Lecture#3
Fermion Solver algorithm and Parallelization

Ken-Ichi Ishikawa

Hiroshima Univ.

1. Quark Solver in Lattice QCD

1-1 Quark solver in HMC and LQCD measurement.

$$\left\langle O[U, D[U]^{-1}] \right\rangle = \frac{1}{Z_{HMC}[0]} \int \prod_{n} d\phi^{\dagger}(n) d\phi(n) \prod_{n,\mu} d\Pi_{\mu}(n) dU_{\mu}(n)$$

$$O[U, D[U]^{-1}] \times \exp\left[-H[\Pi, U, \phi^{\dagger}, \phi] \right]$$

$$\approx \frac{1}{N} \sum_{j=1}^{N} O[U^{(j)}, \underline{D[U^{(j)}]^{-1}}]$$

$$\{U^{(1)}, U^{(2)}, U^{(3)}, \cdots, U^{(N)}\}$$

Noise reduction in correlation operators (observable):
See lecture by Peardon.

$$H[\Pi, U, \phi^{\dagger}, \phi] = \frac{Tr[\Pi^2]}{2} + S_G[U] + \underline{\left| (D[U])^{-1} \phi \right|}^2$$

- Any observables can be estimated via ensemble averaging.
- For LQCD, The observables are function of Link filed.
- Quark operators are replaced to the quark propagators via contraction.

$$\boxed{(D[U])^{-1}}$$

Most time consuming part of LQCD simulations

• Typical Quark action.

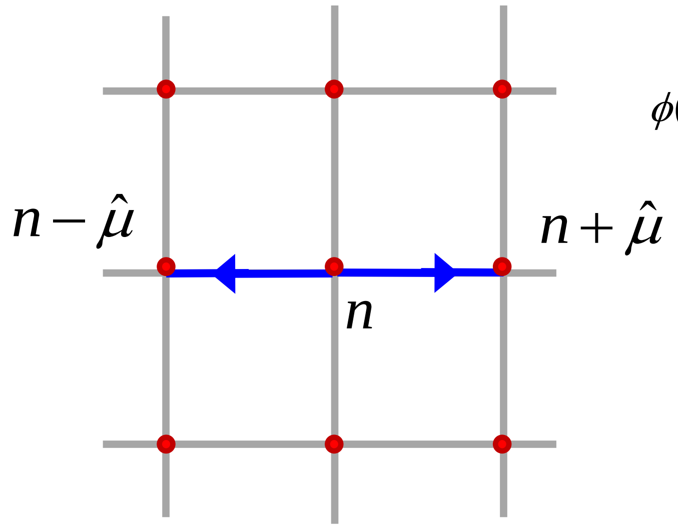
$$S_Q[U,\bar{q},q] = \sum_{f=u=d}\sum_{n,m}\bar{q}_f(n)D_f[U](n,m)\bar{q}_f(m)$$

$$\Rightarrow S_{PF}[U,\phi^\dagger,\phi] = \sum_n \left|\sum_m \left(D_f[U]\right)^{-1}(n,m)\phi(m)\right|^2$$

Chiral symmetiric fermions: Overlap, Domainwall type actions: See lecture by Nayaranan.

Wilson Fermion action. (qaurks)

$$\boxed{\begin{aligned} D_f[U](n,m) &= \delta_{n,m} - \kappa_f \sum_{\mu=1}^{4}\left[(1-\gamma_\mu)U_\mu(n)\delta_{n+\hat{\mu},m} + (1+\gamma_\mu)U_\mu(n-\hat{\mu})^\dagger\delta_{n-\hat{\mu},m}\right] \\ &= \left[1-\kappa_f H_{hop}\right](n,m) \qquad \kappa_f = \frac{1}{2(am_f+4)} \end{aligned}}$$

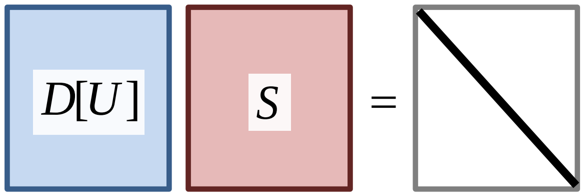


$$\phi(n) \leftarrow \phi(n) - \kappa\sum_{\mu=1}^{4}\left[(1-\gamma_\mu)U_\mu(n)\phi(n+\hat{\mu}) + (1+\gamma_\mu)U_\mu(n-\hat{\mu})^\dagger\phi(n-\hat{\mu})\right]$$

1st order Difference operation.
⇔single site hopping operation.

Wilson hopping operator is a fundamental building block in LQCD/LFT. (used in domainwall/overlap….)

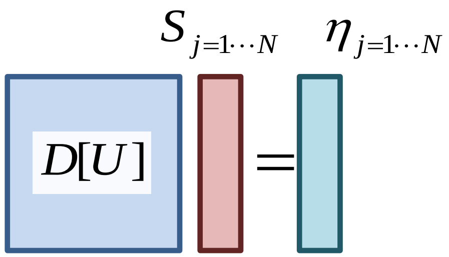
- How to obtain 1/D ?
  - We need quark propagators $\boxed{S = \left(D[U]\right)^{-1}.}$
  - This is done by solving $D[U]S = 1$



  - This is large scale linear equation. The size of matrix is extremely huge and impossible store the data on memory.
  - Fortunately we need only few columns of 1/D[U] on multiplied on a few vectors η.

$$S\eta_j = D^{-1}\eta_j \qquad j = 1,2,3,\cdots << N_{\mathrm{dim}}$$

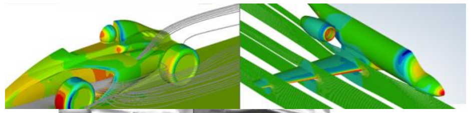
$$D[U]S_j = \eta_j$$

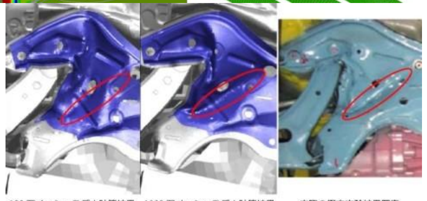


  - We need a large scale linear equation solver. => (numerical algorithms).

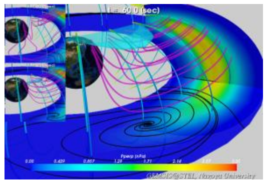
2. Solver algorithms for large scale linear equations.

- There are many opportunities for such a large scale linear (or nonlinear) equations.
  - Electromagnetic field analysis. ⇔Maxwell's eq.
  - Mechanical dynamics analysis for buildings. ⇔Newton's eq.
  - Fluid dynamics analysis for vehicles,
  - cars/airplanes/ships/trains...., ⇔Navier-Stokes eq.
  - And LQCD. ⇔ Lattice Dirac equations.
- There are various algorithms for LARGE SCALE linear equations.
  - Stationary iteration algorithms.
    - Jacobi Iteration.
    - Gauss-Seidel (GS) iteration.
    - Successive Over Relaxation (SOR) iteration.
    - ….
  - Adaptive iteration (Krylov Subspace) algorithms.
    - Conjugate Gradient (CG) algorithms
    - Bi-Conjugate Gradient (BiCG) algorithms.
    - Generalized Minimal Residual (GMRES) algorihtms.
    - ……





提供：(独)海洋研究開発機構・(社) 自動車工業会



- 2-1 Stationary iterative solvers
  - Based on Neumann/Taylor expansion for 1/(1-z).

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots \qquad \text{for } |z| < 1$$

  - Replace *z* to a matrix (1-*A*)

$$A^{-1} = \left(1-\left(1-A\right)\right)^{-1} = 1 + \left(1-A\right) + \left(1-A\right)^2 + \left(1-A\right)^3 + \cdots$$

  - This converges when the spectrum of $\left(1-A\right)$ is in $\left|\left(1-A\right)\right| < 1$
  - We need to solver *Ax = b.* The above series can be reduced to the following iteration.

$$\begin{aligned}
&(\text{step}\,0)\ x = b\\
&(\text{step}\,1)\ r = b - Ax\\
&(\text{step}\,2)\ x = x + r\\
&(\text{step}\,3)\ \text{if } |r| \text{ is sufficiently small then exit else goto step}\,1.
\end{aligned}$$

$$A^{-1}b = b + \left(1-A\right)b + \left(1-A\right)^2 b + \left(1-A\right)^3 b + \cdots$$

- 2-1 Stationary iterative solvers
  - The basic iteration does not converge when $|(1-A)| \geq 1$ and even if it converges it is slow usually.
  - Some modifications are available to improve the convergence.
- Splitting method
  - If we can split the matrix into two parts as

$$A = N + M, \qquad N \ : \ \text{easy to invert and } |N^{-1}M| < 1$$

  - We obtain the following modified equation and converging series.

$$Ax = b \Rightarrow (N+M)x = b \Rightarrow (1 + N^{-1}M)x = N^{-1}b = \tilde{b}$$

$$A^{-1}b = (1 + N^{-1}M)^{-1}\tilde{b} = \tilde{b} + (-N^{-1}M)\tilde{b} + (-N^{-1}M)^2\tilde{b} + (-N^{-1}M)^3\tilde{b} + \cdots$$

$$\text{(step 0) } \tilde{b} = N^{-1}b, x = \tilde{b}$$

$$\text{(step 1) } r = \tilde{b} - (1 + N^{-1}M)x$$

$$\text{(step 2) } x = x + r = \tilde{b} - N^{-1}Mx$$

$$\text{(step 3) if } |r| \text{ is sufficiently small then exit else goto step 1.}$$

  - For a good convergence, we need to find a decomposition for *A=N+M* with good properties.

$$\boxed{N \ : \ \text{easy to invert and } |N^{-1}M| < 1}$$

- 2-1 Stationary iterative solvers
- Here I will show two common versions for the splitting.
  - Jacobi iteration (Diagonal scaling)

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & \ddots & \\ \vdots & \vdots & & \end{pmatrix}=\begin{pmatrix} a_{11} & 0 & 0 & \cdots \\ 0 & a_{22} & 0 & \cdots \\ 0 & 0 & \ddots & \\ \vdots & \vdots & & \end{pmatrix}+\begin{pmatrix} 0 & a_{12} & a_{13} & \cdots \\ a_{21} & 0 & a_{23} & \cdots \\ a_{31} & a_{32} & \ddots & \\ \vdots & \vdots & & \end{pmatrix}=N+M$$

$$N^{-1}A=\begin{pmatrix} 1 & a_{11}^{-1}a_{12} & a_{11}^{-1}a_{13} & \cdots \\ a_{22}^{-1}a_{21} & 1 & a_{22}^{-1}a_{23} & \cdots \\ a_{33}^{-1}a_{31} & a_{33}^{-1}a_{32} & \ddots & \\ \vdots & \vdots & & \end{pmatrix}$$

  - The diagonal part of the matrix is scaled to be 1.
  - For the Wilson fermions this scaling is already incorporated.
    - The diagonal term is the mass term. The mass term is scaled to be 1 and the hopping parameter is introduced.

$$D_f[U](n,m)=\left[1-\kappa_f H_{hop}\right](n,m) \qquad \kappa_f=\frac{1}{2(am_f+4)}$$

    - For improved Wilson actions there are non identity matrix in the site diagonal part. The Jacobi scaling is applicable in this case.
    - The effect of the improvement is limited, otherwise the diagonal part is dominant.

- 2-1 Stationary iterative solvers

– Gauss-Seidel iteration (Triangular spllittng)

$$N^{-1}A = \text{difficult to write down}$$

$$\text{full components....}$$

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & \ddots & \\ \vdots & \vdots & & \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ 0 & a_{22} & a_{22} & \cdots \\ 0 & 0 & \ddots & \\ \vdots & \vdots & & \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & \cdots \\ a_{21} & 0 & 0 & \cdots \\ a_{31} & a_{32} & \ddots & \\ \vdots & \vdots & & \end{pmatrix} = N + M$$

- The matrix can be decomposed to upper(lower) triangular matrix and strictly lower(upper) triangular matrix.
- The inversion for upper(lower) triangular matrix is easily done using the backward(forward) substitution.

$$\sum_{j=1}^{N} U_{ij} x_j = b_i, \qquad U_{ij} = \begin{cases} \neq 0 & \text{for } (i \le j) \\ = 0 & \text{for } (i > j) \end{cases}, \quad \text{as } N = U$$

$$x = U^{-1} b \Leftarrow x_i = b_i - \frac{1}{u_{ii}} \left(\sum_{j=N}^{i+1} u_{ij} x_j \right) \qquad \text{for } i = N, N-1, \cdots, 2, 1$$

Backward substitution

- For Wilson fermions, 4D lattice sites are indexed and ordered to be 1dim. Thus the upper/lowere decomposition depends on the site numbering/ordering scheme. The performance also depends on the ordering.

General Matrix decomposition /computation see also: R.Barret et al,"Templates for the Solution of Linear Systems: Building blocks for Iterative Methods"; Y.Saad "Iterative Methods for Sparse Linear Systems";
Details on the ordering and performance for LQCD: see Y.Oyanagi Comput.Phys.Commun. 42(1986)333; S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.

- ## 2-2 Adaptive (Kyrlov subspace) Solvers
  - Stationary Iterative solvers are not sufficient for High-performance computing (LQCD) and usually it is combined with the Krylov subspace methods.
  - In the Krylov subspace methods, the iterative solvers are sometimes used as the preconditioner of the target system equations.
- Kyrlov Subspace Solvers
  - For any lattice discretised difference equations, The induced linear equations are usually expressed in the finite but extremely large order linear equations.
  - As the dimension is finite we know that the characteristic polynomial is digree *N*.

$$\det[A-\lambda] = p(\lambda) = \lambda^N + c_{N-1}\lambda^{N-1} + c_{N-2}\lambda^{N-2} + \cdots + c_1\lambda + c_0$$

$$A: \quad N \times N \,\mathrm{matrix}$$

  - From the Cayley-Hamilton theorem we have

$$p(A) = A^N + c_{N-1}A^{N-1} + c_{N-2}A^{N-2} + \cdots + c_1 A + c_0 = 0$$

  - This means A^k with $k \geqq N$ can be reduced to a degree $j \leqq N\text{-}1$ polynomial.

- 2-2 Adaptive (Kyrlov subspace) Solvers
- Kyrlov Subspace Solvers
  - Thus for any analytic function *f(x)* we can express the corresponding matrix function as a *N-1* degree polynomial.

The spectrum of *A* should be inside of the convergence radii for this expression. Other expressions are possible for matrix functions.

$$f(A) = g_{N-1}A^{N-1} + g_{N-2}A^{N-2} + \cdots + g_1 A + g_0$$

  - For a matrix inverse we also have

$$A^{-1} = g_{N-1}A^{N-1} + g_{N-2}A^{N-2} + \cdots + g_1 A + g_0$$

det[A]/=0

  - The full construction is impossible for extremely large *N*. We only need an approximation for $x = A^{-1}b$.
  - This corresponds to a optimization problem to minimize the residual

$$r = b - Ax = b - A\left[d_{M-1}A^{M-1}b + d_{M-2}A^{M-2}b + \cdots + d_1 Ab + d_0 b\right]$$

  - By tuning the coeffcients {g_j} for $\boxed{M << N}$.

- 2-2 Adaptive (Kyrlov subspace) Solvers
- Kyrlov Subspace Solvers
  - The Krylov subspace solver is a solver that finds these coefficients adaptively and iteratively within the Krylov subspace defined by

$$Span\{b, Ab, A^2 b, \cdots, A^{M-2} b, A^{M-1} b\} \equiv \mathrm{K}^M (A; b)$$

  - The typical algorithm flow is

(Step 0) Given initial solution $x = x^{(0)}$

(Step 1) Compute initail residual vector $r = b - Ax$

(Step 2) Iterative Loop Start :

Depends on optimization strategy

...

(Step X) Compute a coefficient α and a vector v from previous iteration data/history using linear algebra (inner - product, vecto add.....) to minimize the following residual.

(Step X+1) Update (minimize locally) the residual $r = r + \alpha\ Av$

(Step X+2) Update the solution $x = x - \alpha\ v$

Independent of optimization strategy

....

(Step Z) Goto step 2 untill $|r|$ is sufficiently small.

for a given $r = b - Ax$, the updates

$r' = r + \alpha\ Av$

$x' = x - \alpha v$

does not change $r' = b - Ax'$

- 2-2 Adaptive (Kyrlov subspace) Solvers

Conjugate Gradient (CG) Algorithm

- CG : one of the most common algorithm.
- This can solve $Ax = b$ for a Hermitian Positive matrix A.

- For a Lattice quark opeator *D[U]*, CG is applied to

$$D\phi = \eta \Rightarrow \underline{D^{\dagger} D\phi = D^{\dagger}\eta}$$

$$\Downarrow$$

$$\boxed{\begin{gathered} Ax = b \\ A \equiv D^{\dagger} D, x = \phi, b \equiv D^{\dagger}\eta \end{gathered}}$$

- Squaring is needed since *D* is not Hermitian.
- (This is called Normal equation.)

```
$x = x^{(0)}; r = b - Ax; p = r$
$\rho_0 = \langle r | r \rangle$
do
    $q = Ap$
    $\alpha = \rho_0 / \langle p | q \rangle$
    $x = x + \alpha p$
    $r = r - \alpha q$
    $\rho_1 = \langle r | r \rangle$
    if $|r| = \sqrt{\rho_1}$ is sufficiently small exit do loop
    $\beta = \rho_1 / \rho_0 ; \rho_0 = \rho_1$
    $p = r + \beta p$
end do
```

●Greek characters are scalar variables.
●Roman characters are vectors.
●Capital Romans are matrixes.
This is a typical convention for applied mathematics for matrix computations.

A CG example

- Solve 1D Hermholz equation $\left(-\Delta+\kappa^2\right)\phi=\rho$ with CG.
- Naïve Discretization (N-sites)

$$-\phi(i+1)+(2+\kappa^2)\phi(i)-\phi(i-1)=\rho(i) \quad \text{for } i=1,\cdots,N$$

$$\phi(N+1)=\phi(1),\phi(0)=\phi(N) \qquad \text{(periodic b.c.)}$$

- In Matrix form

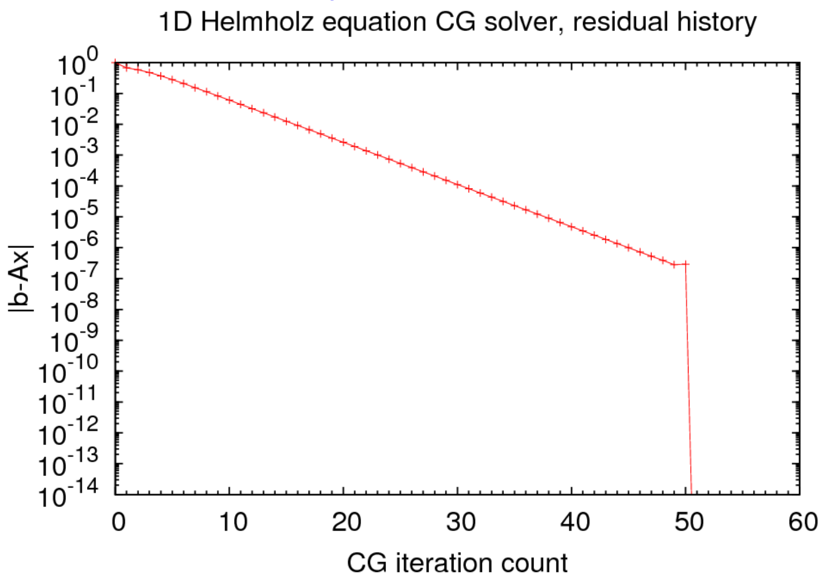
Lattice spacing *a* is absorbed in the normalization of rho, kappa, and number of sites.

$$f=2+\kappa^2 \quad \begin{pmatrix} f & -1 & 0 & \cdots & 0 & -1 \\ -1 & f & -1 & \ddots & & 0 \\ 0 & -1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & f & -1 \\ -1 & 0 & \cdots & 0 & -1 & f \end{pmatrix} \begin{pmatrix} \phi(1) \\ \phi(2) \\ \vdots \\ \vdots \\ \phi(N-1) \\ \phi(N) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \vdots \\ \rho(N-1) \\ \rho(N) \end{pmatrix}$$

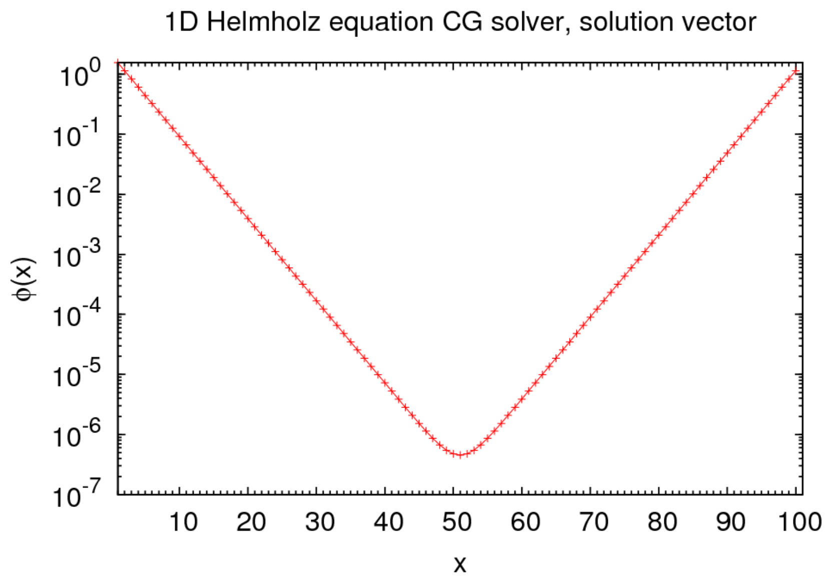
- A Fortran program example:
  [http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/ASLFT2010/Helmholz1DCG.tar.gz]

- 2-2 Adaptive (Kyrlov subspace) Solvers
  - Solve 1D Hermholz equation $\left(-\Delta+\kappa^2\right)\phi=\rho$ with CG.
  - A Fortran program example:
    [http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/ASLFT2010/Helmholz1DCG.tar.gz]
  - Number of sites *N*=100, $\kappa^2=0.1$

Residual History



Solution vector



- Exponential convergence is observed.
- Theoretically 100 iteration is sufficient to solve the equation.
- In this case CG algorithm shows a lucky convergence at 50the iteration. This is because of small matrix dimension. For LQCD this phenomena rarely happen.

- Conjugate Gradient (CG) Algorithm
  - The convergence speed is governed by the spectrum of the coefficient matrix *A*.
  - The error bound for the solution has been analytically obtained as

$$\left\| x^{*} - x^{(m)} \right\|_A \leq 2\left[\frac{\sqrt{K}-1}{\sqrt{K}+1}\right]^{m} \left\| x^{*} - x^{(0)} \right\|_A$$

$$\left\| v \right\|_A \equiv \sqrt{\langle v \,|\, A \,|\, v \rangle}$$

$$\boxed{K = K(A) \equiv \left\| A \right\| \left\| A^{-1} \right\|}$$

Condition number of matrix *A*

x^{*}	True solution
$x^{(0)}$	Initial guess
$x^{(m)}$	CG guess at *m*th iter.

  - When *A* has a bad condition number CG sometimes stagnates.
  - In LQCD with the normal equation, *D* is squared as $A = D^{\dagger}D$ and *A* has a bad condition. Preconditioning or other solvers applicable to non-Hermitian matrix are required. $\kappa(D^{\dagger}D) \approx \left|\kappa(D)\right|^{2}$

Bi-Conjugate Gradient Stabilized (BiCGStab) Algorithm

- As the Wilson Dirac operator is non-Hermitian, it is desired to directly solve linear equations with a non-Hermitian/unsymmetric coefficient matrix.
- Bi-Conjugate Gradient Stabilized (BiCGStab) algorithm has been known as a best solver for Wilson Dirac quarks. [BiCGStab: van der Volst (1992)]
- The algorithm flow is very similar to that of CG, but we need two the matrix vector multiplication in a iteration.
- The algorithm is:

For LQCD Wilson quarks, introduced by [Frommer et al. Int.J.Mod.Phys. C5 (1994) 1073]

```
$x = x^{(0)}; r = b - Ax; p = r;$
Shadow vetctor $\breve{r}$ can be arbitrary.
A simple choice is set $\breve{r} = r$.
$\rho_0 = \langle \breve{r} | r \rangle$
do
    $q = Ap$
    $\alpha = \rho_0 / \langle \breve{r} | q \rangle$
    $x = x + \alpha p$
    $r = r - \alpha q$
    if $|r|$ is sufficiently small exit do loop
    $t = Ar$
    $\omega = \langle t | r \rangle / \langle t | t \rangle$
    $x = x + \omega r$
    $r = r - \omega t$
    if $|r|$ is sufficiently small exit do loop
    $\rho_1 = \langle \breve{r} | r \rangle$
    $\beta = \left( \alpha / \omega \right) \left( \rho_1 / \rho_0 \right); \rho_0 = \rho_1$
    $p = r + \beta (p - \omega q)$
end do
```

- Bi-Conjugate Gradient Stabilized (BiCGStab) Algorithm
  - BiCGStab stagnates when real negative eigenvalues exist in *A*.
- A BiCGStab/CGNE Example: 1D Wilson like operator

$$(D\phi)(i) = \phi(i) - \kappa\left[(1-\sigma_3)\phi(i+1) + (1+\sigma_3)\phi(i-1)\right] = \rho(i)$$

$$\phi(N+1) = \phi(1), \phi(0) = \phi(N) \qquad \text{(periodic b.c.)}$$

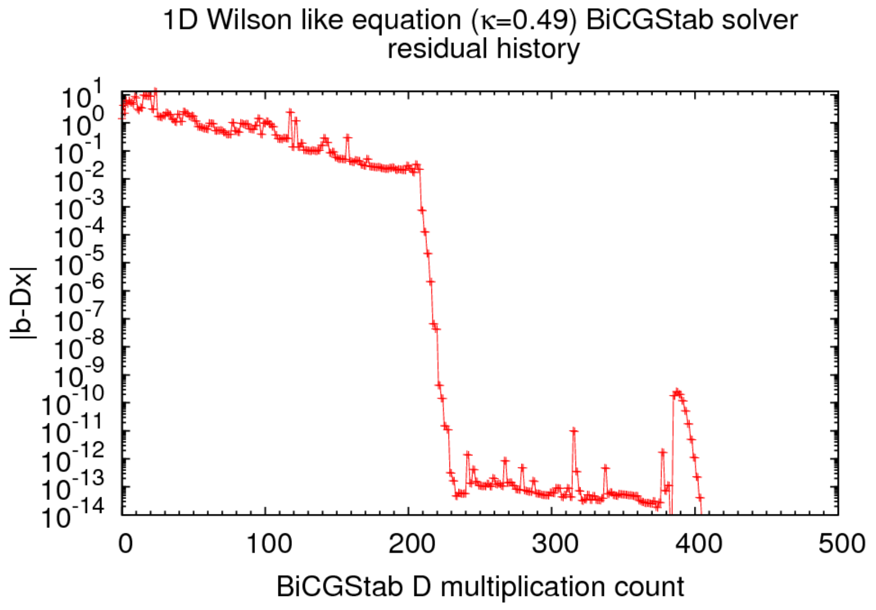
$$\phi(i) \ : \text{2-component spinor}, \quad \sigma_3 : \ \text{3rd-Pauli matrix.}$$

  - Matrix Form $\begin{pmatrix} E & -F & 0 & \cdots & 0 & -B \\ -B & E & -F & \ddots & & 0 \\ 0 & -B & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & E & -F \\ -F & 0 & \cdots & 0 & -B & E \end{pmatrix} \begin{pmatrix} \phi(1) \\ \phi(2) \\ \vdots \\ \vdots \\ \phi(N-1) \\ \phi(N) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \vdots \\ \rho(N-1) \\ \rho(N) \end{pmatrix}$
  - *D* is asymmetric.
  - CG is not applicable for *Dx=b*.
  - A Fortran program example:
    [http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/ASLFT2010/Wilson1DCG.tar.gz]

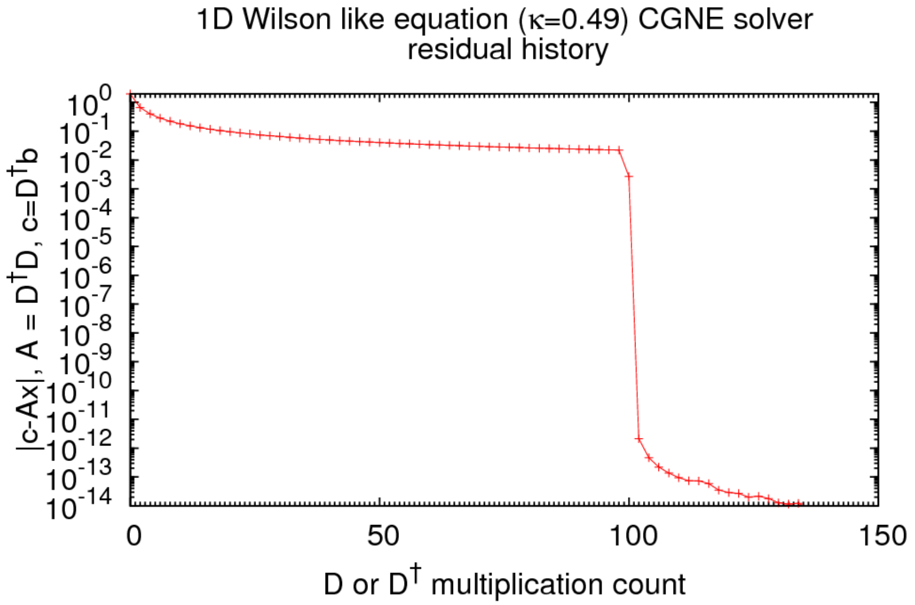
$$E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ F = \begin{pmatrix} 0 & 0 \\ 0 & 2\kappa \end{pmatrix}, \ B = \begin{pmatrix} 2\kappa & 0 \\ 0 & 0 \end{pmatrix}$$

- Bi-Conjugate Gradient Stabilized (BiCGStab) Algorithm
- A BiCGStab/CGNE Example: 1D Wilson like operator (Nsite = 100, K=0.49)
  - Residual history

BiCGStab with random shadow residual.



CG for NE (CGNE)



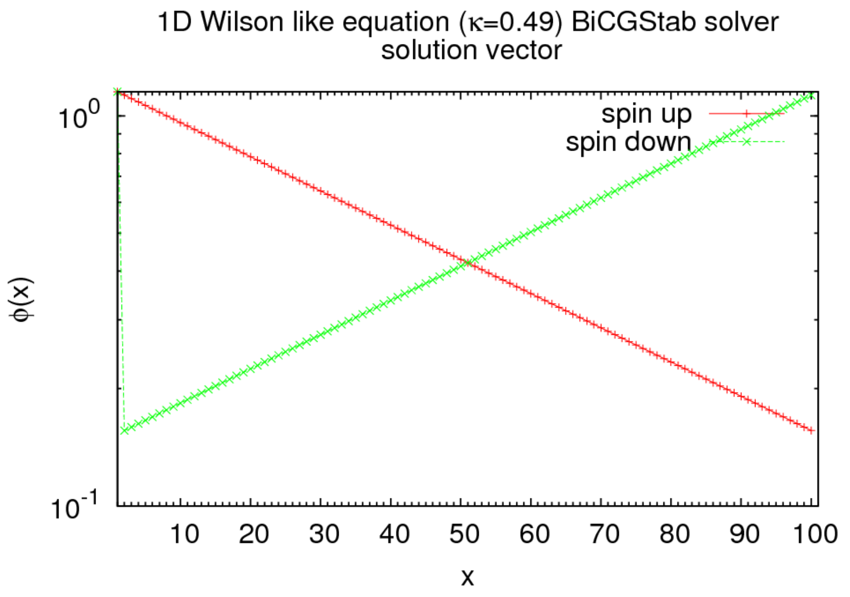
●BiCGStab shows a fluctuating behavior. This behavior is typical for BiCGStab with complex eigenvalues. Due to this behavior one need to maintain the descrepancy between accumulated residual and true residual in the iteration. A reliable update method exists.

●CGNE shows a smooth convergence.

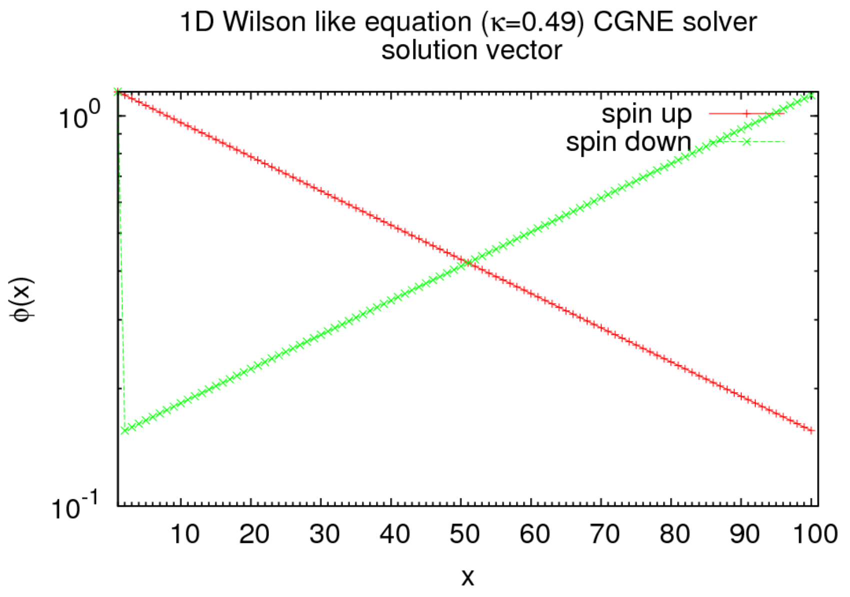
●In this case CGNE is economical than BiCGStab in computational cost.

- Bi-Conjugate Gradient Stabilized (BiCGStab) Algorithm
- A BiCGStab/CGNE Example: 1D Wilson like operator (Nsite = 100, K=0.49)
  - Solution vector

BiCGStab



CG for NE (CGNE)



Both methods converges to a numerically identical solution.
(How about the consistency to the theoretical solution?)

3. Preconditioning

- The Kyrlov subspace methods alone sometimes does not show satisfactory convergence when the condition number of the coefficient matrix is large.

$$Ax = b \qquad \boxed{K(A) \equiv \|A\| \|A^{-1}\| >> 1}$$

- The target equation can be modified to identical equation with a coeffcieint matrix with small condition number. This is done by applying a constant matrix to the equation from left hand or inserting a constant matrix and its inverse in the right hand side of coefficient matrix. If the modified coefficient matrix has a smaller condition number, the Krylov subspace methods show better performance.

$$Ax = b \Rightarrow \begin{cases} (MA)x = Mb & \boxed{K(MA) < K(A)} \\ (AM)z = b, x = Mz & \boxed{K(AM) < K(A)} \end{cases}$$

Left preconditioning

Right preconditioning

- This modification is called "Preconditioning".
- The matrix *M* is called "Preconditioner"

Left-Right preconditioning is also possible.

3. Preconditioning

$$Ax=b \Rightarrow \begin{cases} (MA)x = Mb \\ (AM)z = b, x = Mz \end{cases} \quad \begin{matrix} \boxed{K(MA) < K(A)} \\ \boxed{K(AM) < K(A)} \end{matrix}$$

Left preconditioning

Right preconditioning

Left-Right preconditioning is also possible.

- Preconditioning is effective when the following criterion are satisfied.
  (1) The preconditioned coefficient matrix has a smaller condition number.
  (2) The computational cost of multiplying "*M*" is sufficiently small.

  – The criterion (1) can be satisfied by choosing $M \approx A^{-1}$. However this is the original problem. We have to find an approximation with less computational cost (2) for $M \approx A^{-1}$.

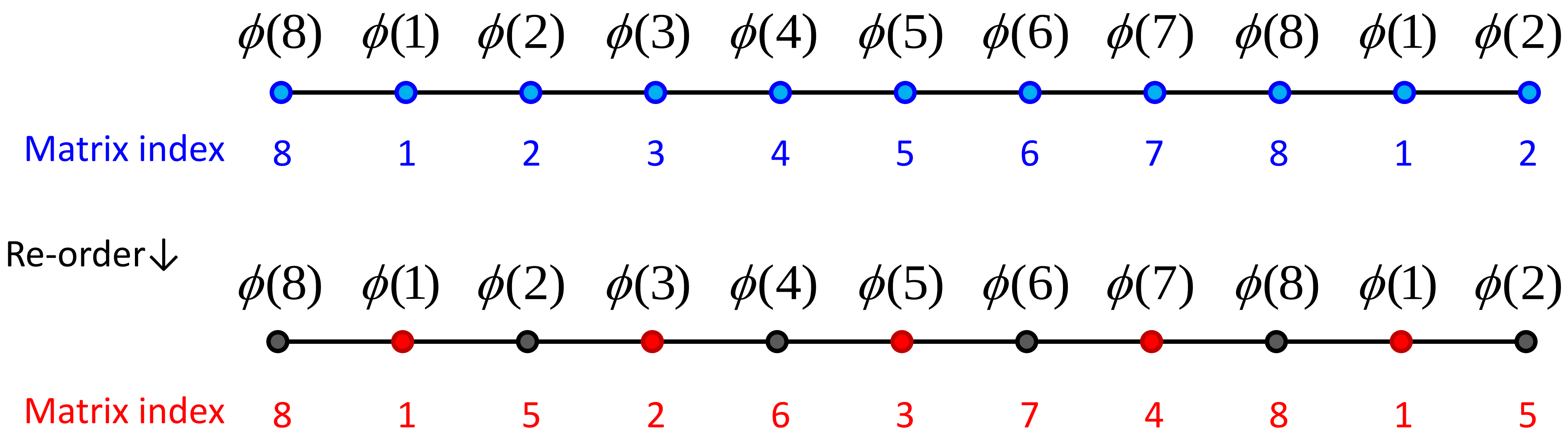
  – Stationary iterative solver is applicable as the preconditioner. This combination, Krylov solver + Stationary solver, has been widely used.

  – For LQCD Wilson type quarks, several effective preconditiners have been known. For Ovelap/Domainwall fermions it is still less known.

- Here I show two examples for the preconditioners.
  - (1) Even/Odd site (Red-Black) preconditioning.
  - (2) Gauss-Seidel/SSOR preconditioning.

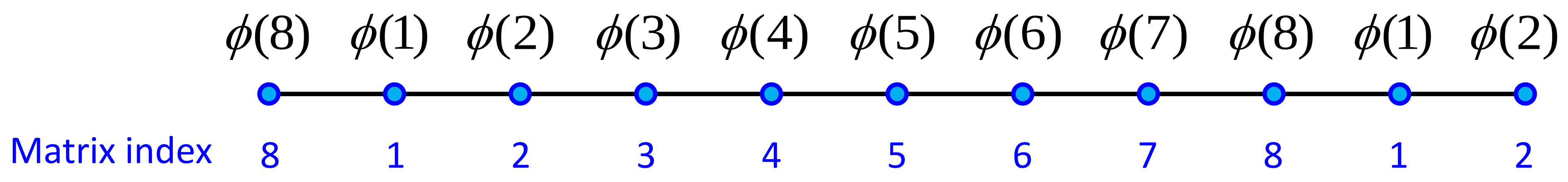
(1) Even/Odd site (Red-Black) preconditioning.

- When the number of sites is a even number we can apply this preconditioning to the Wilson type (single hopping) matrix easily.
- For example we apply this to 1D Wilson like operator. (Nsite=8 case)


$\phi(8)$ $\phi(1)$ $\phi(2)$ $\phi(3)$ $\phi(4)$ $\phi(5)$ $\phi(6)$ $\phi(7)$ $\phi(8)$ $\phi(1)$ $\phi(2)$
Matrix index 8 1 2 3 4 5 6 7 8 1 2
Re-order↓
$\phi(8)$ $\phi(1)$ $\phi(2)$ $\phi(3)$ $\phi(4)$ $\phi(5)$ $\phi(6)$ $\phi(7)$ $\phi(8)$ $\phi(1)$ $\phi(2)$
Matrix index 8 1 5 2 6 3 7 4 8 1 5

(1) Even/Odd site (Red-Black) preconditioning.

- For example we apply this to 1D Wilson like operator. (Nsite=8 case)


$\phi(8)$ $\phi(1)$ $\phi(2)$ $\phi(3)$ $\phi(4)$ $\phi(5)$ $\phi(6)$ $\phi(7)$ $\phi(8)$ $\phi(1)$ $\phi(2)$
Matrix index
8 1 2 3 4 5 6 7 8 1 2

– In Matrix Form

Normal ordering

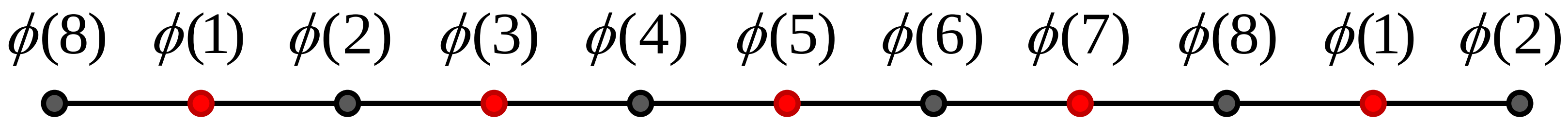
$$\begin{pmatrix} E & -F & & & & & & -B \\ -B & E & -F & & & & & \\ & -B & E & -F & & & & \\ & & -B & E & -F & & & \\ & & & -B & E & -F & & \\ & & & & -B & E & -F & \\ & & & & & -B & E & -F \\ -F & & & & & & -B & E \end{pmatrix} \begin{pmatrix} \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(4) \\ \phi(5) \\ \phi(6) \\ \phi(7) \\ \phi(8) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \rho(3) \\ \rho(4) \\ \rho(5) \\ \rho(6) \\ \rho(7) \\ \rho(8) \end{pmatrix}$$

– Is reordered to

$$E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad F = \begin{pmatrix} 0 & 0 \\ 0 & 2\kappa \end{pmatrix}, \quad B = \begin{pmatrix} 2\kappa & 0 \\ 0 & 0 \end{pmatrix}$$

(1) Even/Odd site (Red-Black) preconditioning.

- For example we apply this to 1D Wilson like operator. (Nsite=8 case)



Matrix index	8	1	5	2	6	3	7	4	8	1	5

– In Matrix Form

Red-Black Ordering (even/odd ordering)

$$\left(\begin{array}{cccc|cccc} E & & & & -F & & & -B \\ & E & & & -B & -F & & \\ & & E & & & -B & -F & \\ & & & E & & & -B & -F \\ \hline -B & -F & & & E & & & \\ & -B & -F & & & E & & \\ & & -B & -F & & & E & \\ -F & & & -B & & & & E \end{array}\right)\begin{pmatrix} \phi(1) \\ \phi(3) \\ \phi(5) \\ \phi(7) \\ \phi(2) \\ \phi(4) \\ \phi(6) \\ \phi(8) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(3) \\ \rho(5) \\ \rho(7) \\ \rho(2) \\ \rho(4) \\ \rho(6) \\ \rho(8) \end{pmatrix}$$

– We can decouple the unknowns on even(black) sites.

$$E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ F = \begin{pmatrix} 0 & 0 \\ 0 & 2\kappa \end{pmatrix},\ B = \begin{pmatrix} 2\kappa & 0 \\ 0 & 0 \end{pmatrix}$$

(1) Even/Odd site (Red-Black) preconditioning.

- For example we apply this to 1D Wilson like operator. (Nsite=8 case)
  - We can decouple the unknowns on even(black) sites.
  - In 2x2 Block matrix form

$$\begin{pmatrix} D_{RR} & D_{RB} \\ D_{BR} & D_{BB} \end{pmatrix}\begin{pmatrix} \phi_R \\ \phi_B \end{pmatrix} = \begin{pmatrix} \rho_R \\ \rho_B \end{pmatrix}, \qquad D_{RR} = D_{BB} = diag(1,1,1,1) = 1$$

  - Left preconditioning

$$\begin{pmatrix} 1 & -D_{RB} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & D_{RB} \\ D_{BR} & 1 \end{pmatrix}\begin{pmatrix} \phi_R \\ \phi_B \end{pmatrix} = \begin{pmatrix} 1 & -D_{RB} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \rho_R \\ \rho_B \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 1 - D_{RB}D_{BR} & 0 \\ D_{BR} & 1 \end{pmatrix}\begin{pmatrix} \phi_R \\ \phi_B \end{pmatrix} = \begin{pmatrix} \rho_R - D_{RB}\rho_B \\ \rho_B \end{pmatrix}$$

  - We have to solve
  - for ϕ_R

$$\boxed{(1 - D_{RB}D_{BR})\phi_R = \rho_R - D_{RB}\rho_B}$$

$$\phi_B = \rho_B - D_{BR}\phi_R$$

3. Preconditioning

(1) Even/Odd site (Red-Black) preconditioning.

- 1D Wilson like operator.
  - In this case The coefficient matrix is modified from D to

$$\hat{D}_{RR} \equiv \left(1 - D_{RB} D_{BR}\right)$$

  - If kappa is sufficiently small $D = 1 + O(\kappa)$

$$\hat{D}_{RR} \equiv 1 - D_{RB} D_{BR} = 1 + O(\kappa^2)$$

  - The preconditioned matrix $\hat{D}_{RR}$ is more close to identity matrix. Thus we expect $K(\hat{D}_{RR}) < K(D)$ and solving

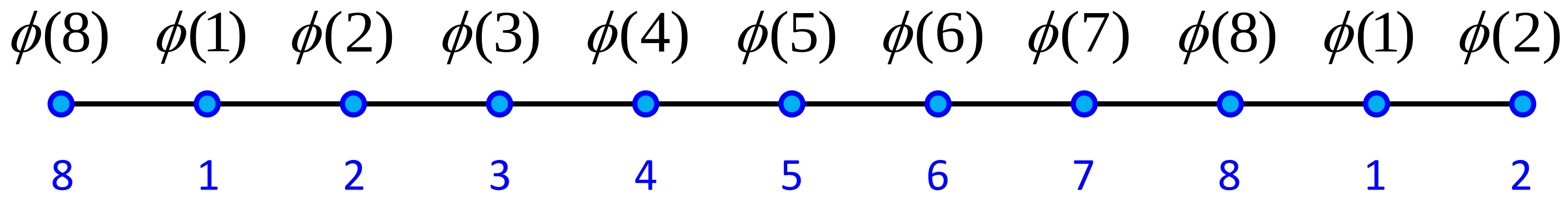
$$\hat{D}_{RR}\phi_R = \hat{\rho}_R, \qquad \hat{\rho}_R \equiv \rho_R - D_{RB}\rho_B$$

  - is more easier than solving the original eq. $D\phi = \rho$

– This technique is also applicable to the 4D Wilson-Dirac type operators and have been used widely.(4D site even/odd indexing is required.)

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- For example we apply this to 1D Wilson like operator. (Nsite=8 case)
- Without changing we can precondition *D* as follows.


$\phi(8)$ $\phi(1)$ $\phi(2)$ $\phi(3)$ $\phi(4)$ $\phi(5)$ $\phi(6)$ $\phi(7)$ $\phi(8)$ $\phi(1)$ $\phi(2)$
Matrix index
8 1 2 3 4 5 6 7 8 1 2

- In Matrix Form

Normal ordering

$$\begin{pmatrix} E & -F & & & & & & -B \\ -B & E & -F & & & & & \\ & -B & E & -F & & & & \\ & & -B & E & -F & & & \\ & & & -B & E & -F & & \\ & & & & -B & E & -F & \\ & & & & & -B & E & -F \\ -F & & & & & & -B & E \end{pmatrix} \begin{pmatrix} \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(4) \\ \phi(5) \\ \phi(6) \\ \phi(7) \\ \phi(8) \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \rho(3) \\ \rho(4) \\ \rho(5) \\ \rho(6) \\ \rho(7) \\ \rho(8) \end{pmatrix}$$

- *D* can be separated to Upper and Lower matrix. $E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $F = \begin{pmatrix} 0 & 0 \\ 0 & 2\kappa \end{pmatrix}$, $B = \begin{pmatrix} 2\kappa & 0 \\ 0 & 0 \end{pmatrix}$

(2) Gauss-Seidel/SOR/SSOR preconditioning.

$$E=\begin{pmatrix}1&0\\0&1\end{pmatrix},\ F=\begin{pmatrix}0&0\\0&2\kappa\end{pmatrix},\ B=\begin{pmatrix}2\kappa&0\\0&0\end{pmatrix}$$

- 1D Wilson like operator. (Nsite=8 case)
  - *D* can be separated to a sum of a Upper and a Lower matrix.

$$\begin{pmatrix}E&-F&&&&&&-B\\-B&E&-F&&&&&\\&-B&E&-F&&&&\\&&-B&E&-F&&&\\&&&-B&E&-F&&\\&&&&-B&E&-F&\\&&&&&-B&E&-F\\-F&&&&&&-B&E\end{pmatrix}=\begin{pmatrix}E&&&&&&&\\-B&E&&&&&&\\&-B&E&&&&&\\&&-B&E&&&&\\&&&-B&E&&&\\&&&&-B&E&&\\&&&&&-B&E&\\-F&&&&&&-B&E\end{pmatrix}+\begin{pmatrix}E&-F&&&&&&-B\\&E&-F&&&&&\\&&E&-F&&&&\\&&&E&-F&&&\\&&&&E&-F&&\\&&&&&E&-F&\\&&&&&&E&-F\\&&&&&&&E\end{pmatrix}$$

$$-diag(E,E,E,E,E,E,E,E)$$

$$D=L+U-1$$

  - The triangular matrixes can be inverted easily by using forward/backward substitution.
  - We can consider the following preconditioning

$$L^{-1}D=1+L^{-1}(U-1),\quad U^{-1}D=1+U^{-1}(L-1)$$ Left prec'd by inv.of *L* or *U*

$$DL^{-1}=1+(U-1)L^{-1},\quad DU^{-1}=1+(L-1)U^{-1}$$ Right prec'd by inv.of *L* or *U*

$$L^{-1}DU^{-1}=U^{-1}+L^{-1}(1-U^{-1}),\quad U^{-1}DL^{-1}=L^{-1}+U^{-1}(1-L^{-1})$$ LR prec'd by inv.of *L* & *U*

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
  - *D* can be separated to a sum of a Upper and a Lower matrix.

$$D = L + U - 1$$

  - This corresponds to a single iteration of Gauss-Seidel iteration.
  - We expect that these

$$L^{-1}D = 1 + L^{-1}(U-1), \quad U^{-1}D = 1 + U^{-1}(L-1)$$
$$DL^{-1} = 1 + (U-1)L^{-1}, \quad DU^{-1} = 1 + (L-1)U^{-1}$$
$$L^{-1}DU^{-1} = U^{-1} + L^{-1}(1-U^{-1}), \quad U^{-1}DL^{-1} = L^{-1} + U^{-1}(1-L^{-1})$$

  - have a better condition number than that of original D.
  - A Krylov solver is applied to the following equations.

$\left[1 + L^{-1}(U-1)\right]x = L^{-1}b$	For Left prec'd by inv. of *L.*
$\left[1 + (U-1)L^{-1}\right]z = b, \quad x = L^{-1}z$	For Right prec'd by inv. of *L.*
$\left[U^{-1} + L^{-1}(1-U^{-1})\right]z = L^{-1}b, \quad x = L^{-1}z$	For LR prec'd by inv.of *L* & *U.*

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
- Note : Computational cost of preconditioned matrix is almost the same as that of original matrix. [Eisenstat's trick]

$$\underline{v = Lw} \Leftrightarrow v_i = w_i + \sum_{j=1}^{i-1} l_{ij} w_j \quad \text{for } i = 1,2,\cdots,N-1,N. \Leftrightarrow \boxed{N(N+1)\,\text{flop}[+,\times]}$$

$$\underline{v = L^{-1}v} \Leftrightarrow v_i = w_i - \sum_{j=1}^{i-1} l_{ij} v_j \quad \text{for } i = 1,2,\cdots,N-1,N. \Leftrightarrow \boxed{N(N+1)\,\text{flop}[+,\times]}$$

$$Dv = (L+U-1)v : \quad \text{Cost}[Dv] \approx \text{Cost}[Lv] + \text{Cost}[Uv]$$

$$L^{-1}Dv = (1+L^{-1}(U-1))v : \quad \text{Cost}[L^{-1}Dv] \approx \text{Cost}[L^{-1}v] + \text{Cost}[Uv] \approx \text{Cost}[Dv]$$

For Left prec'd version

$$L^{-1}DU^{-1}v = U^{-1}v + L^{-1}(1-U^{-1})v$$

$$\boxed{\begin{aligned} &\Rightarrow s = U^{-1}v \\ &L^{-1}DU^{-1}v = s + L^{-1}(v-s) \end{aligned}} : \quad \text{Cost}[L^{-1}DU^{-1}v] \approx \text{Cost}[L^{-1}v] + \text{Cost}[U^{-1}v] \approx \text{Cost}[Dv]$$

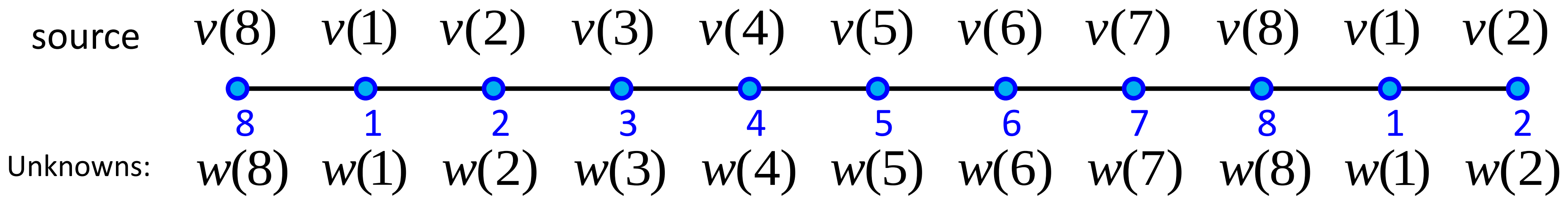
Eisenstat's trick

For LR prec'd version

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

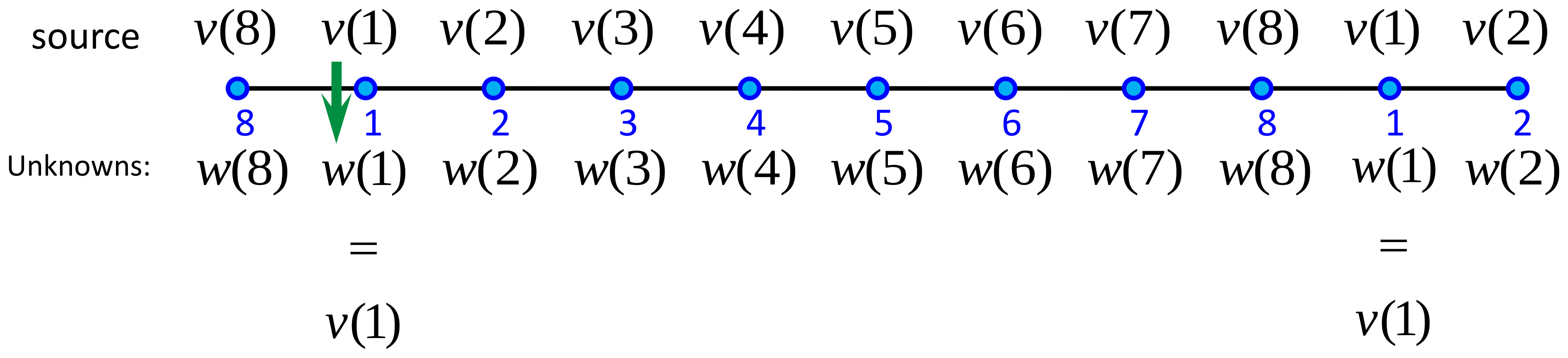
- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver: $Lw = v \rightarrow w = L^{-1} v$


source
$v(8)$ $v(1)$ $v(2)$ $v(3)$ $v(4)$ $v(5)$ $v(6)$ $v(7)$ $v(8)$ $v(1)$ $v(2)$
8 1 2 3 4 5 6 7 8 1 2
Unknowns:
$w(8)$ $w(1)$ $w(2)$ $w(3)$ $w(4)$ $w(5)$ $w(6)$ $w(7)$ $w(8)$ $w(1)$ $w(2)$

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver: $Lw = v \rightarrow w = L^{-1}v$

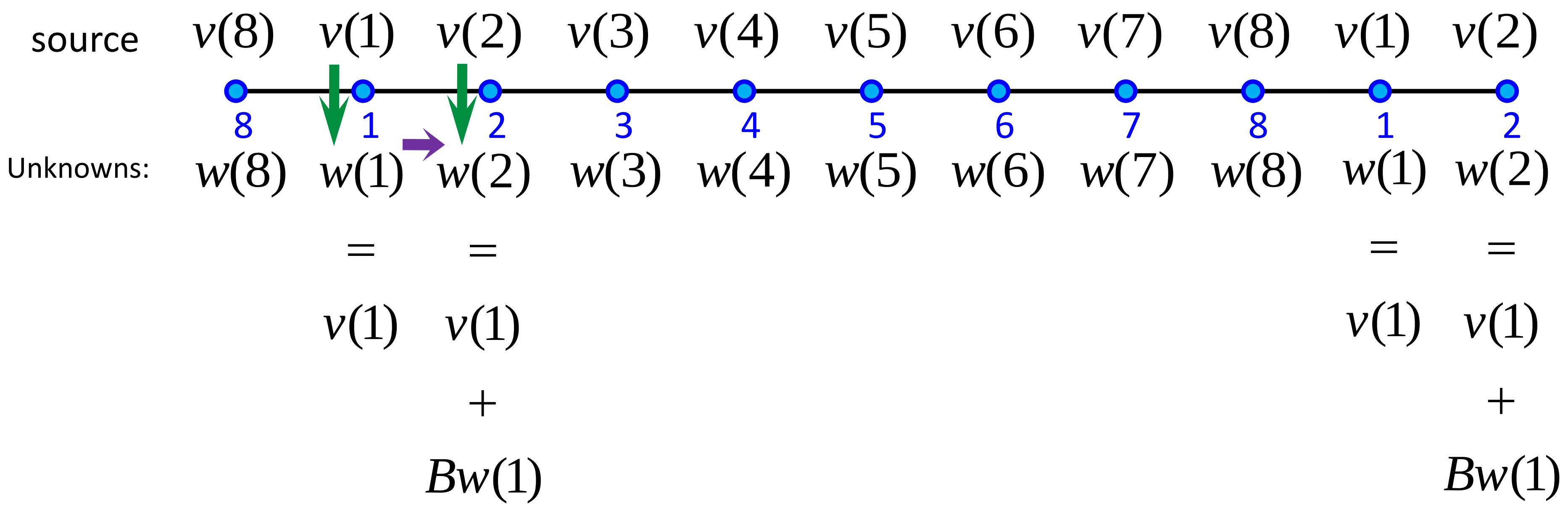

source
$v(8)$ $v(1)$ $v(2)$ $v(3)$ $v(4)$ $v(5)$ $v(6)$ $v(7)$ $v(8)$ $v(1)$ $v(2)$
8 1 2 3 4 5 6 7 8 1 2
Unknowns:
$w(8)$ $w(1)$ $w(2)$ $w(3)$ $w(4)$ $w(5)$ $w(6)$ $w(7)$ $w(8)$ $w(1)$ $w(2)$
$=$ $=$
$v(1)$ $v(1)$

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver:

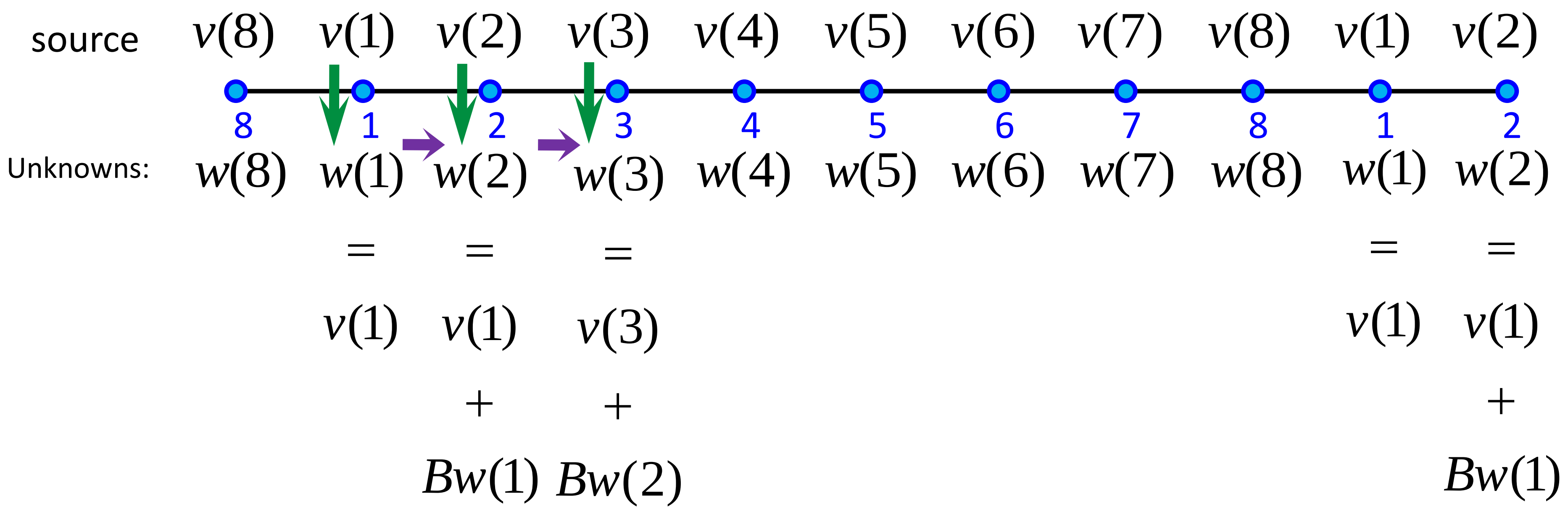
$$Lw = v \rightarrow w = L^{-1}v$$


source
$v(8)$ $v(1)$ $v(2)$ $v(3)$ $v(4)$ $v(5)$ $v(6)$ $v(7)$ $v(8)$ $v(1)$ $v(2)$
8 1 2 3 4 5 6 7 8 1 2
Unknowns:
$w(8)$ $w(1)$ $w(2)$ $w(3)$ $w(4)$ $w(5)$ $w(6)$ $w(7)$ $w(8)$ $w(1)$ $w(2)$
$w(1) = v(1)$
$w(2) = v(1) + Bw(1)$
$w(1) = v(1)$
$w(2) = v(1) + Bw(1)$

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

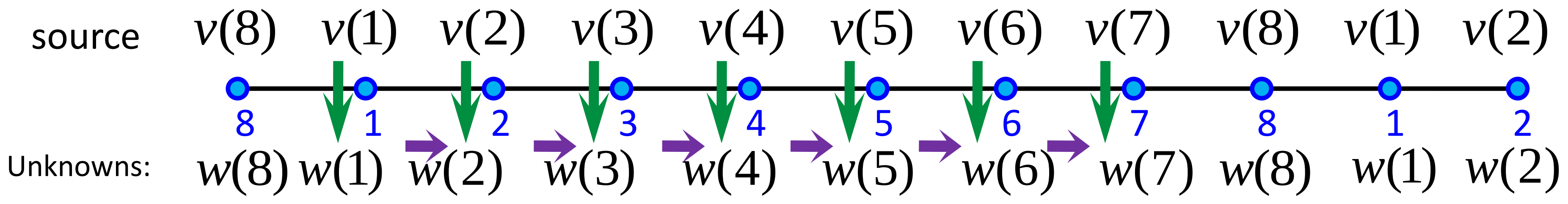
- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver: $Lw = v \rightarrow w = L^{-1}v$


source
$v(8)$ $v(1)$ $v(2)$ $v(3)$ $v(4)$ $v(5)$ $v(6)$ $v(7)$ $v(8)$ $v(1)$ $v(2)$
8 1 2 3 4 5 6 7 8 1 2
Unknowns:
$w(8)$ $w(1)$ $w(2)$ $w(3)$ $w(4)$ $w(5)$ $w(6)$ $w(7)$ $w(8)$ $w(1)$ $w(2)$
$w(1) = v(1)$
$w(2) = v(1) + Bw(1)$
$w(3) = v(3) + Bw(2)$
$w(1) = v(1)$
$w(2) = v(1) + Bw(1)$

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver: $Lw = v \rightarrow w = L^{-1}v$


source
$v(8)$ $v(1)$ $v(2)$ $v(3)$ $v(4)$ $v(5)$ $v(6)$ $v(7)$ $v(8)$ $v(1)$ $v(2)$
8 1 2 3 4 5 6 7 8 1 2
Unknowns:
$w(8)$ $w(1)$ $w(2)$ $w(3)$ $w(4)$ $w(5)$ $w(6)$ $w(7)$ $w(8)$ $w(1)$ $w(2)$

$w(1) = v(1)$

$w(2) = v(1) + Bw(1)$

$w(3) = v(3) + Bw(2)$

$w(4) = v(4) + Bw(3)$

$w(5) = v(5) + Bw(4)$

$w(6) = v(6) + Bw(5)$

$w(7) = v(7) + Bw(6)$

$w(1) = v(1)$

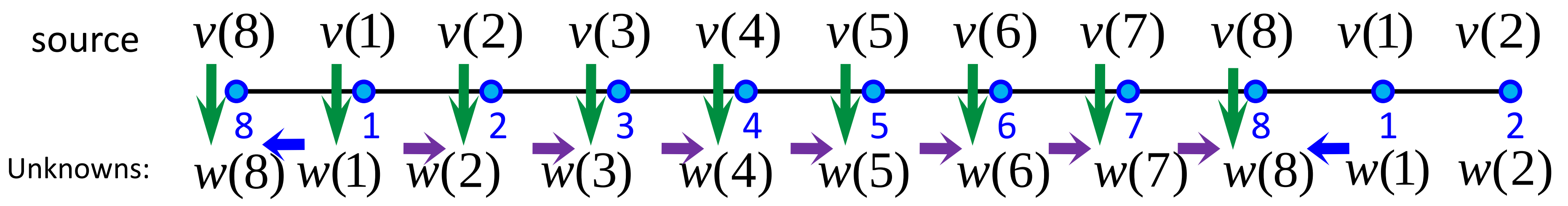
$w(2) = v(1) + Bw(1)$

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

(2) Gauss-Seidel/SOR/SSOR preconditioning.

- 1D Wilson like operator. (Nsite=8 case)
- Example of Forward solver:

$$Lw = v \rightarrow w = L^{-1}v$$



$$\begin{array}{ccccccccccc}
w(8) & w(1) & w(2) & w(3) & w(4) & w(5) & w(6) & w(7) & w(8) & w(1) & w(2) \\
= & = & = & = & = & = & = & = & = & = & = \\
v(8) & v(1) & v(1) & v(3) & v(4) & v(5) & v(6) & v(7) & v(8) & v(1) & v(1) \\
+ & & + & + & + & + & + & + & + & & + \\
Bw(7) & & Bw(1) & Bw(2) & Bw(3) & Bw(4) & Bw(5) & Bw(6) & Bw(7) & & Bw(1) \\
+ & & & & & & & & + & & \\
Fw(1) & & & & & & & & Fw(1) & &
\end{array}$$

- Single hopping structure simplifies the forward/backward substitution code programming. (Difficulties always exist in Boundary condition.)
  Consider Extension to 4D Wilson-Dirac operator.

3. F

(2) Gauss-Seidel/SOR/SSOR precor

[See also : M. Peardon arXiv:hep-lat/0011080;
S. Fischer et al. Comput.Phys.Commun. 98 (1996) 20.]

- The upper and lower part decomposition depends on the index ordering.
- The performance of the preconditioner also depends on the index ordering.
- The data dependency in the forward/backward substitution reflects an aspect of properties of preconditioner.
- It has been known that the high dependency in the resultant decomposition means the high performance of the preconditioner.
- Red/Balck(E/O) preconditioner is a special version of the Gauss-Seidel preconditioner.
- Here I have explained the Gauss-Seidel versions of the preconditioner.
- For successive over-relaxation (SOR)/sysmmetric successive over-relaxation (SSOR). An over-relaxation parameter is inserted in the matrix splitting.

$$
\begin{aligned}
D &= L + U - 1 = (L-1) + (U-1) + 1 \\
&= \frac{1}{\omega}\left(\omega(L-1) + \omega(U-1) + \omega\right) \\
&= \frac{1}{\omega}\left(\left[1+\omega(L-1)\right] + \left[1+\omega(U-1)\right] + \omega - 2\right)
\end{aligned}
$$

$$\Rightarrow \left[1+\omega(L-1)\right]^{-1} D \left[1+\omega(U-1)\right]^{-1}$$

SSOR preconditioned

ω OR parameter: 1<ω <2

4. Parallelization

- All example codes are written for a single process execution.
- For a realistic LQCD simulation, we need more computational power. Parallel computation using multiple process is needed.
- The most timing consuming part of LQCD is the quark solver. Thus at first parallelizing the quark solver is a good experience before parallelizing whole LQCD related programs.
- Solver example coeds can be parallelized in two manners.
  - Parallelization by MPI (Message Passing Interface API).
    - This is based on SPMD (Single program multiple data) model
  - Parallelization by OpenMP
    - This is directive based thread parallelization. This is usefull for Single CPU with Multi-cores to fully drive all cores.

- To test the parallel codes with MPI, we need parallel computer environment. If your Linux Desktop contains OpenMPI package, you can emulate the parallel environment (even if your PC has single CPU).

- To test OpenMP thread parallelization, A compiler that can understand the OpenMP directives is needed. Unfortunately the level of OpenMP compliance of GNU compiler is still low. The latest GNU complier is at almost satisfactory level, common Linux distributions do not contain this latest version.....

4. Parallelization

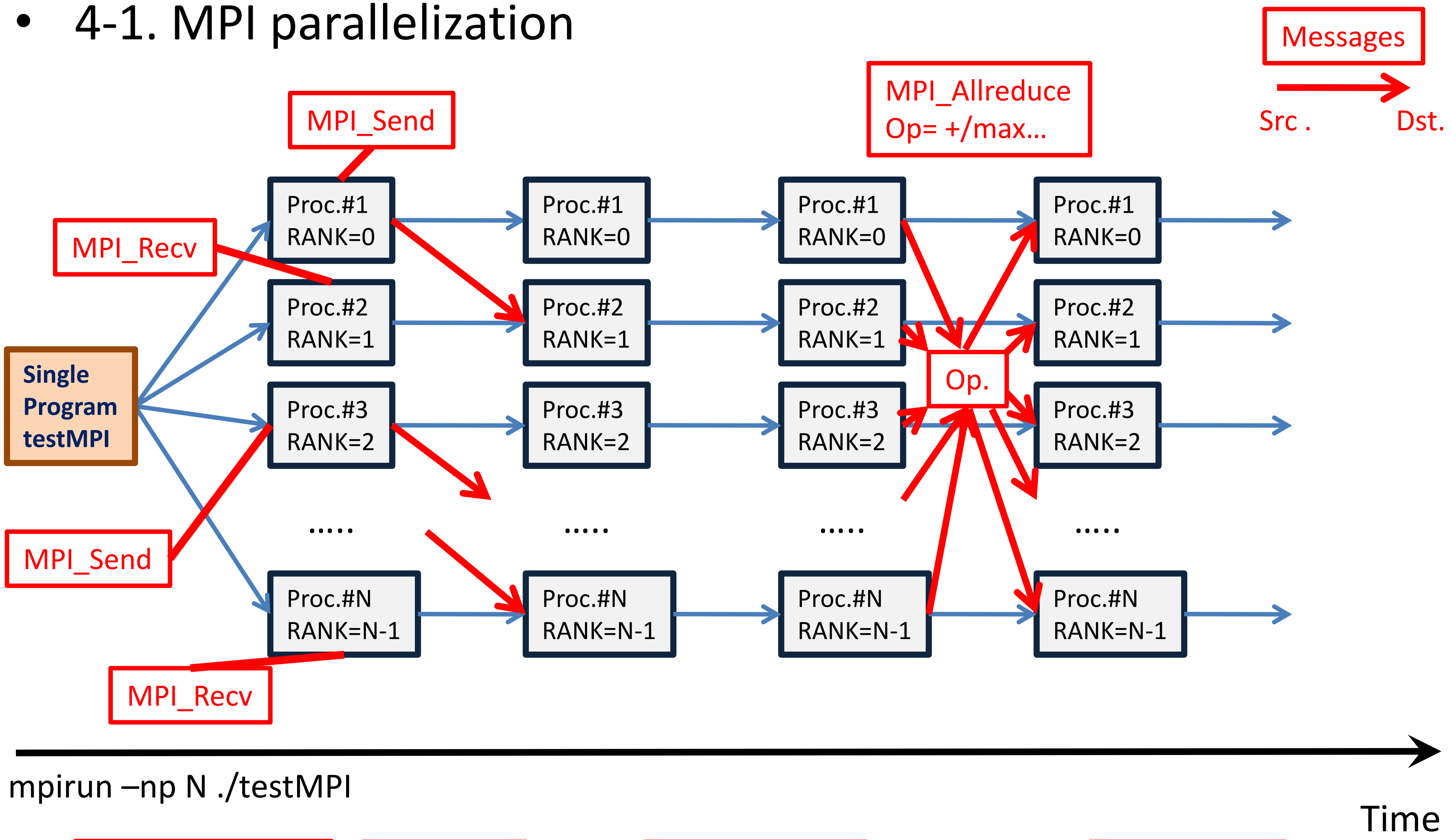
- Here I only explain how parallelize quark solver using MPI.
- I employ again the 1D Wilson like equation as a target problem to be parallelized.

4. Parallelization

- 4-1. MPI parallelization
  - Multiple process using an identical program executable.
  - Run the program code in parallel.
  - Each running process has its own identifier : so called MPI RANK number.
  - Each running process can communicate to/from other processes through the MPI API routine calls. The data to be send/received are called a Message.
  - To identify the destination or the source of a message, we use the MPI RANK.

4. Parallelization

- 4-1. MPI parallelization



```
% mpirun –np N ./testMPI
```

MPI_Init
MPI_Comm_size
MIP_Comm_rank

MPI_Send
MPI_Recv

MPI_Allreduce
MPI_Barrier

MPI_Finalize

- 4-1. MPI parallelization
  - To parallelize the quark solver we need to know the usage of the following limited MPI routines.

MPI_Init
MPI_Comm_size
MIP_Comm_rank
MPI_FInalize

MPI environment set up
MPI environment request
These routines should be called at first and at last in the program.

MPI_Send

MPI_Recv

1-to-1 Communication

These are to be used to exchange data missing in current process.

Used in Wilson-Dirac matrix multiplication.

MPI_Allreduce

MPI_Barrier

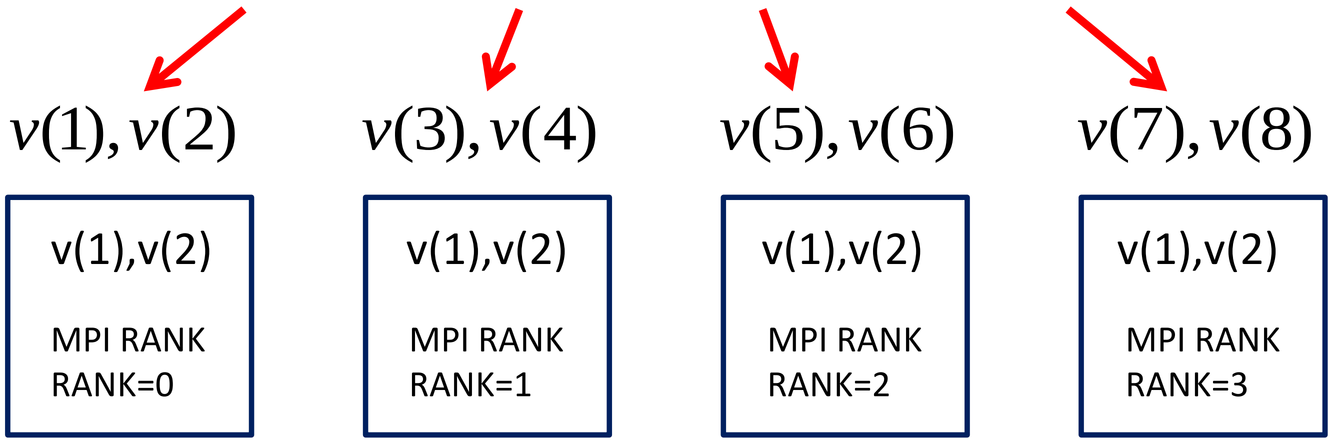
Global communication

Global summation.
Barrier wait.

Used in the inner-product operation in the solver.
Used to take a timing at W.-D. matrix multiplication.

- 4-1. MPI parallelization
- As an example: 1D Helmholz equation. (N_{SITE}=8 case)
  - Data(vector) are sliced into several pieces (number of processes=4).

$$\text{a vector}: v \qquad v(1), v(2), v(3), v(4), v(5), v(6), v(7), v(8)$$



Local array
v(1:2)
N_{LSITE}=2

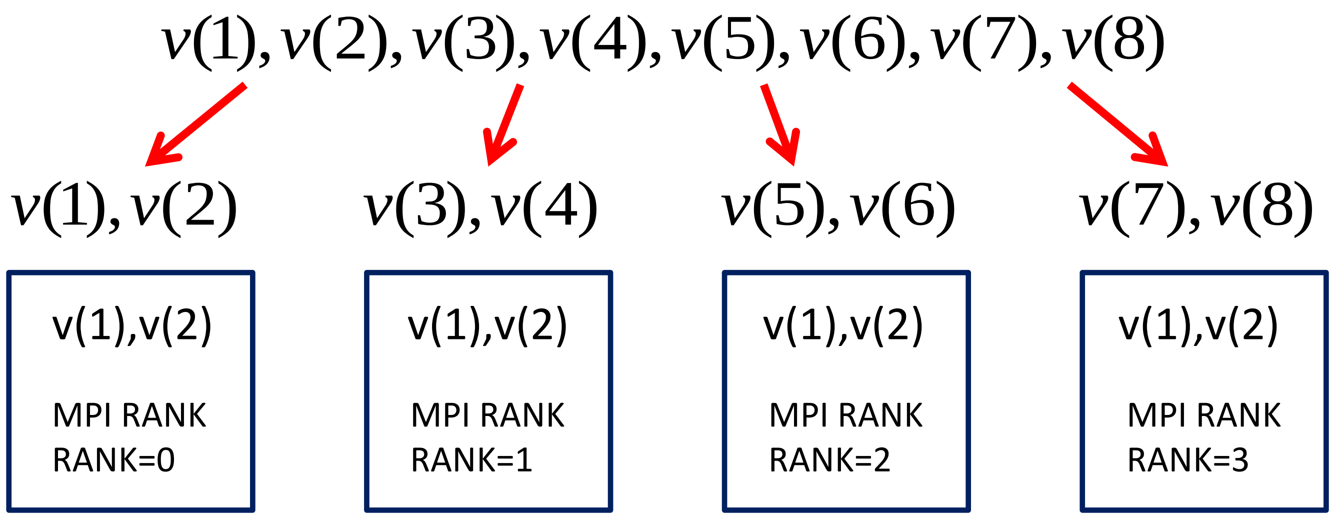
$$n = i + N_{\text{LSITE}} \times \text{RANK} = i + 2 \times \text{RANK}$$

$$i = \text{mod}(n-1, N_{\text{LSITE}}) + 1 = \text{mod}(n-1,2) + 1$$

Global site index *n*
Local site index *i*
Index conversion eq.

- All numerical vector operation are applied to the local arrays. The local index is useful. If you need the global site index, you can recover it using the MPI RANK number and local site index.

- 4-1. MPI parallelization
- 1D Helmholz equation. (N_{SITE}=8 case)

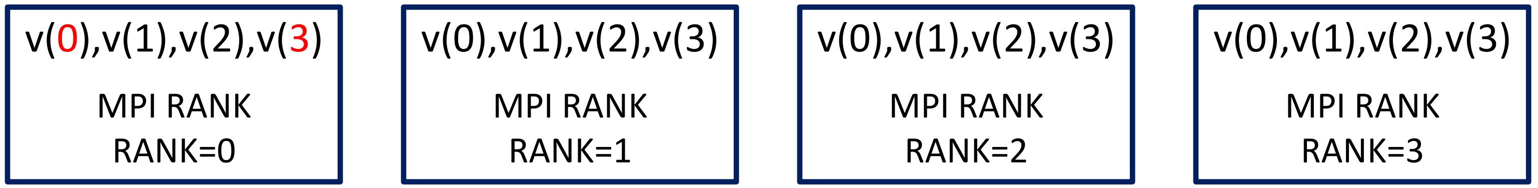


$v(1), v(2), v(3), v(4), v(5), v(6), v(7), v(8)$
$v(1), v(2)$
$v(3), v(4)$
$v(5), v(6)$
$v(7), v(8)$
v(1),v(2) MPI RANK RANK=0
v(1),v(2) MPI RANK RANK=1
v(1),v(2) MPI RANK RANK=2
v(1),v(2) MPI RANK RANK=3

Local array
v(1:2)
N_{LSITE}=2

– Matrix vector operation ($w = \left(-\Delta + \kappa^2\right) v$) needs nearest neighboring data. It has been used a ghost site technique to simplify this data communication.

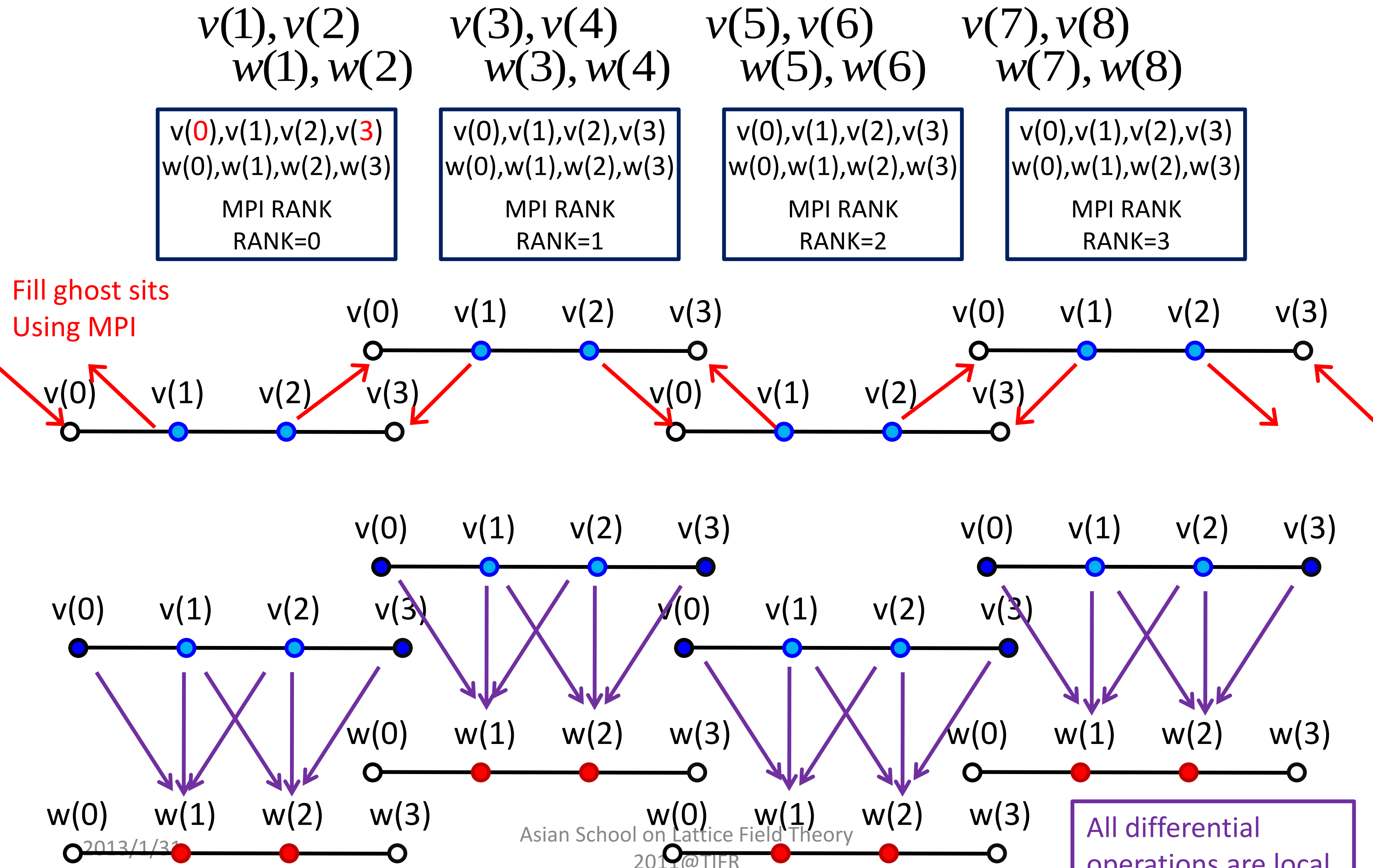
Local array is extended to both sides as v(0:3). The extended sites are called “Ghost sites”.



v(0),v(1),v(2),v(3) MPI RANK RANK=0
v(0),v(1),v(2),v(3) MPI RANK RANK=1
v(0),v(1),v(2),v(3) MPI RANK RANK=2
v(0),v(1),v(2),v(3) MPI RANK RANK=3

- 4-1. MPI parallelization

- 1D Helmholz equation. (N_{SITE}=8 case) $w = \left(-\Delta + \kappa^2\right) v$



All differential operations are local.

- 4-1. MPI parallelization
- 1D Helmholz equation. (N_{SITE}=8 case)

$$w = \left(-\Delta + \kappa^2\right) v$$

```
UP_RANK = mod(RANK+1, $N_{MPI}$)
DN_RANK = mod(RANK-1+$N_{MPI}$, $N_{MPI}$)
call MPI_Barrier(........)
if (mod(RANK,2) == 0) then
  call MPI_Send(v(2),1,MPI_REAL8,UP_RANK,.....)
  call MPI_Send(v(1),1,MPI_REAL8,DN_RANK,.....)
  call MPI_Recv(v(0),1,MPI_REAL8,DN_RANK,.....)
  call MPI_Recv(v(3),1,MPI_REAL8,UP_RANK,.....)
else
  call MPI_Recv(v(0),1,MPI_REAL8,DN_RANK,.....)
  call MPI_Recv(v(3),1,MPI_REAL8,UP_RANK,.....)
  call MPI_Send(v(2),1,MPI_REAL8,UP_RANK,.....)
  call MPI_Send(v(1),1,MPI_REAL8,DN_RANK,.....)
endif
do $i = 1, N_{LSITE}$
  $w(i) = -v(i+1) + \left(2+\kappa^2\right)v(i) - w(i-1)$
enddo
```

Compute Upward RANK number
Compute Downward RANK number

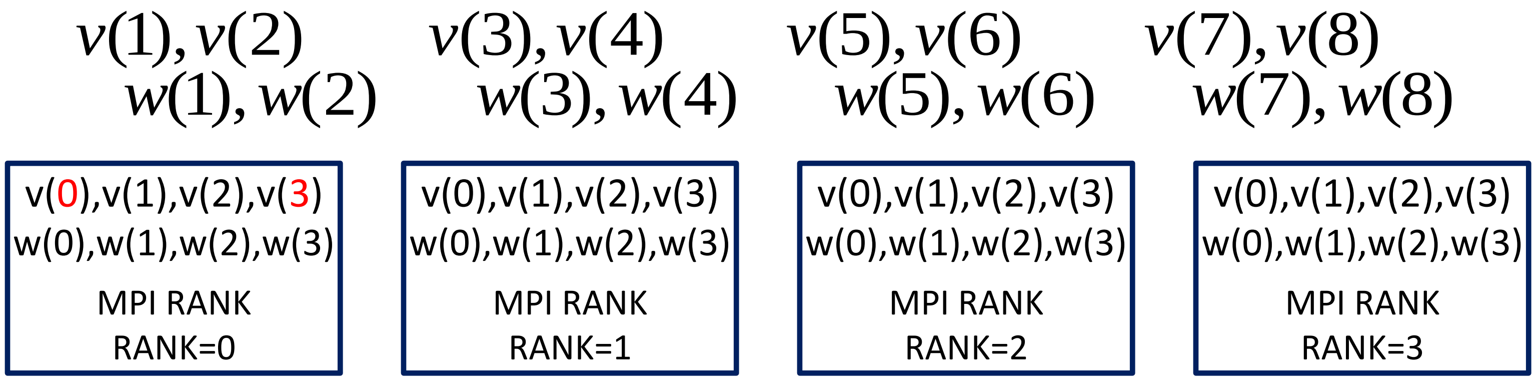
IF RANK number is a even number
1st :send data to odd number RANK processes
2nd :receive data from odd RANK processes.

IF RANK number is a odd number
1st :receive data from odd RANK processes.
2nd :send data to odd number RANK processes

Send and Receive processes should be paired in the 1-to-1 communication. Otherwise we encounter a process deadlock.
This even/odd pairing manner is a well used programming pattern.

- 4-1. MPI parallelization
- 1D Helmholz equation. (N_{SITE}=8 case)

$$w = \left(-\Delta + \kappa^2\right)v$$


$v(1), v(2)$ $w(1), w(2)$
$v(3), v(4)$ $w(3), w(4)$
$v(5), v(6)$ $w(5), w(6)$
$v(7), v(8)$ $w(7), w(8)$
v(0),v(1),v(2),v(3) w(0),w(1),w(2),w(3) MPI RANK RANK=0
v(0),v(1),v(2),v(3) w(0),w(1),w(2),w(3) MPI RANK RANK=1
v(0),v(1),v(2),v(3) w(0),w(1),w(2),w(3) MPI RANK RANK=2
v(0),v(1),v(2),v(3) w(0),w(1),w(2),w(3) MPI RANK RANK=3

- CG/BiCGStab/any Krylov algorithms use inner product operations. This is done as

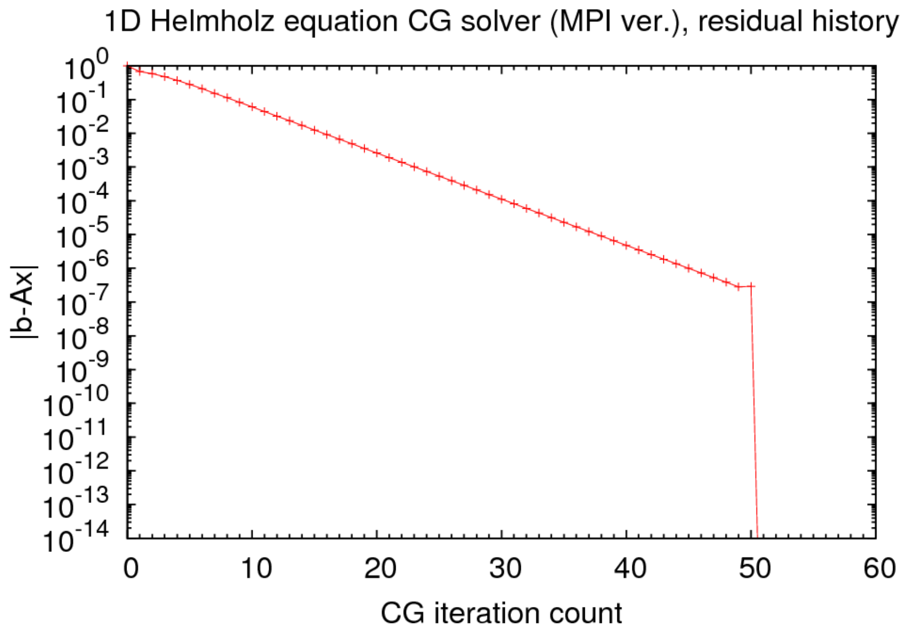
$$\mathrm{prod} = \langle v \mid w \rangle$$

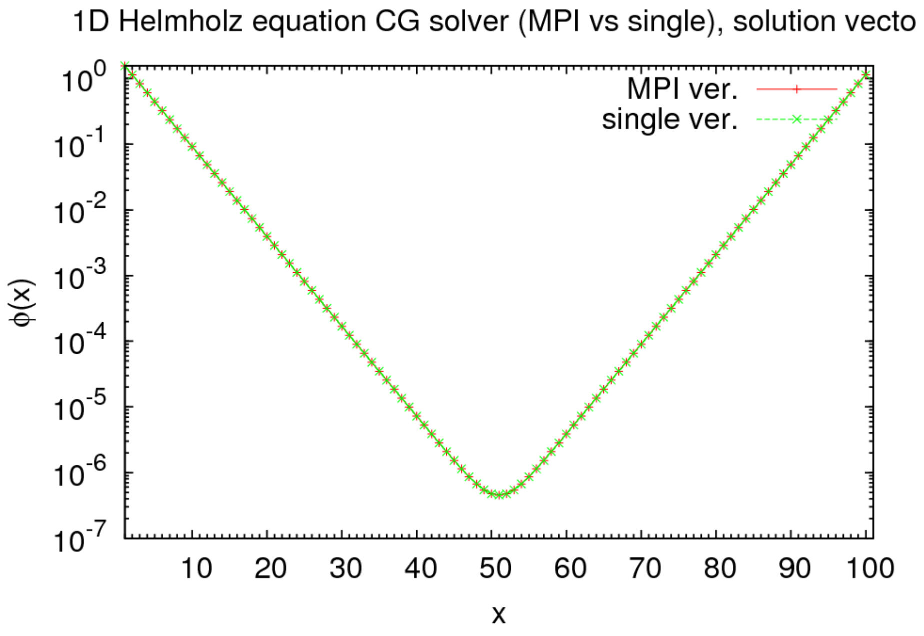
```
prod = 0.0
rtmp = 0.0
do  i = 1, $N_{\mathrm{LSITE}}$
    rtmp = rtmp + v(i) * w(i)
enddo
call MPI_Allreduce(rtmp,prod,1,MPI_REAL8,MPI_SUM,....)
return ! prod
```

All MPI process obtain the same numerical number in the “prod”

- 4-1. MPI parallelization
- 1D Helmholz equation. (N_{SITE}=100 case)
- Fortran program:

[http://theo.phys.sci.hiroshima-u.ac.jp/~ishikawa/ASLFT2010/MPI_Helmholz1DCG.tar.gz]





We have to check the consistency to the non-parallel version in parallel programming.
It is more better to check the consistency by varying the number of processes.

Thank you!

Problems

(1) Try to extend the 1D Helmholz solver to 2D or 3D versions.

(2) Try to extend the 1D Wilson like equation solver to a 4D Free Wilson-Dirac quark solver. In this case we need complex number operations and vectors. You have to extend or slightly modify the BiCGStab algorithm for complex non-Hermitian matrixes.

$$\boxed{\langle v | w \rangle \neq \langle w | v \rangle \text{ for complex vectors.}}$$

(3) Try to write a SSOR preconditioned solver for 1D Wilson like equation solver.

(4) Parallelize the 1D Wilson like equation solver if you can use MPI environment.

(5) Parallelize the 2D/3D Helmholz solver.

(6) [Advanced] How about the SSOR preconditioner in parallel case?

References

- LQCD and simulations

  - H.J.Rothe, “Lattice gauge theories: an Introductuion”, 3rd ed., World Scientific 2005, 978-9812561688.
  - I. Montvay and G.Munster, “Quatntum Fields on a Lattice”, Cambridge Univ. Press 1997, 978-0521599177.
  - C.Itzykson and J.-M.Drouffe, “Statistical field theory vol.2”, Cambridge Univ. Press 1991, 978-0521408059.
  - A.D.Kennedy, “The Hybrid Monte Carlo algorithm on parallel computers”, Parallel Computing **25** (1999) 1311-1339.

- Linear eq. solvers

  - G.H.Golub and C.F.van Loan, “Matrix Computations” 3rd ed., The Johns Hopkins Univ. Press. 1996, 978-0801854149.
  - R.Barret et al.,”Templetes for the Solution of Linear Systems: Building Blocks for Iterative Methods”, SIAM 1987, 978-0898713282.
  - Y.Saad,”Iterative Methods for Sparse Linear Systems”, 2nd ed., SIAM 2003, 978-0898715347.
  - H.A.van der Vorst, “Iterative Krylov Methods for Large Linear Systems”, Cambridge Univ. Press 2009, 978-0521183703.

History

- 2013/01/31: Page 16, CG algorithm convergence theorem was corrected.

$$\left\| x^{*} - x^{(m)} \right\|_{A} \leq 2 \left[\frac{\sqrt{K}+1}{\sqrt{K}-1} \right]^{m} \left\| x^{*} - x^{(0)} \right\|_{A}$$ Wrong

$$\left\| x^{*} - x^{(m)} \right\|_{A} \leq 2 \left[\frac{\sqrt{K}-1}{\sqrt{K}+1} \right]^{m} \left\| x^{*} - x^{(0)} \right\|_{A}$$ Right