hep-lat/9602019 15 Feb 1996

WUB 96-1

HLRZ 4/96

A Parallel SSOR Preconditioner for Lattice QCD

S. Fischer[a], A. Frommer[a], U. Glässner[b], Th. Lippert[c], G. Ritzenhöfer[c], and K. Schilling[b,c]

[a]Department of Mathematics, University of Wuppertal, 42097 Wuppertal, Germany

[b]Department of Physics, University of Wuppertal, 42097 Wuppertal, Germany

[c]HLRZ, c/o KFA-Jülich, D-52425 Jülich, Germany and DESY, Hamburg, Germany

Abstract

We present a parallelizable SSOR preconditioning scheme for Krylov subspace iterative solvers which proves to be efficient in lattice QCD applications involving Wilson fermions. Our preconditioner is based on a locally lexicographic ordering of the lattice points. In actual hybrid Monte Carlo applications with the bi-conjugate gradient stabilized method BiCGstab, we achieve a gain factor of about 2 in the number of iterations compared to conventional odd-even preconditioning. Whether this translates into similar reductions in run time will depend on the parallel computer in use. We discuss implementation issues using the 'Eisenstat-trick' and machine specific advantages of the method for the APE100/Quadrics parallel computer. In a full QCD simulation on a 512-processor Quadrics QH4 we find a gain in cpu-time of a factor of 1.7 over odd-even preconditioning for a $24^3 \times 40$ lattice.

1 Introduction

The computation of the effects of fermionic forces (from virtual quark-antiquark creation and annihilation processes) onto the vacuum structure of quantum chromodynamics (QCD) presents a severe bottleneck to the numerical evaluation of this fundamental theory of strong interactions.

The stochastic sampling of the vacuum state by the standard Hybrid Monte Carlo algorithm (HMC) *e.g.* implies the continual computation of fermionic Greens functions on a stochastic source ϕ. In terms of the discretized Dirac operator M, the fermionic Greens function (or 'quark propagator') is nothing but the solution of the linear equation

$$Mx = \phi, \tag{1}$$

where M is a huge sparse matrix, of rank $r = 3 \times 4 \times V$ and V is the volume of the underlying 4-dimensional space-time lattice. There are different discretizations of the Dirac operator in use. For the study of weak interaction processes, the Wilson fermion scheme [1] appears to be the most attractive.

Today, parallel computers have matured and provide sufficient compute power to drive simulations of full QCD with Wilson fermions to the level of 100 and more Teraflops-hours. This performance allows for sufficient sampling of QCD vacuum configurations on reasonably sized lattices. As a consequence there is topical interest to *(i)* pioneer new algorithms for generating QCD configurations on one hand [2, 3, 4] and *(ii)* achieve progress in improving HMC (as the method of choice) on the other hand[1].

Meanwhile, the stabilized bi-conjugate gradient algorithm (BiCGstab) [5] has been established as an efficient inverter in lattice QCD applications [6, 7, 8, 9] since it requires less iterations and less computing time as compared to the minimal residual algorithm or CG on the normal equations: improvements of about 50 % (with respect to the conjugate gradient) can be achieved in the regime of small quark masses. Numerical studies in Refs. [6, 7, 10] indicate that further progress is now to be expected through preconditioning.

In the past, various approaches for preconditioning the Dirac matrix M have been taken:

1. The first attempt started from the perception that in the weak coupling limit the quark propagator is diagonal in momentum space and hence Fourier transformation would act as a useful preconditioner. Such 'Fourier acceleration' [11], however, needs smoothness of fields and hence gauge fixing. This handicaps the computation with a substantial overhead which at the end of the day eats up most of the previous gain.

2. The state-of-the-art[2] preconditioning approach in lattice QCD[3] rests upon the

[1]It appears promising to find that this complementary strategy of tackling the fermion problem in numerical field theory joins both physicists and applied mathematicians.

[2]Non-Krylov-subspace based methods like multigrid as of today did not achieve a level of maturity to be competitive with inverters like CG in lattice gauge theory. This is due to the disordered nature of the gluonic field configurations that act as fluctuating coefficients within the discretized differential operator stencils.

[3]The transformation of M from natural into odd-even order might be looked upon as a particular SSOR preconditioning, see Section 3.

odd-even decomposition of the matrix M [12] that yields an efficiency gain by a factor of 2 - 3 when inverting M in the context of actual fermionic force and quenched propagator computations [13, 6]. It is crucial that this second preconditioning approach lends itself easily to parallelization.

3. A third and promising method has been evaluated by Oyanagi in the 80s [14]. He used the standard incomplete LU (ILU) factorization of the matrix M based on the natural, *globally lexicographic* ordering of the lattice points. For Wilson fermions with Wilson parameter $r = 1$, the ILU preconditioner is identical to the SSOR preconditioner with respect to that ordering.

   Ref. [14] found ILU preconditioning (for the conjugate residual method) to outperform the odd-even decomposition on vector machines, achieving a large gain factor (in terms of iteration numbers) over conventional unpreconditioned methods. As it stands, Oyanagi's method works satisfactorily on vector machines. However, on local memory or grid-oriented parallel computers, this preconditioner can hardly be implemented efficiently, since parallelism can only be achieved by working concurrently on lattice points lying on the same *diagonal hyper-plane*. Hockney [15, 16] reports that the run times on several parallel and vector computers actually degrade as compared to the odd-even preconditioning.

   Using the so-called 'Eisenstat-trick' (see Section 2), the ILU preconditioning can be implemented in a more efficient manner than in [14, 15][4].

In this paper, which is based on the results in [17], we intend to open the stage for use of general parallel SSOR preconditioning techniques in lattice QCD based on appropriate orderings of the lattice points. Our approach may be regarded as a generalization of the odd-even (or red-black) ordering to a multiple color layout [18] or, alternatively, as a localization of the globally lexicographic ordering. The parallel implementation will be carried out in combining two steps:

1. Partition the whole lattice into equally shaped sublattices and introduce a lexicographic ordering on each sublattice.
2. During the SSOR preconditioning step, in parallel sweep through all sublattices in lexicographic order.

In Section 2 we discuss the SSOR preconditioner in detail and we give efficient implementations for the SSOR preconditioned BiCGstab and MR methods involving the Eisenstat-trick. Section 3 explains how different orderings of the lattice points lead

[4]Incorporating the Eisenstat trick into the Oyanagi ILU preconditioner could thus lead to a revision of some of the conclusions from [15], so that a more detailed numerical study is advisable.

to variants of the SSOR preconditioning procedure. In particular, this section introduces the locally lexicographic ordering through which we obtain the improvements over odd-even preconditioning.

In Section 4, the parallel implementation in conjunction with sub-blocking is discussed.

Section 5 presents the behavior of our new preconditioning method for Hybrid Monte Carlo applications and its performance on APE100/Quadrics parallel computers. On lattices of size $8^3 \times 16$ we test the efficiency of the method numerically where we find improvement factors of about 2 in terms of iteration numbers compared to the standard odd-even method. We discuss the effects of granularity and further report about first experiences with the preconditioner in a large scale simulation using a 512-node APE100/Quadrics on a $24^3 \times 40$ lattice, in the chirally sensible region.

2 SSOR Preconditioning

When preconditioning (1), we take two non-singular matrices V_1 and V_2 which act as a left and a right preconditioner, respectively, i.e. we consider the new system

$$V_1^{-1} M V_2^{-1} \tilde{x} = \tilde{\phi}, \text{ where } \tilde{\phi} = V_1^{-1}\phi, \ \tilde{x} = V_2 x. \tag{2}$$

(In the following, preconditioned quantities will be denoted by tildes.) We could now apply BiCGstab, as given in Algorithm 1 for the original system, (or any other Krylov subspace method) directly to (2), replacing each occurrence of M and ϕ by $V_1^{-1}MV_2^{-1}$ and $\tilde{\phi}$, resp. However, this would yield the preconditioned iterates $\tilde{x}^k$ together with preconditioned residuals. Therefore, one usually reformulates the algorithm incorporating an implicit back-transformation to the unpreconditioned quantities. The resulting algorithm is similar to Algorithm 1, requiring two additional systems with matrix $V = V_1 V_2$ and two systems with matrix V_1 to be solved in each iterative step (see [5]).

The purpose of preconditioning is to reduce the number of iterations and the computing time necessary to achieve a given accuracy. This means that V has to be a sufficiently good approximation to the inverse of M (thus decreasing the number of iterations) while solving systems with V, V_1 should be sufficiently cheap (since these solves represent the overhead of the preconditioned upon the basic method).

In the present work we are only interested in the SSOR (or, more precisely, symmetric Gauß-Seidel) preconditioner. Consider the decomposition of M into its diagonal, strictly lower and strictly upper triangular parts, i.e.

$$M = I - L - U$$

{ initialization }
choose x_0, set $r_0 = \phi - Mx_0, \hat{r}_0 = r_0$
set $\rho_0 = \rho_1 = \alpha_0 = \omega_0 = 1$
set $v_0 = p_0 = 0$
{ iteration }
for $i = 1, 2, \ldots$
$\rho_i = \hat{r}_0^\dagger r_{i-1}$
$\gamma_i = (\rho_i/\rho_{i-1})(\alpha_{i-1}/\omega_{i-1})$
$p_i = r_{i-1} + \gamma_i(p_{i-1} - \omega_{i-1}v_{i-1})$
$v_i = Mp_i$
$\alpha_i = \rho_i/\hat{r}_0^\dagger v_i$
$s_i = r_{i-1} - \alpha_i v_i$
$t_i = Ms_i$
$\omega_i = t_i^\dagger s_i/t_i^\dagger t_i$
$x_i = x_{i-1} + \omega_i s_i + \alpha_i p_i$
$r_i = s_i - \omega_i t_i$

Algorithm 1: BiCGstab method

with L, U strictly lower and upper triangular matrices, respectively. Then the SSOR preconditioner is given by

$$V_1 = I - L, \;\; V_2 = I - U. \tag{3}$$

For the SSOR preconditioner we have $V_1 + V_2 - M = I$. This important relation can be exploited through the 'Eisenstat-trick' [19], because we now have $V^{-1}MV_2^{-1} = V_2^{-1} + V_1^{-1}(I - V_2^{-1})$, so that the matrix vector product $w = V_1^{-1}MV_2^{-1}r$ can economically be computed via

$$v = V_2^{-1}r, \;\; u = V_1^{-1}(r - v), \;\; w = v + u.$$

In this manner we get the standard algorithmic formulation of the SSOR preconditioned BiCGstab method, stated as Algorithm 2.

The Eisenstat trick is not restricted to the BiCGstab method but can be applied in any Krylov subspace method. As another example, we state the SSOR preconditioned minimal residual (MR) method as Algorithm 3.

Note that these algorithms use the preconditioned residuals $\tilde{r}_i$ which are related to the unpreconditioned residuals $r_i = \phi - Mx_i$ via

$$r_i = (I - L)\tilde{r}_i.$$

To save computational costs, a stopping criterion for Algorithm 2 will usually be based on $\tilde{r}_i$. Upon successful stopping, one can then compute r_i and test for convergence

{ initialization }
choose x_0, set $r_0 = \phi - Mx^0$
solve $(I-L)\tilde{r}_0 = r_0$ to get $\tilde{r}_0$ { forward solve }
$\tilde{\hat{r}}_0 = \tilde{r}_0$
set $\rho_0 = \rho_1 = \alpha_0 = \omega_0 = 1$
set $\tilde{v}_0 = \tilde{p}_0 = 0$
{ iteration }
for $i = 1, 2, \ldots$
$\rho_i = \tilde{\hat{r}}_0^\dagger \tilde{r}_{i-1}$
$\gamma_i = (\rho_i/\rho_{i-1})(\alpha_{i-1}/\omega_{i-1})$
$\tilde{p}_i = \tilde{r}_{i-1} + \gamma_i(\tilde{p}_{i-1} - \omega_{i-1}\tilde{v}_{i-1})$
solve $(I-U)z_i = \tilde{p}_i$ to get z_i { backward solve }
solve $(I-L)\tilde{w}_i = \tilde{p}_i - z_i$ to get $\tilde{w}_i$ { forward solve }
$\tilde{v}_i = z_i + \tilde{w}_i$
$\alpha_i = \rho_i/\tilde{\hat{r}}_0^\dagger \tilde{v}_i$
$\tilde{s}_i = \tilde{r}_{i-1} - \alpha_i \tilde{v}_i$
solve $(I-U)y_i = \tilde{s}_i$ to get y_i { backward solve }
solve $(I-L)\tilde{u}_i = s_i - y_i$ to get $\tilde{u}_i$ { forward solve }
$\tilde{t}_i = y_i + \tilde{u}_i$
$\omega_i = \tilde{t}_i^\dagger \tilde{s}_i / \tilde{t}_i^\dagger \tilde{t}_i$
$x_i = x_{i-1} + \omega_i y_i + \alpha_i z_i$
$\tilde{r}_i = \tilde{s}_i - \omega_i \tilde{t}_i$

Algorithm 2: SSOR preconditioned BiCGstab

using r_i. If the solution is not yet accurate enough, one continues the iteration with a

{ initialization }
choose x_0, set $r_0 = \phi - Mx^0$
solve $(I-L)\tilde{r}_0 = r_0$ to get $\tilde{r}_0$ { forward solve }
{ iteration }
for $i = 0, 1, \ldots$
solve $(I-U)w_i = \tilde{r}_i$ to get w_i { backward solve }
solve $(I-L)v_i = \tilde{r}_i - w_i$ to get v_i { forward solve }
$\tilde{p}_i = w_i + v_i$
$\alpha_i = \tilde{r}_i^\dagger \tilde{p}_i / \tilde{p}_i^\dagger \tilde{p}_i$
$x_{i+1} = x_i + \alpha_i w_i$
$\tilde{r}_{i+1} = \tilde{r}_i - \alpha_i \tilde{p}_i$

Algorithm 3: SSOR preconditioned minimal residual stronger stopping criterion. In our numerical experiments to be reported in Section 5 it turned out that such additional steps were never necessary.

Also note that multiplications with M are completely absent in this formulation, the only matrix operations being solves with each of the matrices $I - L$ and $I - U$. Since these matrices are triangular, the solves can be done directly via forward or backward substitution, respectively. For example, denoting the components of a vector y by $(y)_i$ and the entries of L by $(L)_{ij}$ and similarly for U, the forward solve $(I - L)y = p$ and backward solve $(I - U)z = p$ become simply

for $i = 1, \ldots, n$

$$(y)_i = (p)_i + \sum_{j=1}^{i-1} (L)_{ij}(y)_j$$

forward solve

for $i = n, \ldots, 1$

$$(z)_i = (p)_i + \sum_{j=i+1}^{n} (U)_{ij}(z)_j$$

backward solve

Due to the sparsity pattern of M, most of the entries in L and U are zero so that only a few j actually contribute to the sums over j. We will come back to this point in more detail in the next section.

It is important to note that for i fixed, the number of non-zero entries of L and U involved when updating y_i in the forward together with z_i in the backward solve is just the number of non-zero entries of the i-th row of the matrix M. Therefore, in terms of computational cost, a forward followed by a backward solve is quite precisely as expensive as a multiplication with M (which is required in the unpreconditioned method). Hence, due to the Eisenstat-trick, in terms of matrix vector operations, each step of the SSOR preconditioned BiCGstab (or MR) method requires the same work as in the unpreconditioned method.

We finally note that the 'classical' symmetric Gauß-Seidel iteration

$$(I - L)x^{k+1/2} = Ux^k + \phi, \;\; (I - U)x^{k+1} = Lx^{k+1/2} + \phi, \;\; k = 0, 1, \ldots$$

represents a sequence of alternating forward and backward solves. This explains the terminology SSOR preconditioner where SSOR stands for symmetric successive over-relaxation, the over-relaxed variant of the symmetric Gauß-Seidel method. See [20], e.g.

3 Orderings

When writing down equation (1) with Wilson fermion matrix M[5]

$$\begin{aligned} M_{x,y} &= \delta_{x,y} - \kappa \sum_{\mu=1}^{4} (1-\gamma_\mu) U_\mu(x)\,\delta_{x,y-\mu} + (1+\gamma_\mu) U_\mu^\dagger(x-\mu)\,\delta_{x,y+\mu} \\ &= \delta_{x,y} - \kappa \sum_{\mu=1}^{4} m_{x,y}^{-}\,\delta_{x,y-\mu} + m_{x,y}^{+}\,\delta_{x,y+\mu}. \end{aligned} \tag{4}$$

we have the freedom to choose any ordering scheme for the lattice points x. Different orderings yield different matrices M, however, which are permutationally similar to each other, i.e. one matrix can be retrieved from the other via the transformation $M \to P^\dagger M P$ with a permutation matrix P. In general, $P^\dagger L P$ and $P^\dagger U P$ will *not* represent the strictly lower or strictly upper triangular part of $P^\dagger M P$. Consequently, the SSOR preconditioned matrix for $P^\dagger M P$ will *not* be permutationally similar to that of M, so that the quality of the SSOR preconditioner will usually depend on the ordering scheme chosen. One purpose of this paper is to explain that the odd-even preconditioning and the Oyanagi preconditioning can both be regarded as the SSOR preconditioning belonging to particular orderings of the grid points[6]. We will then introduce a new 'locally lexicographic' ordering which is adapted for parallel computation and for which the SSOR preconditioned system can be solved more rapidly than with odd-even preconditioning with respect to both, the number of iterations as well as actual run time on particular parallel computers.

Consider an arbitrary numbering (ordering) of the lattice points. For a given grid point x, the corresponding row in the matrix L or U contains exactly the coupling coefficients of those nearest neighbors of x which have been numbered before or after x, resp. Therefore, a generic formulation of the forward solve for this ordering is given by Algorithm 4. The backward solves are done similarly, now running through the grid points in *reverse* order and taking those grid points $x \pm \mu$ which were numbered *after* (instead of *before*) x. Due to this analogy, we will restrict our discussion to the forward solves in the sequel.

Odd-even ordering. As a first specific example, let us consider the familiar odd-even ordering where all odd lattice points are numbered before the even ones. From our generic formulation in Algorithm 4 we then get Algorithm 5 for the forward solve.

In traditional QCD computations, the odd-even preconditioning is *not* implemented by

[5] The 4 matrices γ_μ are 4×4 Dirac matrices, and the U_μ are 3×3 SU(3) matrices that represent the gluonic degrees of freedom on the lattice.

[6] It is shown in [17] that the SSOR preconditioner for the Wilson fermion matrix (with Wilson parameter $r = 1$) is identical to the ILU preconditioner for *any* ordering of the lattice points

for all grid points x in the given order
{ update y_x }
$y_x = p_x$
for $\mu = 1, \ldots, 4$
if $x - \mu$ was numbered before x then
$y_x = y_x + \kappa \cdot m^{+}_{x,x-\mu} y_{x-\mu}$
for $\mu = 1, \ldots, 4$
if $x + \mu$ was numbered before x then
$y_x = y_x + \kappa \cdot m^{-}_{x,x+\mu} y_{x+\mu}$

Algorithm 4: Generic forward solve

for all odd grid points x
$y_x = p_x$
for all even grid points x
{ update y_x }

$$y_x = p_x + \kappa \left(\sum_{\mu=1}^{4} m^{+}_{x,x-\mu}\, y_{x-\mu} + \sum_{\mu=1}^{4} m^{-}_{x,x+\mu}\, y_{x+\mu} \right)$$

Algorithm 5: Odd-even forward solve

using the above formulation of the forward (and backward) solve in Algorithm 2. This is due to the fact that—very exceptionally—for this particular ordering the inverses of $I - L$ and $I - U$ can be determined directly: With the odd-even ordering, the matrix M has the form

$$M = \begin{pmatrix} \mathbf{1} & -\kappa D_{oe} \\ -\kappa D_{eo} & \mathbf{1} \end{pmatrix} \tag{5}$$

so that

$$I - L = \begin{pmatrix} \mathbf{1} & 0 \\ -\kappa D_{eo} & \mathbf{1} \end{pmatrix} \text{ with } (I-L)^{-1} = \begin{pmatrix} \mathbf{1} & 0 \\ \kappa D_{eo} & \mathbf{1} \end{pmatrix}$$

and

$$I - U = \begin{pmatrix} \mathbf{1} & -\kappa D_{oe} \\ 0 & \mathbf{1} \end{pmatrix} \text{ with } (I-U)^{-1} = \begin{pmatrix} \mathbf{1} & \kappa D_{oe} \\ 0 & \mathbf{1} \end{pmatrix}.$$

Hence

$$(I-L)^{-1} M (I-U)^{-1} = \begin{pmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{1} - \kappa^2 D_{eo} D_{oe} \end{pmatrix},$$

where $\mathbf{1} - \kappa^2 D_{eo} D_{oe}$ is called the matrix of the odd-even reduced system. Very exceptionally again, in this particular case, the preconditioned matrix $(I-L)^{-1}M(I-U)^{-1}$ thus has such a simple structure that it is possible to apply Algorithm 1 directly, using the preconditioned matrix explicitly. Since the matrix $(I-L)^{-1}M(I-U)^{-1}$ is 2×2 block diagonal with the first diagonal block being the identity, the computations for the second diagonal block are completely decoupled from those of the first block and they are identical to those which are performed when applying the algorithm for the odd-even reduced system directly. This is precisely what is done in traditional odd-even preconditioning. So traditional odd-even preconditioning is equivalent to SSOR preconditioning for the odd-even ordered system.

So far, odd-even preconditioning is generally considered as the only successful preconditioner in a parallel computing environment. For typical, realistic configurations, it gains a factor 2-3 in the numbers of iterations and also in computing time as compared to solving the unpreconditioned system.

Globally lexicographic ordering. Assume now that M is given with respect to the natural (lexicographic) ordering of the lattice points. This means that grid point $x=(i_1,i_2,i_3,i_4)$ is numbered before $x'=(i_1',i_2',i_3',i_4')$ if and only if $(i_4<i_4')$ or ($i_4=i_4'$ and $i_3<i_3'$) or ($i_4=i_4', i_3=i_3'$ and $i_2<i_2'$) or ($i_4=i_4', i_3=i_3', i_2=i_2'$ and $i_1<i_1'$). The corresponding forward solve is given as Algorithm 6.

for $i_4=1,\ldots,n_4$
    for $i_3=1,\ldots,n_3$
        for $i_2=1,\ldots,n_2$
            for $i_1=1,\ldots,n_1$
                $x:=(i_1,i_2,i_3,i_4)$
                { update y_x }
                $y_x=p_x$
                for $\mu=1,\ldots,4$
                    if $i_\mu>1$ then $y_x=y_x+\kappa\, m^+_{x,x-\mu}y_{x-\mu}$

Algorithm 6: Lexicographic forward solve

The SSOR preconditioning for the lexicographic ordering, applied to the MR and other conjugate residual methods, was considered by Oyanagi [14]. He showed that it yields a further improvement over odd-even preconditioning as far as the number of iterations is concerned. However, its parallel implementation is more difficult and less efficient, since only grid points lying on the same *diagonal hyper-plane* can be worked upon in parallel in the forward and backward solves. Hockney [16] reports that on the 592 processor ACPMAPS at Fermi-Lab, the run time (without the Eisenstat trick) actually degrades as compared to odd-even preconditioning. See also [15].

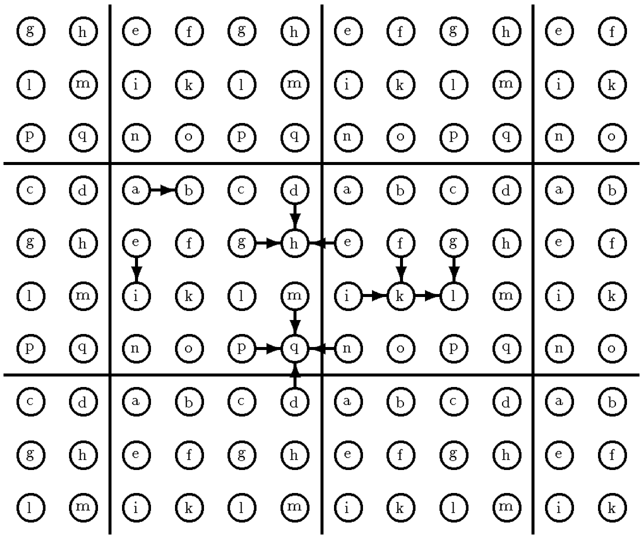

Figure 1: Locally lexicographic ordering and forward solve in 2 dimensions

Locally lexicographic ordering. Unlike the lexicographical and the odd-even ordering, the ordering we propose now is adapted to the parallel computer used to solve equation (1). We assume that the processors of the parallel computer are connected as a $p_1 \times p_2 \times p_3 \times p_4$ 4-dimensional grid. Note that this includes lower dimensional grids by setting some of the p_i to 1. The total number of processors is $p = p_1p_2p_3p_4$. The space-time lattice can be matched to the processor grid in an obvious natural manner, producing a local lattice of size $n_1^{loc} \times n_2^{loc} \times n_3^{loc} \times n_4^{loc}$ with $n_i^{loc} = n_i/p_i$ on each processor. Here, for simplicity, we assume that each p_i divides n_i and that we have $n_i^{loc} \geq 2$ for $i = 1, \ldots, 4$.

Let us partition the whole lattice into $n^{loc} = n_1^{loc}n_2^{loc}n_3^{loc}n_4^{loc}$ groups. Each group corresponds to a fixed position of the local grid and contains all grid points appearing at this position within their respective local grid. Associating a color with each of the groups, we can interpret this process as a coloring of the lattice points. For the 2-dimensional case, such a coloring is depicted in Figure 1 (with $n_1^{loc} = n_2^{loc} = 4$), where the 16 colors are denoted by the letters $a - q$.

We now consider an ordering where all points of color b are ordered after all those of color a if on the local grids the position belonging to color a is lexicographically less than that of color b. In Figure 1, this corresponds to the alphabetic ordering of the colors $a - q$. Such an ordering is termed *locally lexicographic.* From now on we will use the prefix '*ll*-' in expressions like '*ll*-first', '*ll*-ordering' to make clear that they refer to the locally lexicographic ordering.

Since we assumed $n_i^{loc} \geq 2$ for all i, all nearest neighbors of a given grid point have colors different from that point. This implies that when performing the forward and backward solves in Algorithm 2, grid points having the same color can be worked upon in parallel, thus yielding an optimal parallelism of p, the number of processors. A formulation of the ll-forward solve is given as Algorithm 7. Here, we use '$\leq_{ll}$' as a symbol for 'll-less than'.

for all colors in lexicographic order
  for all processors
    $x :=$ grid point of that color on that processor
    { update y_x }

$$y_x = p_x + \kappa \left(\sum_{\mu,\, x-\mu \leq_{ll} x} m^{+}_{x,x-\mu} y_{x-\mu} + \sum_{\mu,\, x+\mu \leq_{ll} x} m^{-}_{x,x+\mu} y_{x+\mu} \right)$$

Algorithm 7: ll-forward solve

For grid points lying in the 'interior' of each local grid, we have $x - \mu \leq_{ll} x \leq_{ll} x + \mu$ for $\mu = 1, \ldots, 4$. The update thus becomes

$$y_x = p_x + \kappa \left(\sum_{\mu=1}^{4} m^{+}_{x,x-\mu} y_{x-\mu} \right),$$

whereas on the 'local boundaries' we will have between 0 (for the ll-first point) and 8 (for the ll-last point) summands to add to p_x. The second sum in the update formula then contains those grid points which belong to neighboring processors. The arrows in Figure 1 illustrate this situation for the 2-dimensional analogue, where 2 grid points contribute to the update at locally interior points (colors f, g, k, l) and 0 to 4 on the local boundaries. (An arrow from point $x \pm \mu$ to x means that $x \pm \mu$ appears in the sum for updating point x.)

With the lexicographic ordering on the *whole* lattice, a forward solve followed by a backward solve has the effect that the information at any grid point will have spread out to all other grid points. This yields a heuristic justification of why this ordering is superior (as far as convergence properties are concerned) to the odd-even ordering, where the information is passed to the (second-) nearest neighbors only. The ll-ordering can be regarded as a compromise between these two extreme cases, spreading information to all points within a local grid. The ll-forward and ll-backward solves parallelize efficiently and better than with the lexicographic ordering on the whole grid. The parallelism achieved is p and thus less than with the odd-even ordering, but it is optimal since we have p processors. If we change the number of processors, the ll-ordering, and consequently the properties of the corresponding SSOR preconditioner will change, too. Heuristically, we expect the convergence properties to degrade as the size of the local grid becomes smaller but to always remain better than with the odd-even preconditioner (as long as $n_i^{loc} \geq 2$ for $i = 1, \ldots, 4$).

4 Parallel Implementation

In this section we give a detailed discussion on the realization of the ll-forward solve on a parallel computer. In order to render all necessary communication transparent we assume a model parallel computer where communication is done via explicit message passing between processors.

As in the previous section, we assume that the processors are connected as a 4-dimensional (or lower dimensional) grid and that the space-time lattice is mapped onto that grid in the obvious manner, yielding a local subgrid on each processor. The local coordinates of a grid point x within a subgrid are denoted by the 4-tuple $x = (i_1, i_2, i_3, i_4)$. For processor π of the processor grid we denote its neighbors in dimension μ by $\pi \pm \mu$. Incorporating all necessary data transmission into Algorithm 7 via message passing statements—sending messages as soon as possible while receiving them as late as possible—we end up with Algorithm 8 which formulates the ll-forward solve for processor π.

```
for i_4 = 1,...,n_4^loc
   for i_3 = 1,...,n_3^loc
      for i_2 = 1,...,n_2^loc
         for i_1 = 1,...,n_1^loc
            x := (i_1, i_2, i_3, i_4)
            { update y_x }
            y_x = p_x
            for μ = 1,...,4
               if i_μ > 1 then y_x = y_x + κ m^+_{x,x-μ} y_{x-μ}
            for μ = 1,...,4
               if i_μ = 1 then
                  send y_x to processor π - μ
               if i_μ = n_μ^loc then
                  receive y_{x+μ} from processor π + μ
                  y_x = y_x + κ m^-_{x,x+μ} y_{x+μ}
```

Algorithm 8: ll-forward solve on processor π

Note that if processor π issues a 'send' to transfer y_x to processor $\pi - \mu$ because of $i_\mu = 1$, the corresponding 'receive' (for $i_\mu = n_\mu^{loc}$) in processor $\pi - \mu$ reads this data as $y_{x'}$ with $x' = x + \mu$. If $\mu = 1$, exactly $n_1^{loc} - 2$ updates are done between a matching send/receive pair. If $\mu = 2$ this number increases to $n_1^{loc}(n_2^{loc} - 2)$, for $\mu = 3$ it becomes $n_1^{loc}n_2^{loc}(n_3^{loc} - 2)$ and for $\mu = 4$ it is $n_1^{loc}n_2^{loc}n_3^{loc}(n_4^{loc} - 2)$. Therefore, if messages can be handled in parallel to the computation, the message passing can be hidden behind

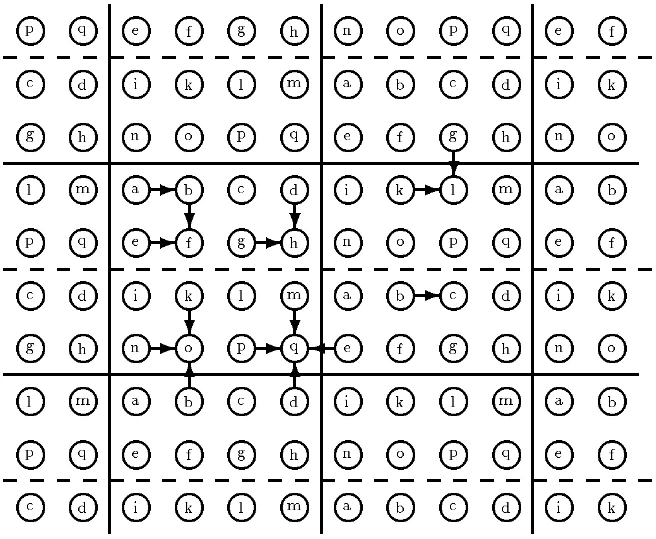

Figure 2: $ll \times oe$ ordering and forward solve, 2 dimensions

the computation so that the communication overhead will be small. Also note that if $p_\mu = 1$ the send/receive pairs for direction μ have to be dropped in Algorithm 8.

Algorithm 8 has to communicate the local boundary values all one by one. This is inherent in the ll-ordering and can become a disadvantage on machines which have a large start-up time for communication, irrespective of the message length. In these cases, it is preferable to issue only a small number of (large) messages. Then the following outer product' of the ll-ordering with an odd-even ordering can be used: Divide each local grid into two halves G_1, G_2 by bisecting perpendicularly to direction 4 (say), where n_4^{loc} is supposed to be a multiple of 2. Identify odd and even processors on the processor grid, neglecting the 4-th coordinate, i.e., processor $\pi = (j_1, j_2, j_3, j_4)$ is odd if and only if $j_1 + j_2 + j_3$ is odd. On each even processor, we first take a locally lexicographic ordering on G_1 and then on G_2, on odd processors we reverse the role of G_1 and G_2. The forward solve with respect to this ordering consists of two half steps. First, the odd processors update G_1 while the even processors work on G_2. No communication is necessary there. Then the updated values on the local boundaries have to be communicated (requiring a total of 7 large messages, one for direction 4 and two for each other direction). Now the second half step can be performed, where odd processors update G_2 and even processors update G_1. An illustration of this $oe \times ll$–ordering for the 2-dimensional case is given in Figure 2.

Implementing the ll-SSOR preconditioning requires to distinguish between grid sites along a given direction. Assume in a forward solve that a grid point is located at the 'left' boundary of the local lattice. This site is *not* updated with data from the left neighbor, and this fact has to be inquired by a conditional statement or a case statement. The second case is that a grid site is located at a 'right' boundary where the right neighbor has to be communicated from the neighbor sublattice (in this situation the neighbor processor). The third possibility is that the site is not a boundary grid point. This leads to 81 different cases a site can be associated with in four dimensions.

5 Results

Our numerical tests of the locally lexicographic SSOR preconditioner were performed on APE100/Quadrics machines, a SIMD parallel architecture optimized for fast floating point arithmetic on block data-structures like 3×3 SU(3) matrices or any other data structure that can be blocked similarly. Arithmetic on the Quadrics is compliant with the IEEE 754 single precision standard. We had access to a 32-node Quadrics Q4 at Wuppertal University and a 512-node Quadrics QH4 at the computer center of ENEA/Casaccia Rome.

In order to speed up the performance on the Quadrics it is favorable (as it is the case on most high speed parallel computers) to partition the code into code blocks *i.e.* sections that can be computed entirely from the registers without recourse to the local or remote memory, thus using the pipelining features of the processors efficiently. This requires to modify Algorithm 8 by moving all if-statements out of the four-fold nested loop over $i_1, \ldots, i_4$ resulting in a sequence of 81 nested loops, one for each possible combination of the three cases $i_\mu = 1, 1 < i_\mu < n_\mu^{loc}, i_\mu = n_\mu^{loc}$ for $\mu = 1, \ldots, 4$.

Evidence of improvement. In Fig. 3 first evidence of improvement by *ll*-SSOR preconditioning of BiCGstab is presented. Shown are three different iteration numbers of a valence mass "trajectory" for a given thermalized full-QCD configuration at $\beta = 5.6$ and $\kappa_{\text{sea}} = 0.156$ corresponding to an intermediate pion mass. The standard BiCGstab solution of (1) is compared with the odd-even preconditioned method and the new locally lexicographic ordering scheme. The measurements are taken on a lattice of size $8^3 \times 16$, hence the computational problem being of granularity $g = N/p = 256$ on the Quadrics Q4 with 32 nodes. In this exploratory implementation, as discussed in the previous sections, the sublattice administrated by a given processor is used as the local lattice in the *ll*-SSOR scheme. At this place, the convergence stopping criteria,

$$\text{stop iteration if} \qquad \frac{||Mx - \phi||_2}{||\phi||_2} < \epsilon, \tag{6}$$

are applied on the three different residuals used in the three variant algorithms, respectively, with equal numerical value $\epsilon = 10^{-8}$. We verified that the 'true' residuals

$Mx - \phi$, calculated explicitly from the solution x, eventually were smaller for the preconditioned variants of the BiCGstab solver. The gain in iterations is more than a factor of 2 for $\kappa_{\text{valence}} = \kappa_{\text{sea}}$ as is the case within the entire valence κ range considered. Compared to the unpreconditioned BiCGstab the gain is even a factor of 4.

The improvement in terms of cpu-time, on the Q4, is smaller than the gain in iteration numbers since the *ll*-SSOR preconditioning leads to certain breaks in code blocks due to the 81 different cases mentioned above. Thus, pipelined arithmetic is not exploited as well as in the other cases. For the given choice of parameters we find an improvement of about a factor of about 1.5 compared to standard *oe*-preconditioning throughout the valence mass range considered.

The convergence behavior of the three methods is demonstrated in Fig. 4 on a typical configuration at the same parameter values for $\kappa_{\text{sea}} = \kappa_{\text{val}} = 0.156$. In addition to the acceleration, the convergence behavior turns out to be much smoother for the *ll*-preconditioned BiCGstab than in the unpreconditioned case. We have plotted both the accumulated and the 'true' residual. The latter approaches a horizontal line for $\|r\|_2 \approx 10^{-6}$ which indicates the limits of single precision accuracy. For *ll*-preconditioning, however, the final residual that can be achieved appears to be somewhat more accurate than for pure BiCGstab.

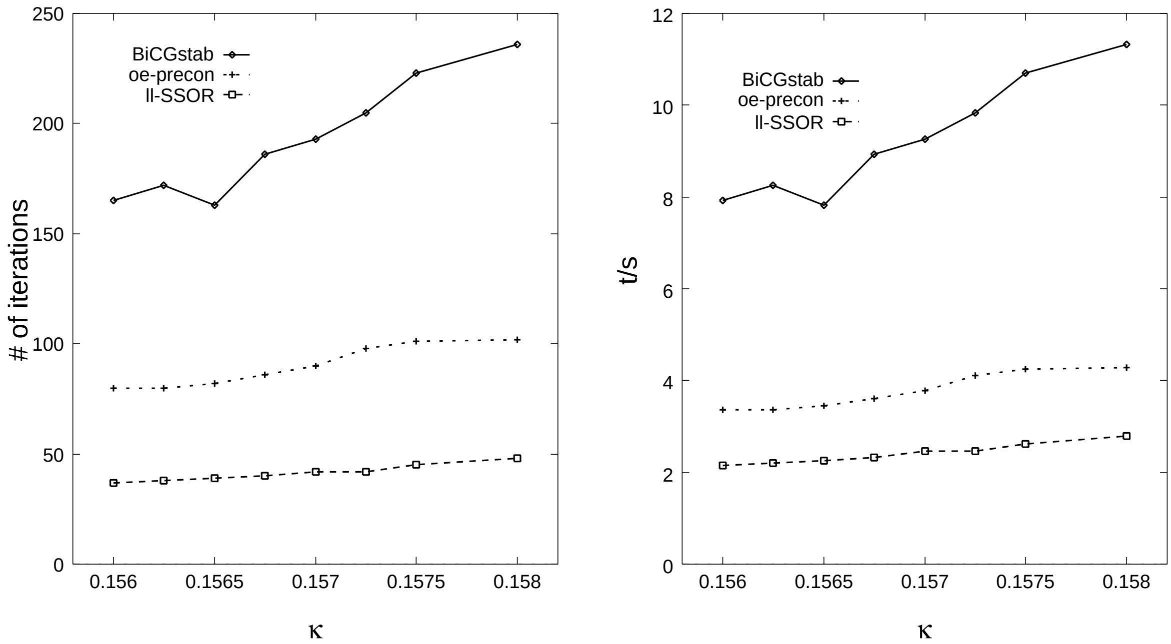

Figure 3: Iteration numbers of pure BiCGstab •, the *oe*- + and the *ll*-preconditioned version □ for a series of valence masses on a $8^3 \times 16$ lattice. The second plot shows the cpu-time needed on a 32-node Quadrics Q4 at granularity $g = 256$.

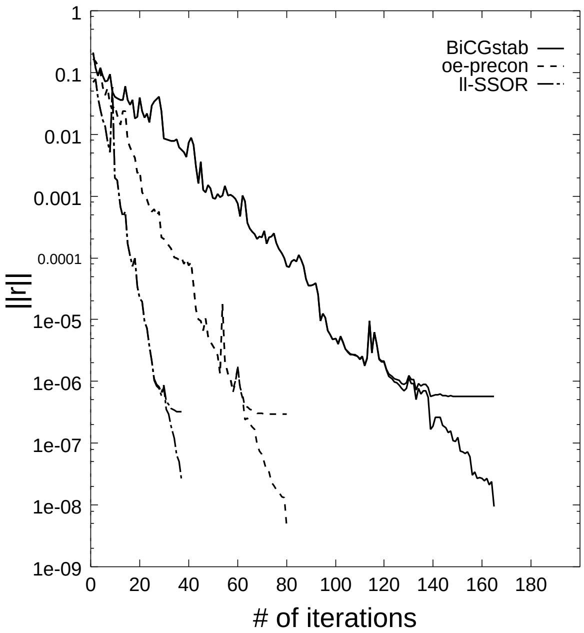

Figure 4: Convergence behavior of BiCGstab and its *oe*- and *ll*-preconditioned variants on an $8^3 \times 16$ lattice.

Local lattice size dependency. It is important to assess the dependency of the improvement due to *ll*-SSOR preconditioning on the size of the local lattice. As our Q4 is equipped with a fixed number of 32 processors, we emulated larger numbers of local lattices by sub-blocking the system on each processor. The smallest local lattice size we can investigate is a 2^4 lattice, apart from the unpreconditioned case (which we treat as having local lattice size 1) and the *oe*-realization of the solution of (1) (local lattice size 2).

The dependency of the number of iterations on the local lattice size is depicted in Fig. 5 on an equally sized lattice as used before and at equal physical parameters but with $\kappa_{\rm sea} = 0.1575$. As expected, smaller local lattices lead to a less efficient preconditioning. However, the number of iterations at first increases only slightly if we move from a local lattice size of 256 to 16. This behavior might of course be dependent on $\kappa_{\rm sea}$, hence the crossover to the *oe*-result (the diamond in Fig. 5) might occur at a larger local lattice size for $\kappa_{\rm sea}$ closer to κ_c.

Speeding up QCD computations. Next we discuss improvements due to the *ll*-SSOR preconditioning implemented in an actual Hybrid Monte Carlo simulation of full QCD with Wilson fermions. We depict in Fig. 6a a time series of iteration numbers that are each averages over a molecular dynamics trajectory of a length of 125 steps. The measurements are taken on a system of size $24^3 \times 40$ at $\beta = 5.6$ and $\kappa_{\rm sea} = 0.1575$.

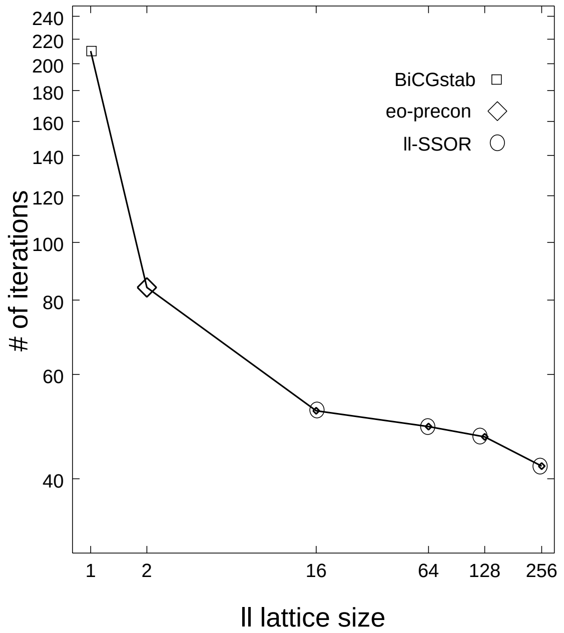

Figure 5: Dependence of iteration numbers on the ll-lattice size, the lattice volume is $8^3 \times 16$.

The first part of the curve has been generated using standard oe-preconditioning for the BiCGstab inverter. The sudden step in iterations is due to the fact that we switched the preconditioner to ll-SSOR at this location. The iteration numbers show a quite stable value in both branches of the curve and hence nicely demonstrate the stability of the preconditioning on different background field configurations within the canonical ensemble. In the actual large scale simulation, the ll-SSOR preconditioning scheme indeed gives an improvement of nearly a factor of 2 as to the iteration numbers for a quite light pion mass [21].

We remark that we have verified in our simulation that the educated guess procedure as advocated by Brower et al. [22] leads to an improvement that is *additive* to the gain as achieved by the use of ll-SSOR preconditioning.

The time behavior of the system on the 512-node APE100/Quadrics QH4 is depicted in Fig. 6b. We remark that on this machine topology with $8 \times 8 \times 8$ nodes on a 3-dimensional grid, we arrive at a local grid size of $3 \times 3 \times 24 \times 5$ for the $24^3 \times 40$ lattice. On a strict SIMD architecture without local addressing, as is the case on the Quadrics, oe-schemes are quite tricky to implement and lead to a substantial loss in performance in the range of 30 %. Therefore ll-SSOR can help a lot to overcome the SIMD odd-even limitations. The total improvement as to cpu-time is about a factor of 1.7.

Details on our implementation and the experience with ll-SSOR preconditioning in the context of large scale HMC simulations in full QCD will be published elsewhere [23].

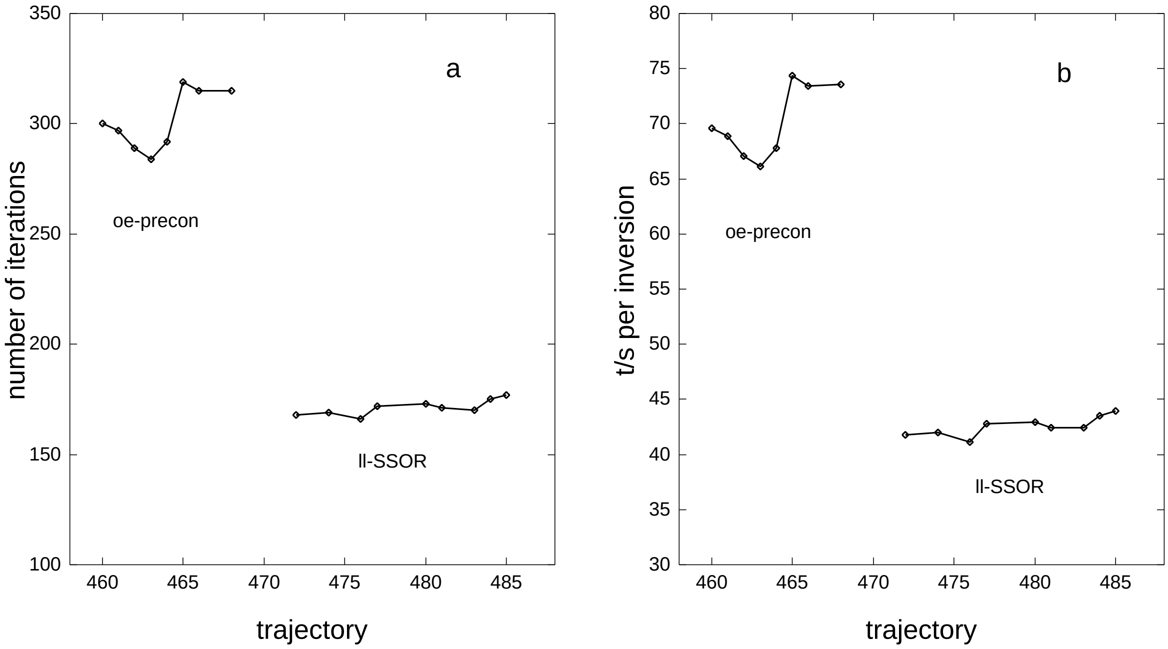

Figure 6: Section of the time series of iteration numbers together with cpu-times required on a QH4 from a full QCD HMC simulation on a $24^3 \times 40$ lattice. The graphics shows the location where we switched from the oe-representation of the fermionic matrix M to the ll-scheme.

6 Conclusion and Outlook

We have presented a new local grid point ordering scheme that allows to carry through efficient preconditioning of Krylov subspace solvers like BiCGstab in the context of computations of Greens functions for lattice quantum chromodynamics with Wilson fermions on parallel computers.

Using the Eisenstat-trick we can avoid the explicit multiplication by the fermionic matrix M and we remain with a forward and a backward solve to invert the upper and lower triangular parts of M.

In actual Hybrid Monte Carlo simulations with Wilson fermions at small pion masses and on large lattices, we verify an improvement of a factor of 2 in BiCGstab concerning iteration numbers as compared to the state-of-the-art odd-even preconditioned scheme. In this manner, our code is thus speeded up on a 512-node APE100/Quadrics by a factor of about 1.7 at $\beta = 5.6$ and $\kappa_{sea} = 0.1575$.

We are confident that the ll-SSOR scheme will help to drive future simulations with dynamical Wilson fermions deeper into the chiral region dominated by a small pion mass.

Acknowledgments

Part of the numerical test in this paper have been carried out on the Quadrics QH2 at ENEA/Casaccia/Roma. Th. L. and G. R. thank Profs. N. Cabibbo and A. Mathis for their kind hospitality and Dotts. R. Cannata and R. Guadagni for very valuable support. We are grateful to our friends from the SESAM group for important discussions and their patience while we were blocking the Quadrics Q4 at Wuppertal University and HLRZ/KFA-Jülich. We acknowledge helpful discussions with P. Fiebach from the department of Mathematics, University of Wuppertal and G. Hockney from Fermi-Lab, Batavia, IL.

The work of Thomas Lippert is supported by the Deutsche Forschungsgemeinschaft DFG under grant No. Schi 257/5-1.

References

1 Wilson, K. G.: in A. Zichichi (ed.), New Phenomena in Subnuclear Physics, Plenum, New York (1975).

2 Lüscher, M.: A new approach to the problem of dynamical quarks in the numerical simulation of lattice QCD, Nucl. Phys. B 418 637-648 (1994).

3 Alexandrou, C., Borrelli, A., de Forcrand, Ph, Galli, A., Jegerlehner, F.: Full QCD with the Lüscher Local Bosonic Action, preprint PSI-PR-95-09, IPS Report 95-14, UCY-PHY-95/5.

4 Jegerlehner, B.: Improvements of Lüscher Local Bosonic Fermion Algorithm, preprint MPI-PhT/95-119, HEP-LAT 9512001.

5 van der Vorst, H.: BiCGstab: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-symmetric Linear Systems, SIAM J. Sc. Stat. Comp. **13**, 631–644 (1992).

6 Frommer, A., Hannemann, V., Nöckel, B., Lippert, Th. and Schilling, K.: Accelerating Wilson Fermion Matrix Inversions by Means of the Stabilized Bi-conjugate Gradient Algorithm, Int. J. of Mod. Phys. C Vol. 5 No. 6, 1073–1088 (1994).

7 Borici, A. and de Forcrand, P.: Fast Krylov Space Methods for Calculation of Quark Propagators, IPS Research Report No. 94-03, ETH Zürich (1994).

8 Schilling, K., Frommer, A., Hannemann, V., Lippert, Th. and Nöckel, B.: Tuning up an Oldtimer: Hybrid Monte Carlo with Wilson Fermions, Nucl. Phys. B (Proc. Suppl.) 42 894-896 (1995).

9 de Forcrand, P.: Review talk given at Lattice '95, Melbourne 1995, to appear.

10 Frommer, A.: Talk given at Workshop 'Grosse Gleichungssysteme auf Parallelrechnern', University of Wuppertal, Oct. 6, 1994.

11 Katz, G., Batrouni, G., Davies, C., Kronfeld, A., Lepage, P., Svetitsky, B., Wilson, K.: Phys. Rev, D 37 1589 (1988).

12 DeGrand, T. and Rossi, P., Comp. Phys. Comm. 60 211 (1990).

13 Gupta, R. et al.: QCD with dynamical Wilson fermions, Phys. Rev. D 40 2072 (1989).

14 Oyanagi, Y.: An Incomplete LDU Decomposition of Lattice Fermions and its Application to Conjugate Residual Methods, Comp. Phys. Comm. 42, 333 (1986).

15 Hockney, G.: Nuclear Physics B (Proc. Suppl.) 17 301 (1990).

16 Hockney, G.: private communications.

17 Fischer, S.: Parallele SOR/SSOR-Präkonditionierer für die Inversion der vollen Wilson-Fermion-Matrix durch das BiCGstab-Verfahren auf der Quadrics, master's thesis, Department of Mathematics, University of Wuppertal (1996).

18 Block, U., Frommer, A., Mayer, G.: Block Coloring Schemes for the SOR Method on Local Memory Parallel Computers, Parallel Computing 14 61 (1990).

19 Eisenstat, S.: Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods, SIAM J. Sci. Stat. Comput. 2 1 (1981).

20 Ortega, J.: Introduction to Vector and Parallel Solution of Linear Systems, Plenum (1990)

21 For a discussion of the physical parameters as chosen in the SESAM project see: SESAM-collaboration—Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Luo, X., Ritzenhöfer, G., Schilling, K., and Siegert, G.: QCD with dynamical Wilson fermions—first results from SESAM, talk given at lattice '95, to appear in Nucl. Phys. B (Proc. Suppl.).

22 Brower, R.C., Ivanenko, T., Levi, A.R., Orginos, K. N.: Chronological Inversion Method for the Dirac Matrix in Hybrid Monte Carlo, preprint BU-CCS-950601 and hep-lat/9509012.

23 SESAM-collaboration—Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Ritzenhöfer, G., Schilling, K., and Siegert, G.: Toolkit for HMC practitioners, technical report, HLRZ Jülich, in preparation.