ISE-TR-87-64

Hyperplane vs. Multicolor Vectorization of Incomplete LU Preconditioning for the Wilson Fermion on the Lattice

Yoshio Oyanagi

August 5, 1987

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

ISE-TR-87-64
August 5, 1987

Hyperplane vs. Multicolor Vectorization of Incomplete LU Preconditioning for the Wilson Fermion on the Lattice

Yoshio Oyanagi

Institute of Information Sciences, University of Tsukuba
Sakura-mura, Niihari-gun, Ibaraki, 305 JAPAN

We compare the hyperplane and 16-color vectorizations of minimal residual and conjugate residual methods with an incomplete LU preconditioning to solve the lattice Dirac equation in the Wilson formulation. The performance was assessed in terms of the Euclidean distance from the true solution for various hopping parameters on a quenched gauge configuration at β=5.5 on an 8^4 lattice. The 16-color vectorization requires 2~4 times more iterations than the hyperplane vectorization. Even with the best choice of the acceleration parameter, the latter method is preferable than the former unless the computing time for one iteration differs by a factor of two or more.

(submitted to Journal of Information Processing)

1. Introduction

The numerical simulations of the lattice gauge theory, the quantum chromodynamics on the lattice[1], has proven to be useful for extracting quantitative predictions about the hadron physics from the first principles. The lattice gauge theory is formulated on a four-dimensional hypercubic lattice. In the Wilson formulation[1], the fermion (quark) field $\psi_{i\alpha}(x)$ is a 12 component complex quantity allocated on a lattice site and has two indices: one for the Dirac index (i=1, 2, 3, 4), the other for the color (α=1, 2, 3). On the other hand, the gauge (gluon) field U(x,y) is a 3×3 unitary matrix with unit determinant and is defined on a link, i.e. the side between the nearest neighbor lattice sites, x and y. The rows and columns correspond to the color indices. The reciplocity requires $U(y,x) = U(x,y)^{\dagger}$.

In the numerical simulation of the lattice gauge theory, most of the computer time is spent in solving the Dirac equation on the lattice,

$$A \psi = b \tag{1}$$

where A is a large sparse non-Hermitian matrix which depends on the background gauge field U and a parameter κ called "hopping parameter" as described in the following section. The hopping parameter is a measure of coupling between the quark field components at the nearest neighbor sites and is related to the mass of the quark m_q in such a way that m_q is proportional to $1/\kappa - 1/\kappa_c$. When κ approaches κ_c, the equation becomes nearly singular and difficult to solve.

Since the size of the coefficient matrix A is very large, the amount of work and storage required in direct methods such as Gauss elimination is nearly prohibitive except for small lattices, such as 4^4. The standard procedure so far has been the conjugate gradient (CG) method[2] for $A^\dagger A$ or $AA^\dagger$, which gives the exact solution in finite steps if the round-off error is absent. In the previous paper[3] we presented a fast method based on the conjugate residual (CR) method and an incomplete LU (ILU) decomposition and proposed a hyperplane vectorization of the ILU preconditioning. Recently a different vectorization based on a 16-color classification appeared in the literature[4] and based upon this vectorization, they claimed that the ILU preconditioning was worse than other preconditioning methods especially when the hopping parameter κ is close to the critical value κ_c.

In the present paper we will compare the hyperplane and 16-color vectorizations of an incomplete LU preconditioning as applied to the minimal residual (MR) as well as the conjugate residual (CR) methods. For the two ways of vectorization, we want to assess the performance as a function of the hopping parameter κ on a quenched gauge configuration at β=5.5 on an 8^4 lattice. We measured the error $\|\psi_i - A^{-1}b\|$, where ψ_i is the value of ψ at the i-th iteration, as a function of i in the two cases. The CPU time is not a good measure of the performance, since it critically depends on the architecture and especially on the fine-tuning of the code[3].

In the next section we will describe the Wilson fermion on the lattice; section 3 discusses the conjugate residual methods; section 4 contains a thorough description of the incomplete LU decomposition; section 5 discusses the vectorization of multiplication and ILU preconditioning of the Wilson fermion matrix; section 6 gives a brief description of 16-color vectorization; section 7 discusses numerical results.

2. Wilson Lattice Fermions

The coefficient matrix A is defined as a block matrix {A(x,y)} where x and y represent generic lattice sites on a four-dimensional hypercubic lattice of arbitrary size $n_1 \times n_2 \times n_3 \times n_4$. The site x is specified by four integer coordinates (x_1, x_2, x_3, x_4) where $x_\mu = 1, 2, \ldots n_\mu$ $(\mu=1, 2, 3, 4)$.

Each block A(x,y) has a structure due to the internal degrees of freedom of the quark field. It is a 12×12 complex matrix, whose rows (and columns also) are specified by the pair of Dirac (i,j=1, 2, 3, 4) and color indices (α,β=1, 2, 3).

In greater detail the Wilson fermion matrix is given as follows:

$$\begin{aligned}
A(x,y)_{i\alpha,j\beta} &= \delta_{ij}\,\delta_{\alpha\beta}, && \text{if } y = x\\
A(x,y)_{i\alpha,j\beta} &= -\kappa(1-\gamma^\mu_{ij})\,U(x,y)_{\alpha\beta} && \text{if } y = x+\hat{\mu}\\
A(x,y)_{i\alpha,j\beta} &= -\kappa(1+\gamma^\mu_{ij})\,U(x,y)_{\alpha\beta} && \text{if } y = x-\hat{\mu}\\
A(x,y)_{i\alpha,j\beta} &= 0 && \text{otherwise}
\end{aligned} \tag{2}$$

Here $y = x \pm \hat{\mu}$ means that the site y lies next to the site x in the positive (negative) μ-direction. The 4×4 complex matrices γ^μ are the Dirac's γ matrices defined by

$$\gamma^1=\begin{bmatrix}0&0&0&-i\\0&0&-i&0\\0&i&0&0\\i&0&0&0\end{bmatrix},\ \gamma^2=\begin{bmatrix}0&0&0&-1\\0&0&1&0\\0&1&0&0\\-1&0&0&0\end{bmatrix},\ \gamma^3=\begin{bmatrix}0&0&-i&0\\0&0&0&i\\i&0&0&0\\0&-i&0&0\end{bmatrix},\ \gamma^4=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&-1&0\\0&0&0&-1\end{bmatrix}. \quad (3)$$

We note that only one element in each row is nonzero and the value is either ± 1 or $\pm i$.

Since A is related to the discretized version of the Dirac equation in the continuum, the matrix A as a whole has a structure similar to the matrix generated by discretization of elliptic or parabolic partial differential equation; that is, an off-diagonal block A(x,y) is non-zero only when x and y are adjacent with each other.

3. Conjugate Residual Methods

A class of iterative methods for solving the system of linear equations by decreasing the norm of the residual vector $\|A\psi - b\|_2$ has been proposed[5-6]. The algorithm consists of iterative steps starting with

$$r = b - A\psi,\ p = r$$

and repeating

```
$\alpha = (r, Ap) / (Ap, Ap)$
$\psi = \psi + \alpha p$
$r = r - \alpha Ap$
update p
```

till convergence is achieved. Here the complex coefficient α is so determined as to minimize the norm of the new residual $\|r - \alpha Ap\|_2$.

The variants differ in the way to update the new direction vector p. The simplest choice would be to set p = r. In this case only three vectors ψ, r and q=Ar have to be stored in the memory. We call this algorithm the minimal residual (MR) method.

The convergence would be faster if the coefficient matrix A is approximately proportional to a unit matrix. More precisely, if we denote r_i the residual vector at the i-th iteration, we have[6],

$$\frac{\| r_{i+1} \|^2}{\| r_i \|^2} \leq 1 - \frac{\lambda_{min}(H)^2}{\lambda_{max}(A^\dagger A)} , \qquad (4)$$

provided H=(A† + A)/2, the Hermitian part of A, is positive-definite. Here λ_{min} and λ_{max} denote the minimum and maximum eigenvalues, respectively. In the algorithm, the positivity of H is crucial. If the positivity is lost (e.g. for $\kappa > \kappa_c$), the coefficient α becomes zero or very small and the ψ is no longer improved.

In the case of the CG method for positive symmetric linear equation Ax=b, the conjugacy of the correction vectors $\{p_i\}$ with respect to A plays an important role in reducing the bilinear form (x, b - Ax). By only making p_{i+1} conjugate to p_i, the former is automatically conjugate to all preceding correction vectors p_{i-1}, p_{i-2}, ... , p_1. In our case the conjugacy of the correction vectors with respect to $A^\dagger A$ also plays an important role in reducing $\|r\|^2$. The conjugacy, however, is not passed on to the new direction vectors, so that we can make p_{i+1} conjugate only to the correction vectors stored in the memory. We call this class of algorithm as conjugate residual (CR) method.

The simplest choice (CR(1) method) is to make p_{i+1} conjugate only to the previous vector p_i. More specifically, we start with

$$r = b - A\psi, \; p = r, \; q = Ap$$

and repeating

$$\begin{aligned}
\alpha &= (q, r) / (q, q) \\
\psi &= \psi + \alpha p \\
r &= r - \alpha q \\
s &= Ar \\
\beta &= -(q, s) / (q, q) \\
p &= r + \beta p \\
q &= s + \beta q
\end{aligned}$$

until the convergence is attained. One step of the CR(1) method entails one matrix multiplication, three inner products and four vector addition with scalar multiplication, that is it has one more inner product and two more vector additions as compared with the MR. The memory necessary to implement the CR(1) is larger, since it has two more working vectors p and s. Although the upper bound for the ratio of residuals (4) is valid for the CR methods, the residual for the CR decreases faster than that for the MR.

4. Incomplete LU Decomposition

Since the relative reduction of the norm of the residual in the MR and CR methods is bounded by (4), one can improve the convergence by transforming A to a matrix which is approximately equal to the unit matrix (or its scalar multiple).

If we have a matrix $\tilde{A}$, which is a good approximation to A, we can expect $\tilde{A}^{-1}A$ is closer to the unit matrix than A itself. If the solution of the equation $\tilde{A}\,x = y$ requires relatively small amount of computation, it would be easier to solve the equation

$$\tilde{A}^{-1} A\,\psi = \tilde{A}^{-1}\,b, \tag{5}$$

in stead of eq.(1). This observation is at the basis of any preconditioning technique. The more the $\tilde{A}$ resembles A, the better will be the convergence of the MR or CR methods. On the other hand, the solution of $\tilde{A}\,x = y$ should not be too time-exhausting, since we have to solve it once in each step.

Some years ago Meijerink and van der Vorst[7] proposed an incomplete LU factorization for the matrix originated in partial differential equations, which approximately decomposes A as

$$A = L\,U - N, \tag{6}$$

where L and U are lower and upper triangular matrices and N is the error of decomposition. By suitably choosing the non-zero entries of L and U, one can make L and U as sparse as the original A. If the error N is small, the factorized form LU plays the role of $\tilde{A}$ in eq.(5).

In the case of the Wilson fermions on the lattice, it can readily be shown[3] that the block triangular splitting of A,

$$\begin{aligned} L(x,y) &= A(x,y) && (x \geq y) \\ &= 0 && (x < y) \end{aligned} \tag{7}$$

$$R(x,y) = 0 \qquad (x > y)$$
$$\quad = A(x,y) \qquad (x \leq y)$$

provides an incomplete LU decomposition due to the projection operators $(1 \pm \gamma^{\mu})$, that is

$$L\ R = A + O(\kappa^2). \tag{8}$$

We use the symbol R instead of U for the right (upper) triangular matrix, since we have to reserve U for the gauge field.

The inequality of x and y is to be understood according to the site number x (denoted by IX in the program) in the usual manner,

$$x = (((x_4-1)\times n_3 + x_3-1)\times n_2 + x_2-1)\times n_1 + x_1. \tag{9}$$

The site number runs from 1 to n, where $n = n_1n_2n_3n_4$ is the total number of the lattice sites. The detailed algorithm of ILUMR and ILUCR methods is given in [3].

We found[3,8] that the convergence rate is further improved by a Gustafsson-type acceleration[7]. This is a trick of replacing the hopping parameter κ in the preconditioner LR by $c\kappa$, c being an appropriate constant. This acceleration can be understood as approximating cA by LR. The error N = L R - cA now has non-zero diagonal entries

$$N(x,x) = -(c-1)\ I \tag{10a}$$

as well as the usual off-diagonal entries

$$N(x,y) = c^2\kappa^2(1-\gamma^\mu)(1+\gamma^\nu)\ U(x,x+\hat{\mu})U^\dagger(y,y-\hat{\nu}) \tag{10b}$$

for the next-nearest pairs (x, y) with $y=x+\hat{\mu}-\hat{\nu}$. When the gauge field U as well as the fermion field ψ are nearly aligned, the effect of those two errors tends to cancel with each other, so that $(LR)^{-1}A$ is effectively closer to a constant multiple of a unit matrix. We found the best choice of c is 1.1~1.3. Unlike the acceleration parameter ω in the SOR method, the number of iterations needed to fulfill a convergence criterion does not critically depend on the choice of c.

5. Vectorization

We now need to discuss how to carry out the computation on a vector processor. The vectorization of MR and CR methods offers no problem since they consist of vector operations and matrix multiplication of a vector. We show in Fig. 1 the core of the code which gives r = A q. The arrays Q(IX, i, α) and R(IX, i, α) represent quark field where i and α denote the Dirac and color indices. The link connecting the site IX and its nearest neighbor in the positive μ-direction is numbered as LL = 4*IX - 4 + μ, so that the link number LL runs from 1 to 4*N. IGAM(i,μ) gives the Dirac index j for which $(\gamma_\mu)_{ij}$ = GAM(i,μ) $\neq$ 0. The array elements NRR(IX,μ) and NLL(IX,μ) give the site number which lies next to IX in the positive and negative μ-directions respectively. Due to the periodic boundary condition, NRR(IX,μ) cannot be given in terms of a linear form of IX. The hopping parameter is denoted by HK. Since there is no data dependency in the innermost loop DO 10, it would be straightforward for a compiler to vectorize the loop.

On the other hand, quite a bit more complex is the vectorization of the solution of the triangular equations $L\ p = q$ and $R\ s = p$. These equations are solved recursively in terms of the forward and backward substitutions, as

```
do x = 1, n
  $p(x) = q(x) - \sum_{y=1}^{x-1} L(x,y)\ p(y)$
```

and

```
do x = n, 1, -1
  $s(x) = p(x) - \sum_{y=x+1}^{n} R(x,y)\ s(y)$.
```

The algorithm is hard to vectorize since the previous variables are referred to in the loops. In order that the vectorized code may produce the same results as the scalar computers, we have to find a subset of lattice sites which are independent with each other and therefore can be computed concurrently. This cannot be done by dividing the lattice into sublattices with doubled lattice spacing. For example $p(1,1,1,3)$ depends on $p(1,1,1,1)$ via $p(1,1,1,2)$.

It is easily seen in the forward substitution, $p(x)$ depends on $p(y)$ if and only if there exists at least one sequence of lattice sites, $x=z^{(1)}, z^{(2)}, z^{(3)}, \dots, z^{(s)}=y$ for which $z^{(i)}$ and $z^{(i+1)}$ are adjacent with each other and $z^{(i)} > z^{(i+1)}$. One can prove by induction that $p(x_1, x_2, x_3, x_4)$ is dependent on $p(y_1, y_2, y_3, y_4)$ if and only if $x_\mu > y_\mu$ holds for all μ. This statement is also valid when we take the periodic (or antiperiodic) boundary conditions into account.

In our previous paper[3] we presented a hyperplane vectorization, which was originally proposed years ago for ILLIAC IV[10] and later revived for the ILU preconditioning of partial differential equations[11]. This approach is based upon the observation that the sites lying on a p-th hyperplane defined by

$$x_1 + x_2 + x_3 + x_4 = p = \text{const.} \qquad (11)$$

are independent of each other and that if p(x) depends on p(y) then y lies on a hyperplane with smaller p. We can start with p=4 and increment the constant p after each step, until p reaches its maximum value $NP = n_1 + n_2 + n_3 + n_4$.

A slight modification of the program in Fig. 1 yields a solver of L p = q, which is shown in Fig. 2. Here the solution p is overwritten on q in order to save the storage. The site numbers of those lattice sites whose nearest neighbor site in the positive μ-direction has smaller site number than themselves (i.e. connected in the matrix L) are reordered according to the hyperplane number IP and NNLR(IXP,μ) contains the IXP-th of such site numbers. The largest IXP on the IP-th hyperplane is given in NBLR(IP,μ). In the same way, the lattice sites whose nearest neighbor site in the negative μ-direction has smaller site number are stored in NNLL(*,μ). We note NBLR(NP,μ)+NBLL(MP,μ) = n for any μ. Since the compiler cannot identify the independency of the operations on Q in the loops DO 10 and DO 20, we have to put a compiler directive. Fig. 2 shows the one for HITAC S810.

6. Quasi-vectorization by Multicolor Method

Recently P. Rossi, C.T.H. Davies and G.P. Lepage[4] implemented the ILU preconditioning in terms of a 16-color sublattice "vectorization". They expressed the coordinate x on the lattice as

$$x = 2y + \eta \tag{12}$$

with

$$y = (y_1, y_2, y_3, y_4) \qquad 1 \le y_\mu \le n_\mu/2$$

and

$$\eta = (\eta_1, \eta_2, \eta_3, \eta_4) \qquad \eta_\mu = 0 \text{ or } 1,$$

and treated all the sites labeled by different y but identical η simultaneously.

Although the sites with the same η are not connected directly, they are not independent as we saw in the previous section, so that this method does not give the same result as the original ILU preconditioning. It should be regarded as a different solver based on the vector iteration in the sense of Schendel[12].

We show in Fig. 3, the relative error e of the 16 color ILU preconditioning for a complex random righthand vector b (both real and imaginary parts are normal random numbers and normalized as $\|b\|_2=1$). The error e is defined by

$$e = \| U_{16}^{-1}L_{16}^{-1}\, b - U^{-1}L^{-1}\, b \|_2 \;/\; \| U^{-1}L^{-1}\, b \|_2 \tag{13}$$

where L_{16}^{-1} and U_{16}^{-1} represent the 16-color vectorization of L^{-1} and U^{-1}. As is expected, the error is small for small κ, whereas it gets worse when κ becomes larger. One may think that since the ILU preconditioning itself is an approximation, exact identity of numerical algorithm may not be necessary in the vectorization. We will see the effect in the next section.

7. Summary and Results

In in Fig. 4 the Euclidean norm of the error $\|\psi_i - A^{-1} b\|$ is given as a function of i, the iteration number, for the ILUMR algorithm vectorized by the hyperplane and 16-color methods. The acceleration parameter c is set to 1.0 (no acceleration). The gauge configuration was taken from a quenched simulation at $\beta=5.5$ on an 8^4 lattice. The hopping parameters are $\kappa=0.17$, 0.18, 0.181 and 0.183. The critical value κ_c for which the pion mass vanishes in this configuration is 0.1844 ± 0.0009[8]. The righthand side b is a complex gaussian random vector described in the previous section. The initial value ψ_0 is set equal to b. We do not plot the residual $\|b-A\psi_i\|_2$, since it would require an extra computation. The vector r in ILUCR or ILUMR algorithm is a modified residual $(LR)^{-1}(b-A\psi_i)$ and depends on the vectorization and acceleration of ILU preconditioning.

We also show in Table 1 the number of iterations until $\|\psi_i - A^{-1}b\| < 10^{-4}$ is attained for the various choices of c for ILUMR and ILUCR methods in both hyperplane (column h) and 16-color (column m) vectorizations. For all values of hopping parameters the 16-color version requires 2 ~ 4 times more iterations. From the result in Table 1, the optimum value of c in 16-color vectorization is larger than that in the hyperplane method. Even if we compare the best choices of c for each case, the difference is large, especially for the range of κ of our interest, which is close to κ_c. The critical slowing down is more striking in the multicolor vectorization.

This result can be understood in terms of the velocity of information running across the lattice. In the 16-color vectorization of ILU preconditioning, an element of the residual vector influences only the elements of ψ in a hypercube which lies next to it. The velocity is only $\sqrt{7}$ per iteration in the lattice unit. This feature exhibits a striking contrast to the hyperplane vectorization, in which any element of the residual can give influence on the value on any other lattice site, albeit incomplete.

On the other hand, the multicolor vectorization posesses some computational advantages. For one thing, the vector length in the 16-color vectorization is n/16, which is in practical cases larger than $n/2(n_1+n_2+n_3+n_4-3)$, the average vector length in the hyperplane vectorization. Moreover the access to the memory is more regular in the former method, so that the execution time for one iteration is shorter in the multicolor vectorization than in the hyperplane one, especially on a vector machine with slow memory access. This fact may cover the shortage of the former that more iterations are necessary than the latter. It is also to be noted that the multicolor method can be easily implemented on a highly parallel array of processors with distributed memory. We conclude, however, that the hyperplane vectorization is superior than the multicolor vectorization, unless the execution of the former is at least twice (or more) faster than that of the latter.

Acknowledgement

The numerical calculation for the present work was carried out on HITAC S-810/10 at KEK, National Laboratory for High Energy Physics. We thank the Theory Division of KEK for its warm hospitality. We are indebted to M. Fukugita, A. Ukawa, Y. Iwasaki, M. Mori, M. Natori, K. Murata and U. Ushiro for stimulating discussions. We are also greatful to S. Fujino and K. Takeda for informing us of Lamport's paper[8]. Our work was supported in part by the Grants-in-Aid for Scientific Research of Ministry of Education, Science and Culture (No. 61540142).

References

1. K. G. Wilson, Phys. Rev. D10, (1974), 2445-2459.
2. M. R. Hestenes and E. Stiefel, J. Res. Nat. Bur. Standards 49, (1952), 409.
3. Y. Oyanagi, Comput. Phys. Commun. 42, (1986), 333-343.
4. P. Rossi, C. T. H. Davies and G. P. Lepage, University of California, San Diego report UCSD-PTH 87/08.
5. P. Concus and G. H. Golub, in : Lecture Notes in Economics and Mathomatical Systems, vol. 134, eds. R. Glowinski and J. L. Lions (Springer-Verlag, Berlin, 1976) p. 56-65.

P. K. W. Vinsome, in : Proc. Fourth Symp. on Reservoir Simulation, Soc. Petroleum Eng. AIME (1976) p. 149.

6. S. L. Eisenstat, H. C. Elman and M. H. Schults, SIAM J. Numer. Anal. 20, (1983), 345-357.

7. J. A. Meijerink and H. Z. van der Vorst, Math. Comput. 31, (1977), 148-162.

8. M. Fukugita, Y. Oyanagi and A. Ukawa, Phys. Rev. D 36, 2, (1987) in press.

9. I. Gustafsson, BIT 18, (1978), 142.

10. L. Lamport, Comm. of ACM 17, (1974), 83-93.

11. Y. Ushiro, M. Nishikata and F. Nagahori, Hitachi Hyoron 65, (1983), 557-562 (in Japanese).

12. U. Schendel, Introduction to Numerical Methods for Parallel Computers, trans. by B. W. Conolly, Ellis Horwood Limited, Chichester, 1984, p.35.

Figure Captions

Fig. 1 A program for R = A Q

Fig. 2 A program for $Q \leftarrow L^{-1} Q$

Fig. 3 Relative error of 16-color preconditioning as a function of the hopping parameter.

Fig. 4 The error $\|\psi_i - A^{-1}b\|$ as a function of the number of iterations i for 16-color preconditioning (upper curve) and hyperplane preconditioning (lower curve).

```
      COMPLEX U(4*N,3,3),Q(N,4,3),R(N,4,3),QQ1,QQ2,QQ3,GAM(4,4),GM
      INTEGER IGAM(4,4),NRR(N,4),NLL(N,4)
C
      DO 5 IALPHA=1,3
      DO 5 I=1,4
      DO 5 IX=1,N
    5   R(IX,I,IALPHA)=Q(IX,I,IALPHA)
C
      DO 10 MU=1,4
      DO 10 I=1,4
        J=IGAM(I,MU)
        GM=GAM(I,MU)
        DO 10 IX=1,N
           NR=NRR(IX,MU)
           LR=4*IX-4+MU
           QQ1=Q(NR,I,1) - GM*Q(NR,J,1)
           QQ2=Q(NR,I,2) - GM*Q(NR,J,2)
           QQ3=Q(NR,I,3) - GM*Q(NR,J,3)
           R(IX,I,1)=R(IX,I,1) - HK*(U(LR,1,1)*QQ1 + U(LR,1,2)*QQ2 +
     1                              U(LR,1,3)*QQ3)
           R(IX,I,2)=R(IX,I,2) - HK*(U(LR,2,1)*QQ1 + U(LR,2,2)*QQ2 +
     2                              U(LR,2,3)*QQ3)
           R(IX,I,3)=R(IX,I,3) - HK*(U(LR,3,1)*QQ1 + U(LR,3,2)*QQ2 +
     3                              U(LR,3,3)*QQ3)
C
           NL=NLL(IX,MU)
           LL=4*NL-4+MU
           QQ1=Q(NL,I,1) + GM*Q(NL,J,1)
           QQ2=Q(NL,I,2) + GM*Q(NL,J,2)
           QQ3=Q(NL,I,3) + GM*Q(NL,J,3)
           R(IX,I,1)=R(IX,I,1) - HK*(CONJG(U(LL,1,1))*QQ1 +
     1        CONJG(U(LL,2,1))*QQ2 + CONJG(U(LL,3,1))*QQ3)
           R(IX,I,2)=R(IX,I,2) - HK*(CONJG(U(LL,1,2))*QQ1 +
     2        CONJG(U(LL,2,2))*QQ2 + CONJG(U(LL,3,2))*QQ3)
           R(IX,I,3)=R(IX,I,3) - HK*(CONJG(U(LL,1,3))*QQ1 +
     3        CONJG(U(LL,2,3))*QQ2 + CONJG(U(LL,3,3))*QQ3)
   10 CONTINUE
```

Fig. 1 A program for R = A Q

```
      COMPLEX U(4*N,3,3),Q(N,4,3),QQ1,QQ2,QQ3,GAM(4,4),GM
      INTEGER IGAM(4,4),NRR(N,4),NLL(N,4),NBLR(NP,4),NBLL(NP,4),
     1     NNLR(N,4),NNLL(N,4)
C
      DO 30 IP=4,NP
        DO 30 MU=1,4
        DO 30 I=1,4
          J=IGAM(I,MU)
          GM=GAM(I,MU)
*VOPTION INDEP(Q)
          DO 10 IXP=NBLR(IP-1,MU)+1,NBLR(IP,MU)
            IX=NNLR(IXP,MU)
            NR=NRR(IX,MU)
            LR=4*IX-4+MU
            QQ1=Q(NR,I,1) - GM*Q(NR,J,1)
            QQ2=Q(NR,I,2) - GM*Q(NR,J,2)
            QQ3=Q(NR,I,3) - GM*Q(NR,J,3)
            Q(IX,I,1)=Q(IX,I,1) + HK*(U(LR,1,1)*QQ1 +
     1                U(LR,1,2)*QQ2 + U(LR,1,3)*QQ3)
            Q(IX,I,2)=Q(IX,I,2) + HK*(U(LR,2,1)*QQ1 +
     2                U(LR,2,2)*QQ2 + U(LR,2,3)*QQ3)
            Q(IX,I,3)=Q(IX,I,3) + HK*(U(LR,3,1)*QQ1 +
     3                U(LR,3,2)*QQ2 + U(LR,3,3)*QQ3)
   10     CONTINUE
C
*VOPTION INDEP(Q)
          DO 20 IXP=NBLL(IP-1,MU)+1,NBLL(IP,MU)
            IX=NNLL(IXP,MU)
            NL=NLL(IX,MU)
            LL=4*NL-4+MU
            QQ1=Q(NL,I,1) + GM*Q(NL,J,1)
            QQ2=Q(NL,I,2) + GM*Q(NL,J,2)
            QQ3=Q(NL,I,3) + GM*Q(NL,J,3)
            Q(IX,I,1)=Q(IX,I,1) + HK*(CONJG(U(LL,1,1))*QQ1 +
     1         CONJG(U(LL,2,1))*QQ2 + CONJG(U(LL,3,1))*QQ3)
            Q(IX,I,2)=Q(IX,I,2) + HK*(CONJG(U(LL,1,2))*QQ1 +
     2         CONJG(U(LL,2,2))*QQ2 + CONJG(U(LL,3,2))*QQ3)
            Q(IX,I,3)=Q(IX,I,3) + HK*(CONJG(U(LL,1,3))*QQ1 +
     3         CONJG(U(LL,2,3))*QQ2 + CONJG(U(LL,3,3))*QQ3)
   20     CONTINUE
   30 CONTINUE
```

Fig. 2 A program for $Q \leftarrow L^{-1}Q$

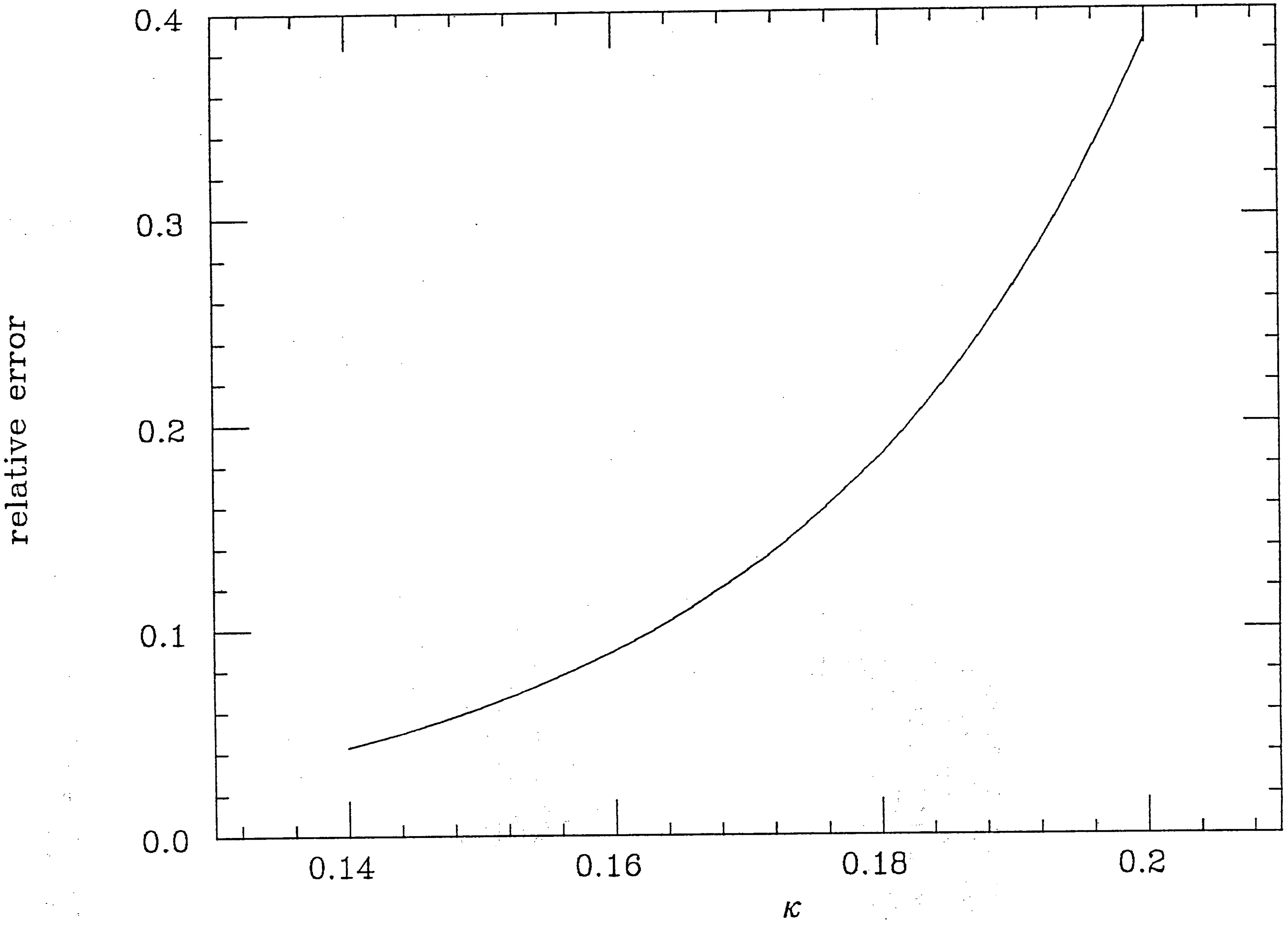
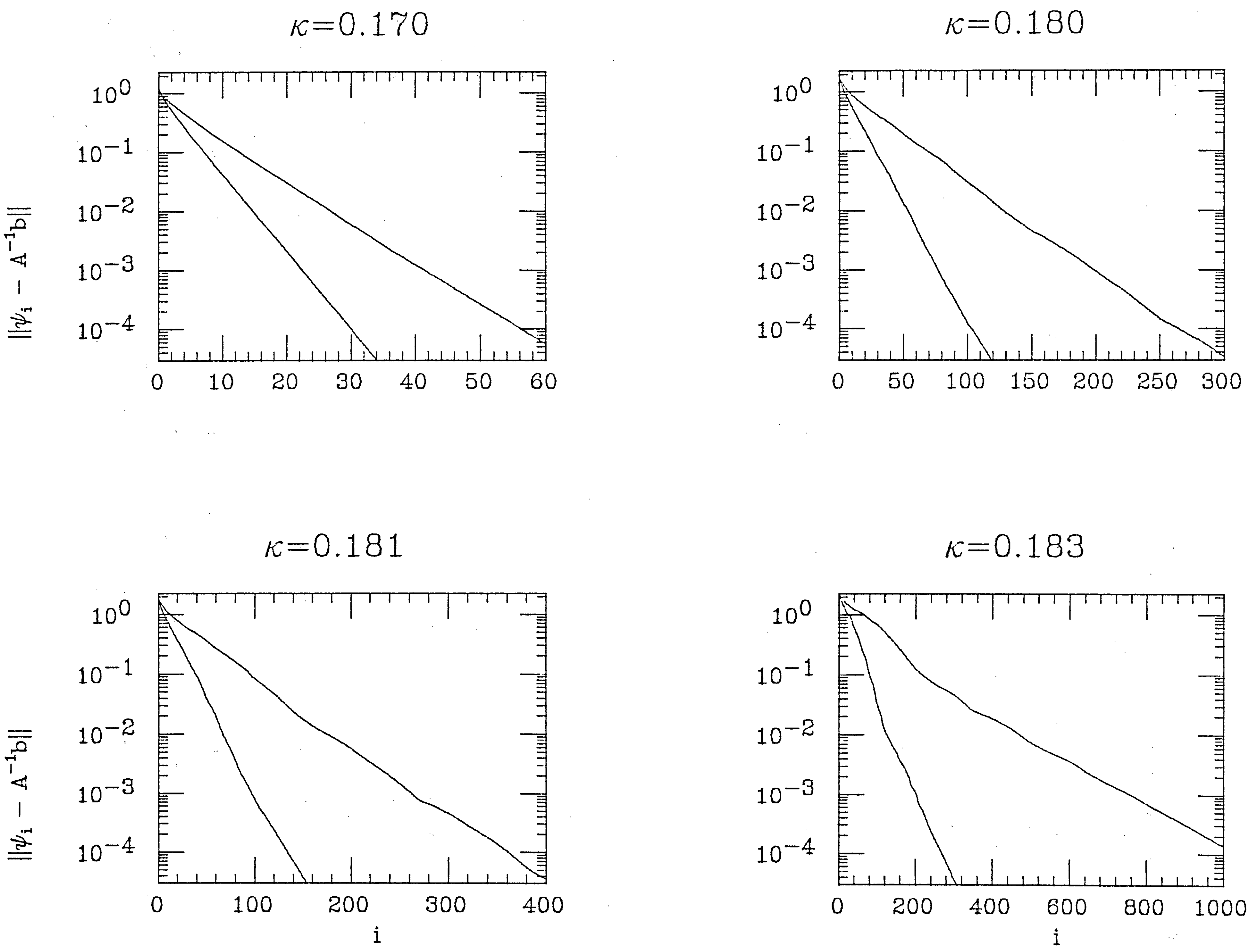


Fig. 3



κ=0.170
κ=0.180
κ=0.181
κ=0.183
$\|\psi_i - A^{-1}b\|$
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0 10 20 30 40 50 60
0 50 100 150 200 250 300
0 100 200 300 400
0 200 400 600 800 1000
i

Fig. 4

κ	0.170		0.180		0.181		0.183	
$(m_\pi a)^2$	0.83		0.24		0.18		0.02	
ILUMR	h	m	h	m	h	m	h	m
c=1.0	31	57	105	265	134	361	270	1045
c=1.1	25	52	79	227	97	312	178	838
c=1.2	$\underline{22}$	48	$\underline{68}$	201	$\underline{81}$	274	$\underline{138}$	687
c=1.3	25	45	78	183	94	244	147	574
c=1.4	38	43	117	168	153	221	394	495
c=1.5	80	$\underline{42}$	---	157	---	204	---	432
c=1.6	--	$\underline{42}$	---	$\underline{151}$	---	$\underline{195}$	---	400
c=1.7	--	45	---	160	---	204	---	$\underline{392}$
c=1.8			---	191	---	239	---	450
c=1.9			---	294	---	386	---	811
ILUCR(1)	h	m	h	m	h	m	h	m
c=1.0	26	46	82	183	100	245	191	660
c=1.1	23	43	69	165	82	219	149	543
c=1.2	$\underline{22}$	41	$\underline{66}$	150	$\underline{78}$	197	$\underline{131}$	452
c=1.3	24	39	74	139	89	180	139	388
c=1.4	36	$\underline{37}$	118	130	149	167	257	345
c=1.5	74	$\underline{37}$	---	123	---	157	---	316
c=1.6	--	$\underline{37}$	---	119	---	$\underline{151}$	---	297
c=1.7	--	40	---	$\underline{112}$	---	152	---	$\underline{292}$
c=1.8			---	138	---	169	---	319
c=1.9			---	202	---	254	---	466

Table 1
The number of iterations needed to attain $\|\psi_i - A^{-1}b\| < 10^{-4}$ in the hyperplane (column h) and 16-color (column m) vectorizations. The bar (---) denotes the failure of convergence. We underline the best choice of c in each case.

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-87-64

TITLE

Hyperplane vs. Multicolor Vectorization of Incomplete LU Preconditioning for the Wilson Fermion on the Lattice

AUTHOR(S)

Yoshio Oyanagi

REPORT DATE August 5, 1987	NUMBER OF PAGES 22
MAIN CATEGORY	CR CATEGORIES

KEY WORDS

Lattice gauge theory, Wilson fermions, vectorization, conjugate residual method, preconditioning

ABSTRACT

We compare the hyperplane and 16-color vectorization of minimal residual and conjugate residual methods with an incomplete LU preconditioning to solve the lattice Dirac equation in the Wilson formulation. The performance was assessed in terms of the Euclidean distance form the true solution for various hopping parameters on a quenched configuration at β=5.5 on an 8^4 lattice. The 16-color vectorization requires 2-4 times more iterations than the hyperplane vectorization. Even with the best choice of the acceleration parameter, the latter method is preferable than the former unless the computing time for one iteration differes by a factor to two or more.

SUPPLEMENTARY NOTES