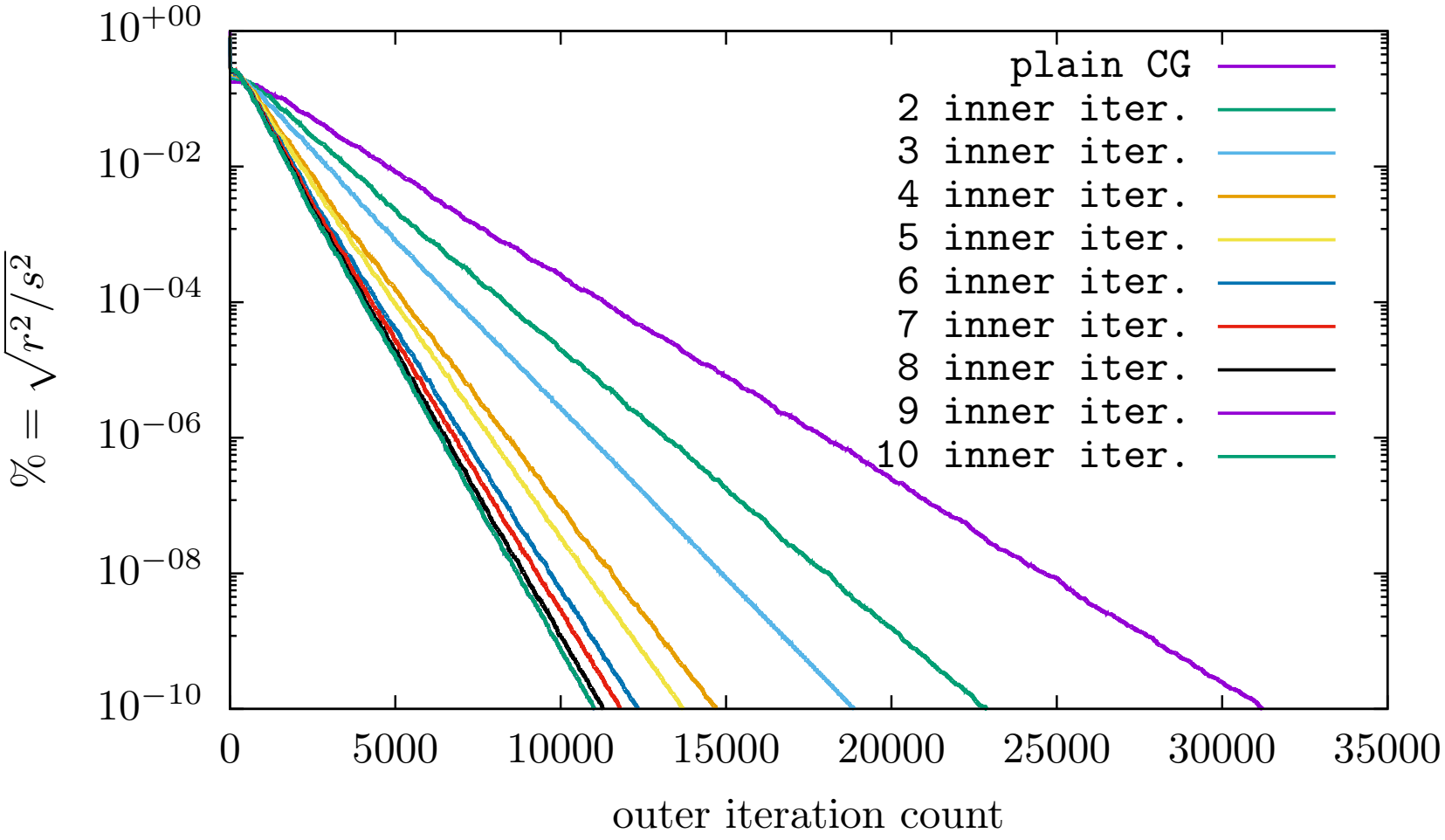


Figure 4: MSPCG solve on the same lattice. Test performed on 64 nodes at Piz Daint. Solving $D^\dagger D x = b$ instead of $D^\dagger D x = D^\dagger b$. Numbers from Kate Clark.

Result: $64^3 \times 128$

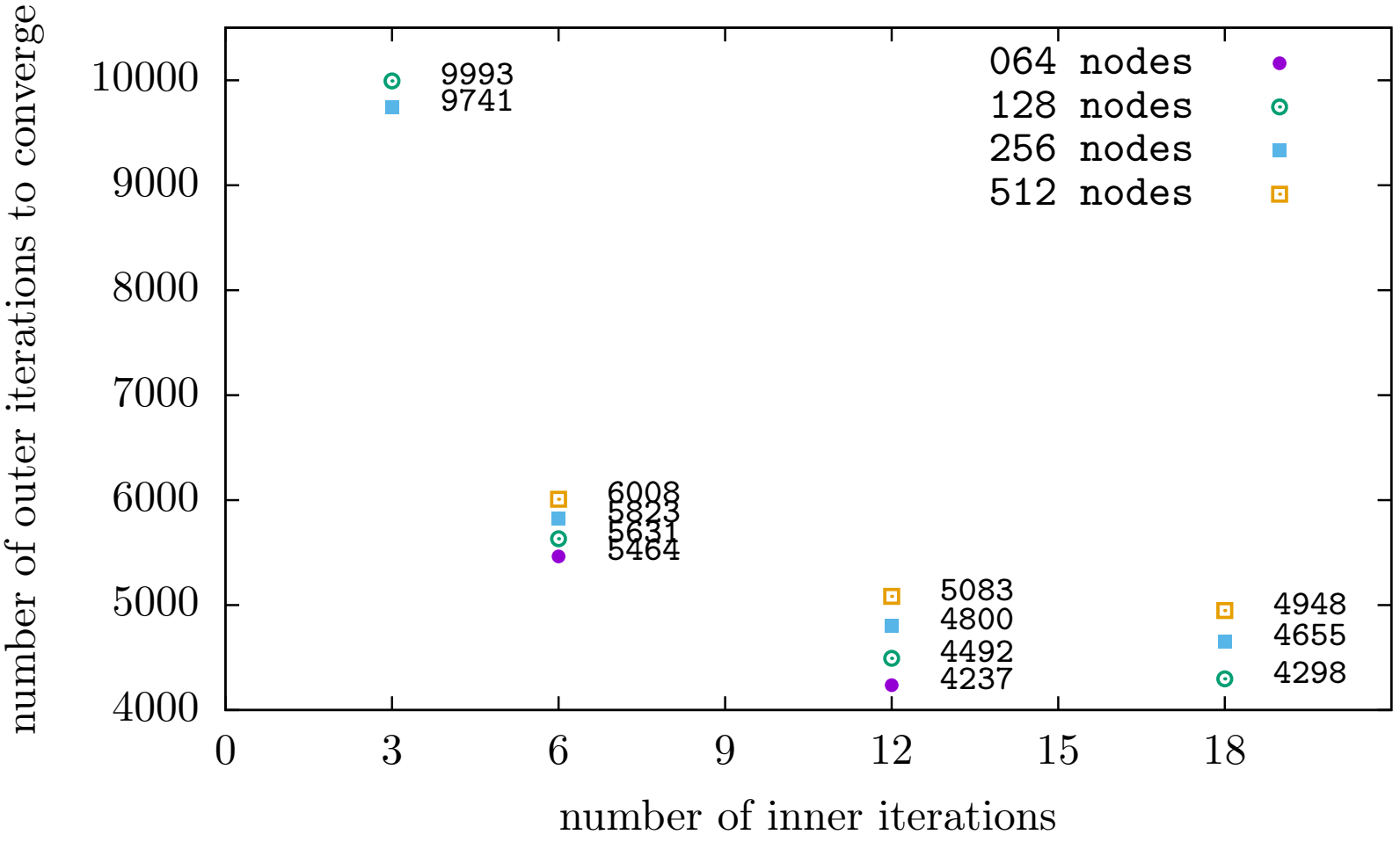


Figure 5: MSPCG solve on a $64^3 \times 128$ lattice ($a^{-1} = 2.36$ GeV) with physical pion mass. Plain CG takes 18092 iterations to converge to the same precision(10^{-10}). KNL at CORI.

Result: $80^2 \times 96 \times 192$

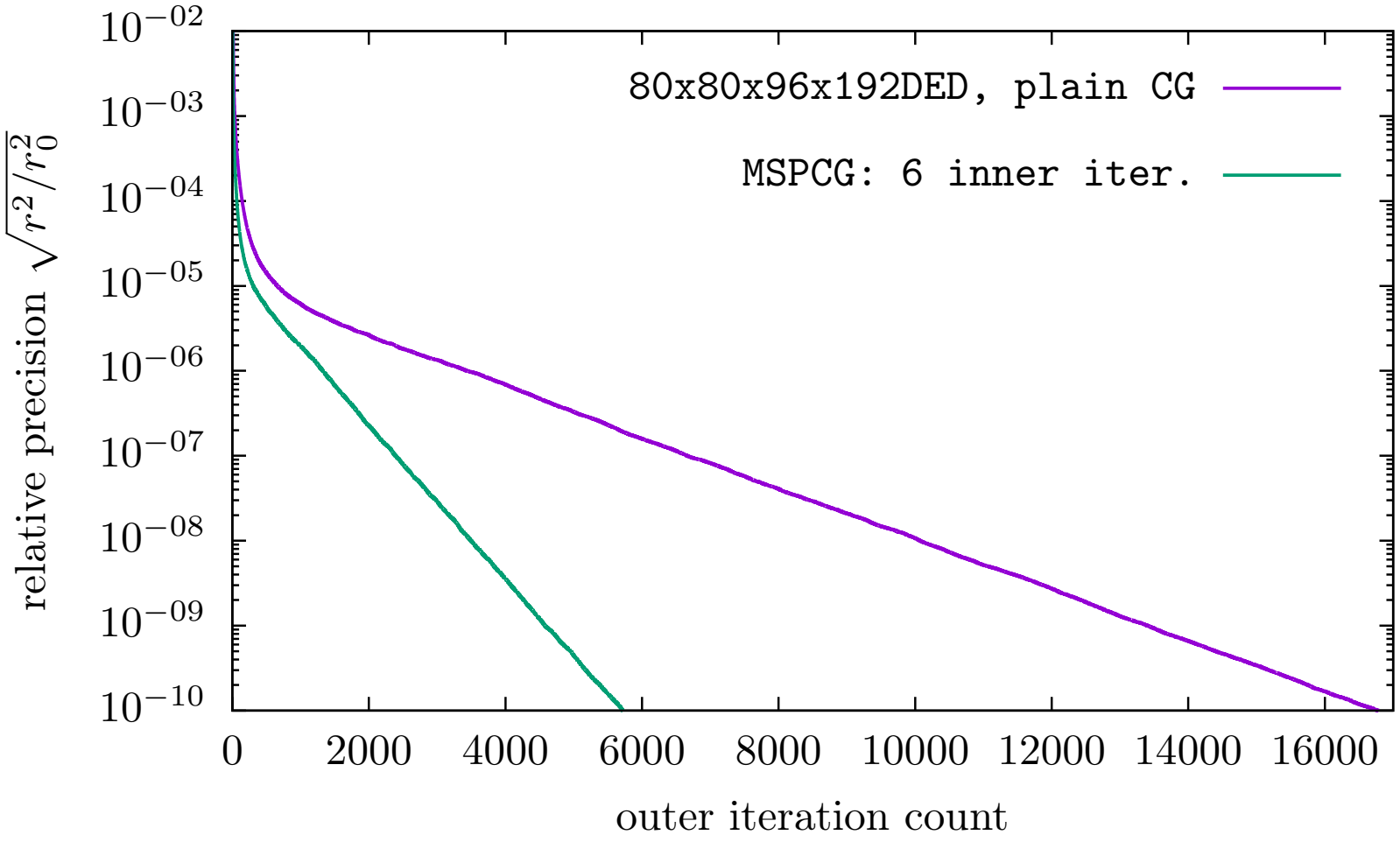


Figure 6: MSPCG solve on a $80^2 \times 96 \times 192$ lattice ($a^{-1} = 3.00$ GeV) with physical pion mass. Test performed on CORI at NERSC with 1024 KNL nodes.

Sloppy Preconditioner Solve

- We observe that the number of iterations for outer CG is greatly reduced even if the inner preconditioner is solved in a sloppy way, e.g. iterating only 3-6 times.
- Our observation is supported by several theoretical works, say, [**G. Golub 1999**] and [**V. Simoncini 2003**].
- Thus the number of preconditioner solve is a parameter that can be tuned to achieve maximum speed up.

SUMMIT

For $16 \times 12^3 \times 12$ local volume on 4 Volta GPUs,

preconditioner	14.13 Tflops

With the same local volume on 1024 6-Volta-nodes,

outer	1.55 Tflops/GPU

Assuming a factor of 3 in outer iteration count reduction with 6 inner iterations, the speed up from MSPCG is:

$$\left(\frac{3}{1.55}\right) \Big/ \left(\underbrace{\frac{6 \times \overbrace{1.87}^{\text{more work from snake terms}}}{14.13 \times (6/4)}}_{\text{precon. cost}} + \underbrace{\frac{1}{1.55}}_{\text{outer cost}} \right) = 1.65$$

Code Implementation

- First tested in `CPS`.
- Fully implemented in `Grid`[1] and `Quda`[2] with help from `Qlattice`[3].
- Great thanks to Kate Clark from NVIDIA.

[1] https://github.com/paboyle/Grid
[2] https://github.com/lattice/quda
[3] https://github.com/waterret/Qlattice

Conclusion

- The amount of inter-processor communication could be reduced at the expense of more local floating point computation by using the multisplitting algorithm as a preconditioner for CG.
- If the local floating point computation is cheap enough this greatly speeds up domain wall fermion Dirac equation solving.

Future Work

- On going work on `Quda`: Speed up preconditioner dslash as much as possible.
- The same approach is expected to work for staggered fermion as well.
- Spectrum analysis of the matrix A and the preconditioner M.