JOURNAL OF COMPUTATIONAL PHYSICS 70, 500–520 (1987)

Algorithms for Calculating Quark Propagators on Large Lattices

C. B. CHALMERS, R. D. KENWAY, AND D. ROWETH

Department of Physics, University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland

Received April 11, 1986; revised September 12, 1986

We describe methods for solving large sparse systems of linear equations on computers with limited fast memory, or high ratios of processor speed to bandwidth between main and fast memory. Our algorithms are designed for calculating quark propagators, columns of the inverse of the fermion matrix in Lattice Quantum Chromodynamics, but are more generally applicable. We compare their rates of convergence and the balance between CPU time and I/O overhead. We present a block-iterative algorithm which when implemented on the DAP is 5 times as efficient as the Conjugate Gradient Method for $16^3 \times 24$ lattices (complex linear systems of size approximately 3×10^5). © 1987 Academic Press, Inc.

1. INTRODUCTION

In this paper we describe and compare a variety of algorithms for computing columns of the inverse of a large sparse matrix resulting from the discretisation of the partial differential equation for the quark propagator in lattice gauge theory. This step is the dominant part of the calculation of hadron masses in lattice Quantum Chromodynamics (QCD). The mass estimates that we have obtained will be described elsewhere [1]. The work was performed on an ICL Distributed Array Processor (DAP) [2]. This is a SIMD square array of 4096 bit-serial processing elements, each with 4K bits of memory and nearest-neighbour connections, operating in parallel. Algorithms have to be modified to take account of the hardware features of this machine, but the following discussion is general and applies to most memory-limited implementations. An implementation of an algorithm becomes memory-limited when the processors require data to be moved from slow to fast memory at a higher rate than the machine can sustain. This problem arises frequently in algorithms requiring repeated matrix-vector products. Our slow memory is the disks of the mainframe hosting the DAP, and data rates between these disks and the main memory are low. Similar problems can arise on supercomputers with very high CPU speeds but only small amounts of fast memory on the processors.

The systems we want to solve are larger than can be contained in fast memory and must be partitioned into blocks of manageable size. These blocks must be repeatedly paged into fast memory while computation is proceeding. Thus we are looking for algorithms with balanced computation and I/O requirements. Many of

0021-9991/87 $3.00

the standard algorithms for solving large systems of linear equations do not meet this criterion. We define the efficiency of an algorithm-hardware combination to be

$$\text{CPU time/connect time} = 1/(\text{stretch factor}), \tag{1.1}$$

where connect time = CPU time + I/O overhead.

Our work is directed towards the computer solution of QCD, which is the quantum field theory of the strong nuclear force. It describes how the observed hadrons, such as protons, neutrons, and pions, are built up as bound states of the elementary quarks via a non-Abelian gauge interaction mediated by gluons. The binding mechanism is essentially non-perturbative and beyond the reach of analytical methods. The gluons and quarks carry a strong "colour" charge, analogous to the electric charge. This takes values in the fundamental representation of the group $SU(3)$, and the whole theory is invariant under spacetime-dependent $SU(3)$ rotations of the fields, called gauge transformations. In 1974, Wilson [3] proposed a gauge invariant formulation of QCD, free from divergences, in which spacetime is replaced by a finite 4-dimensional Euclidean lattice of points

$$\begin{aligned} n = (n_1, n_2, n_3, n_4), \qquad & n_i = 1, \ldots, N_s \quad (i = 1, 2, 3) \\ & n_4 = 1, \ldots, N \end{aligned} \tag{1.2}$$

and the field derivatives by finite differences. We use a hypercubic lattice with unit spacing. The physical lattice volume should be large enough to contain the hadrons being studied, while the grid should be as fine as possible to give a good approximation to the spacetime continuum.

A typical lattice is of size $16^3 \times 24$ (elongated in the time direction, because we obtain hadron masses from the rate of decay of hadron propagators at large Euclidan times). Coupling constants are chosen so that such lattices measure approximately 1–2 fm across, which is just large enough to contain particles like the proton. Even larger lattices are desirable in order to keep finite size effects acceptably small.

The quantities we need to measure are average values, with respect to a Boltzmann-like probability distribution, of functions $O(\bar{\psi}, \psi, U)$ of the elementary quark and gluon fields

$$\langle O \rangle = Z^{-1} \int [dU][d\bar{\psi}][d\psi]\, O(\bar{\psi}, \psi, U)\, e^{-S(\bar{\psi}, \psi, U)}, \tag{1.3}$$

where $\bar{\psi}$ and ψ are quark fields on the sites, the U's are $SU(3)$ matrices associated with the links of the lattice, and Z is a normalisation factor. The action S is given by [3, 4],

$$\begin{aligned} S = S_G + \tfrac{1}{2} \sum_{n\mu} \bar{\psi}(n)\, \gamma_\mu (U_\mu(n)\, \psi(n+\mu) - U_\mu^\dagger(n-\mu)\, \psi(n-\mu)) \\ + m \sum_n \bar{\psi}(n)\, \psi(n), \end{aligned} \tag{1.4}$$

where m is the quark mass, and S_G is the pure gauge part of the action

$$S_G = \frac{-2}{g^2} \sum_{n\mu\nu} \operatorname{Re} \operatorname{Tr}(U_\mu(n)\, U_\nu(n+\mu)\, U_\mu^\dagger(n+\nu)\, U_\nu^\dagger(n)). \tag{1.5}$$

We use Susskind fermions [5] which are defined by a transformation [6] on the above action which diagonalises it in spinor space, so that three of the four spinor degrees of freedom may be discarded

$$\psi(n) = \gamma_1^{n_1}\gamma_2^{n_2}\gamma_3^{n_3}\gamma_4^{n_4}\chi(n). \tag{1.6}$$

Then the new action is

$$S = S_G + \tfrac{1}{2}\sum_{n\mu} \bar{\chi}(n)\, \eta_\mu(n)(U_\mu(n)\, \chi(n+\mu) - U_\mu^\dagger(n-\mu)\, \chi(n-\mu))$$
$$+ m \sum_n \bar{\chi}(n)\, \chi(n), \tag{1.7}$$

where

$$\eta_\mu(n) = \begin{cases} 1, & \mu = 1 \\ (-1)^{n_1}, & \mu = 2 \\ (-1)^{n_1+n_2}, & \mu = 3 \\ (-1)^{n_1+n_2+n_3}, & \mu = 4. \end{cases} \tag{1.8}$$

This saves a factor of 4 in storage. It is possible to perform the average over the quark fields, which are Grassmann variables, analytically. If we write $\mathscr{M} = m\mathscr{I}$ and

$$\begin{aligned}\mathscr{D}_{nm}[U] &= \tfrac{1}{2}\eta_1(n)[U_1(n)\, \delta_{n_1+1,m_1} - U_1^\dagger(m)\, \delta_{n_1-1,m_1}]\, \delta_{n_2m_2}\, \delta_{n_3m_3}\, \delta_{n_4m_4} \\ &+ \tfrac{1}{2}\eta_2(n)[U_2(n)\, \delta_{n_2+1,m_2} - U_2^\dagger(m)\, \delta_{n_2-1,m_2}]\, \delta_{n_1m_1}\, \delta_{n_3m_3}\, \delta_{n_4m_4} \\ &+ \tfrac{1}{2}\eta_3(n)[U_3(n)\, \delta_{n_3+1,m_3} - U_3^\dagger(m)\, \delta_{n_3-1,m_3}]\, \delta_{n_1m_1}\, \delta_{n_2m_2}\, \delta_{n_4m_4} \\ &+ \tfrac{1}{2}\eta_4(n)[U_4(n)\, \delta_{n_4+1,m_4} - U^\dagger(m)\, \delta_{n_4-1,m_4}]\, \delta_{n_1m_1}\, \delta_{n_2m_2}\, \delta_{n_3m_3},\end{aligned} \tag{1.9}$$

where $U_\mu(n)$ ($\mu = 1,..., 4$) is an $SU(3)$ matrix, and $\mathscr{I}$ the identity, the resulting formula for the quark propagator from the origin 0 to the site n is

$$\langle \psi(0)\, \bar{\psi}(n) \rangle = Z^{-1} \int [dU](\mathscr{D} + \mathscr{M})_{n0}^{-1} \det(\mathscr{D} + \mathscr{M})\, e^{-S_G(U)}. \tag{1.10}$$

Note that $\mathscr{D}$ is an $L \times L$ complex matrix (where $L = 3N_s^3 N$); indices for the complex components of the $SU(3)$ matrices at each site in (1.9) are suppressed for clarity. This matrix acts on complex vectors x of size L, a complex triplet at each site of the lattice.

In the quenched approximation the determinant term is neglected, and the remaining gauge field average is evaluated using a Monte Carlo algorithm [7] to generate a sequence of configurations distributed with probability $\exp(-S_G)$ so that

$$\langle \chi(0)\,\bar{\chi}(n)\rangle = \frac{1}{C} \sum_{\substack{i=1\\ \text{configs}}}^{C} (\mathscr{D}[\{U\}_i] + \mathscr{M})^{-1}_{n0}. \tag{1.11}$$

For a statistically significant estimate we should calculate $(\mathscr{D}+M)^{-1}_{n0}$ for many independent configurations. Each of these requires solving the system of linear equations

$$(\mathscr{D}[\{U\}_i] + \mathscr{M})^{ac}_{nm} x^c_m = \delta_{n0}\,\delta^{ab} \qquad (b = 1, 2, 3), \tag{1.12}$$

where the indices a, b, c refer to colour. For each initial spacetime origin 0 we calculate three columns x of the inverse, corresponding to the three values of the colour index b. This paper is concerned with finding the best way to calculate x for a given configuration of U matrices.

The sparse matrix $\mathscr{D}$ is not explicitly stored, instead we use the expression

$$(\mathscr{D}x)^a_n = \tfrac{1}{2} \sum_{\mu=1}^{4} \eta_\mu(n)(U_\mu(n)^{ac} x^c(n+\mu) - U^\dagger_\mu(n-\mu)^{ac} x^c(n-\mu)) \tag{1.13}$$

to define the action of a matrix operator $\mathbf{D}$ on a vector x, given a set of link matrices $\{U\}$. Note that the vector argument and result are complex triplets (in colour) at each site. In the remainder of our work we will use the operator forms $\mathbf{D}$, $\mathbf{M}$, and $\mathbf{I}$ rather than the equivalent matrices $\mathscr{D}$, $\mathscr{M}$, and $\mathscr{I}$.

The matrix operator $\mathbf{D}$ connects a site on the lattice to each of its nearest neighbours, but not to itself, so we can divide the lattice up into two classes of sites, even and odd [8] (depending on whether $n_1+n_2+n_3+n_4$ is even or odd) in such a way that the calculation of $\mathbf{D}x$ on even sites requires data on odd sites only (and vice versa). This allows us to split the propagator equation into two parts

$$\begin{aligned} \mathbf{D}x^{\text{odd}} + \mathbf{M}x^{\text{even}} &= \delta^{\text{even}} \\ \mathbf{D}x^{\text{even}} + \mathbf{M}x^{\text{odd}} &= \delta^{\text{odd}}, \end{aligned} \tag{1.14}$$

where the vector x^{even} is zero on all the odd sites, x^{odd} zero on all the even sites, and $x = x^{\text{even}} + x^{\text{odd}}$. By restricting our source term to the even sites we have

$$(-\mathbf{D}^2 + \mathbf{M}^2)\frac{x^{\text{even}}}{m} = \delta^{\text{even}} \tag{1.15a}$$

$$x^{\text{odd}} = -\mathbf{D}\frac{x^{\text{even}}}{m}. \tag{1.15b}$$

From (1.13), $\mathbf{D}^\dagger = -\mathbf{D}$, i.e., $\mathbf{D}$ is anti-hermitean, hence $-\mathbf{D}^2$ is hermitean. We construct $\mathbf{D}^2 x^{\text{even}}$ via two applications of (1.13). Solving (1.15a) for x^{even} and reconstructing x^{odd} with (1.15b) rather than solving for x directly (Eq. (1.12)), halves the size of the system of linear equations.

By storing only 2 rows of the $SU(3)$ link matrices as scaled 16 bit integers and reconstructing the third row when needed, we can compress the storage of the non-zero entries of **D** into 2×10^6 words of 32 bit memory. This is still 4 times the total memory of the DAP. We have exploited the gauge freedom to transform the configuration into temporal gauge (i.e., we have rotated all the links in the time direction to 1). This achieves a further 25% saving in storage.

A convenient partitioning of the operator $(\mathbf{D}+\mathbf{M})$ is into 3-dimensional timeslices, i.e.,

$$(\mathbf{D}+\mathbf{M}) = \left| \begin{array}{cccccc} (\mathbf{D}_1+\mathbf{M}) & \mathbf{T} & & & & \\ -\mathbf{T} & (\mathbf{D}_2+\mathbf{M}) & \mathbf{T} & & & \\ & -\mathbf{T} & (\mathbf{D}_3+\mathbf{M}) & \mathbf{T} & & \\ & \ddots & \ddots & \ddots & & \\ & & -\mathbf{T} & (\mathbf{D}_{N-1}+\mathbf{M}) & \mathbf{T} & \\ & & & -\mathbf{T} & (\mathbf{D}_N+\mathbf{M}) & \end{array} \right|, \tag{1.16}$$

where the operators $\mathbf{D}_t$ are the 3-dimensional equivalent of (1.13) (i.e., μ runs from 1 to 3) for links on timeslice t. Here $\mathbf{M} = m\mathbf{I}$ and

$$T = \tfrac{1}{2}(-1)^{n_1+n_2+n_3+n_4}\mathbf{I}, \tag{1.17}$$

where **I** is the 3-dimensional identity operator (of size $3N_s^3$). The operators $\mathbf{D}_t$ and **T** anti-commute

$$\mathbf{D}_t\mathbf{T} + \mathbf{T}\mathbf{D}_t = 0 \tag{1.18}$$

and

$$\mathbf{T}^2 = \tfrac{1}{4}\mathbf{I}. \tag{1.19}$$

We have chosen our fermionic boundary conditions to be Dirichlet in time (fermionic fields are zero for $t < 1$ and $t > N$ and all entries apart from the blocks shown in (1.16) are zero) and anti-periodic in the spatial directions. Previous calculations have shown that these boundary conditions are appropriate for the extraction of hadron masses [9]. Dirichlet boundary conditions have the additional advantage that, should subsequent analysis reveal that the time extent of the lattice is too short, prpagators may be extended to longer times using the Distant Source Method [10] without having to re-do the whole calculation. Each of the blocks $\mathbf{D}_t$ has dimension $3N_s^3$ and is banded, but with a very large bandwidth due to the anti-periodic boundary conditions in space. **D** is complex, block tridiagonal, anti-hermitean and sparse. For a $16^3 \times 24$ lattice we must solve a system of 294,912 complex linear equations.

By applying the operator **D** to itself and using (1.18) and (1.19) we arrive at a timesliced partitioning for $-\mathbf{D}^2$ in (1.15),

$$-\mathbf{D}^2=\left|\begin{array}{ccccc}
\frac{1}{4}\mathbf{I}-\mathbf{D}_1^2 & (\mathbf{D}_2-\mathbf{D}_1)\mathbf{T} & -\frac{1}{4}\mathbf{I} & & \\
(\mathbf{D}_2-\mathbf{D}_1)\mathbf{T} & \frac{1}{2}\mathbf{I}-\mathbf{D}_2^2 & (\mathbf{D}_3-\mathbf{D}_2)\mathbf{T} & -\frac{1}{4}\mathbf{I} & \\
-\frac{1}{4}\mathbf{I} & (\mathbf{D}_3-\mathbf{D}_2)\mathbf{T} & \frac{1}{2}\mathbf{I}-\mathbf{D}_3^2 & (\mathbf{D}_4-\mathbf{D}_3)\mathbf{T} & -\frac{1}{4}\mathbf{I} \\
\ddots & \ddots & \ddots & \ddots & \ddots \\
 & & & & -\frac{1}{4}\mathbf{I} \\
 & & & \frac{1}{2}\mathbf{I}-\mathbf{D}_{N-1}^2 & (\mathbf{D}_N-\mathbf{D}_{N-1})\mathbf{T} \\
 & & -\frac{1}{4}\mathbf{I} & (\mathbf{D}_N-\mathbf{D}_{N-1})\mathbf{T} & \frac{1}{4}\mathbf{I}-\mathbf{D}_N^2
\end{array}\right|, \tag{1.20}$$

$(-\mathbf{D}^2+\mathbf{M}^2)$ is hermitean positive definite and (still) very sparse. We are particularly interested in (1.15) for low values of m ($m \leqslant 0.1$), under these circumstances the operator is not diagonally dominant.

The rest of the paper is organised as follows. Section 2 describes our use of the Conjugate Gradient Method to solve (1.15). In Section 3 we consider strategies for preconditioning this algorithm. In Section 4 we investigate the application of an Iterative Block Gauss Elimination algorithm [11] to Eq. (1.12) and discuss its failings. In contrast we find that the Block Iterative Methods of Section 5 perform well, and we describe the implementation of an efficient Iterative Block Successive Over-Relaxation (SOR) algorithm. (We use the term Iterative Block SOR to refer to a formal solution of $N \times N$ block equations such as (1.15) with $-\mathbf{D}^2$ partitioned according to (1.20) using an SOR algorithm on the blocks and iterative solution of the reduced rank systems that arise thereby, likewise the term Iterative Block Gauss Elimination.) Our conclusions are presented in Section 6.

2. The Conjugate Gradient Algorithm

Our starting point for solving the propagator equation (1.15) is the Conjugate Gradient (CG) Algorithm [12], intoduced as an exact method for solving systems of linear equations, but now established as an iterative method for solving large sparse systems [13].

To solve $\mathbf{A}x=b$ for hermitean $\mathbf{A}=(-\mathbf{D}^2+\mathbf{M}^2)$

```
initial guess $x_0$
$p_0 = r_0 = b - \mathbf{A}x_0$
loop while $(r_k, r_k) > \varepsilon$ for $k = 0, 1, 2,...$
    $\alpha_k = (r_k, r_k)/(p_k, \mathbf{A}p_k)$
    $x_{k+1} = x_k + \alpha_k p_k$                                  (2.1)
    $r_{k+1} = r_k - \alpha_k \mathbf{A}p_k$
    $\beta_k = (r_{k+1}, r_{k+1})/(r_k, r_k)$
    $p_{k+1} = r_{k+1} + \beta_k p_k$
end loop
```

The Lanczos algorithm is also used in this context [14]; its relationship to the CG algorithm is described in [15].

Our choice of CG as a starting point was motivated by properties of the matrix operator **D**. First we must, under all circumstances, preserve its sparsity and structure. Second, we wish to investigate solutions to Eq. (1.15) for a range of values of m for each configuration, and we are most interested in the small m region for which $(-\mathbf{D}^2+\mathbf{M}^2)$ is not diagonally dominant. In this limit CG converges significantly faster than relaxation methods [8].

The CG algorithm is known to converge best for matrices with clustered eigenvalues and low condition number $K(\mathbf{A})$. For hermitean positive definite **A**,

$$K(\mathbf{A})=\frac{\Lambda_{\max}}{\Lambda_{\min}}, \tag{2.2}$$

where $\Lambda_{\max}$ and $\Lambda_{\min}$ are the largest and smallest eigenvalues of **A**. We have information on the spectrum of **D** from earlier work on small latticcs [16]; the distribution of high and low eigenvalues for an 8^4 configuration is shown in Fig. 2.1. Let **D** have eigenvalues $i\lambda$, λ real. The eigenvalues Λ of $(-\mathbf{D}^2+\mathbf{M}^2)$ will be λ^2+m^2. $\Lambda_{\max}\sim\lambda^2_{\max}\sim 18.5$ for values of m in our range of interest. When $m\gg\lambda_{\min}$, $\Lambda_{\min}\sim m^2$, and so the condition number will be approximately $18.5/\mathrm{m}^2$. However, when m is very small, it is $\lambda_{\min}$ that controls convergence. We know that this eigenvalue is very unstable, varying by many orders of magnitude from one configuration to the next. Hence, provided we do not lower m too far, we would expect [13],

$$\log(r_k, r_k)\sim k\log((\sqrt{K-1})/(\sqrt{K+1}))\sim -mk. \tag{2.3}$$

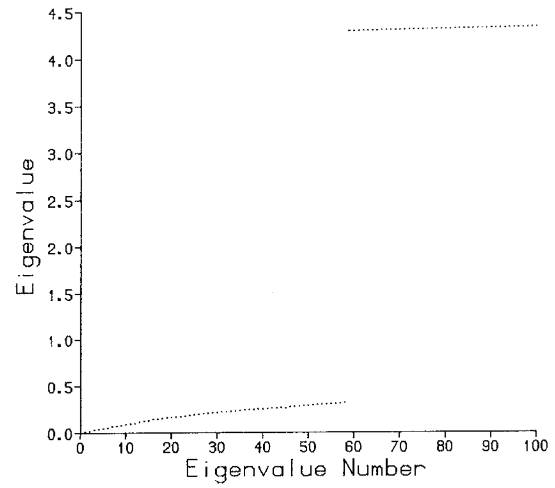

FIG. 2.1. Spectrum of the absolute values of the eigenvalues of an 8^4 **D** operator. The first 100 eigenvalues to converge using a Lanczos algorithm are plotted.

We have implemented the CG method for lattices of size 16^4 on the DAP. (Similar work has been done on a Cyber 205 [17].) Empirically, as shown in Fig. 2.2, we find that the iteration scheme converges smoothly, at a linear rate approximately proportional to m (for $0.01 \leqslant m \leqslant 0.50$) as in (2.3), up to some level determined by the precision used to do the calculation.

The departure of r_k, the iterative residual vector, from $b - \mathbf{A}x_k$ is a sign of the onset of roundoff errors in the CG algorithm; the importance of such errors can be judged by restarting the system with x_k as the new x_0. We see a marked increase in the residual on restarting after 700 CG iterations (see Table I) in 32-bit arithmetic at $m = 0.01$, indicating that roundoff effects have become significant. We restart the solver after 500 iterations at the lowest mass, and run for a further 200 iterations, by which time the desired accuracy (Table II) is attained. We use the following citeria to decide how long to run the CG algorithm:

(i) That baryon propagators (sums over space and colour of x^3) should be unchanged in the third decimal place on all timeslices under further iteration.

(ii) That on restarting, the baryon propagators should not change and the norm of the residual vector should not increase significantly.

(iii) That the baryon propagators obtained should be the same (to the above tolerance) as those obtained from the same configuration after it has undergone a random gauge transformation. (The baryon propagators are independent of choice of gauge, but the quark propagators x are not). We performed this test on our first configuration to check the program.

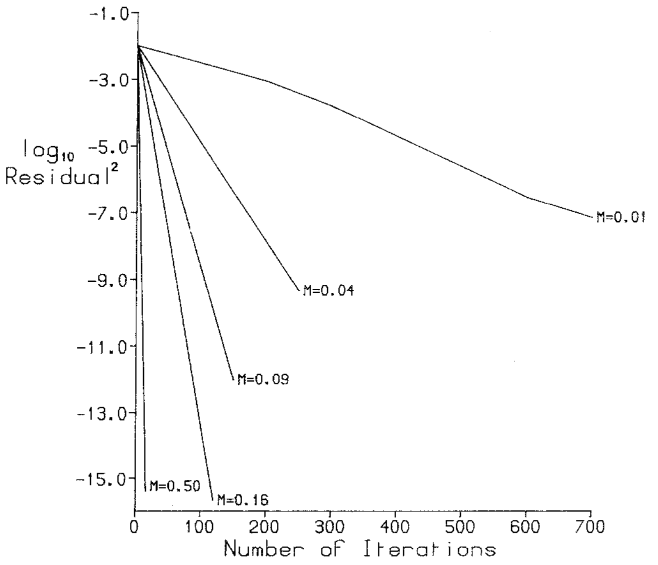

FIG. 2.2. Rate of convergence of conjugate gradient algorithm, showing the dependence on mass for a 16^4 lattice.

TABLE I

Measurements of (r_k, r_k) for CG Algorithm on a 16^4 Lattice, and the Effects of Restarting the Iteration Scheme; $m = 0.01$, $g^2 = 1.0$

Number of sweeps	(r_k, r_k)	(r_k, r_k) on restart
100	$0.285D-2$	
200	$0.322E-3$	
300	$0.406E-4$	
400	$0.746E-5$	
500	$0.771E-6$	$0.780E-6$
600	$0.903E-7$	
700	$0.856E-8$	$0.150E-7$

The values of (r_k, r_k) sufficient to satisfy these conditions for $0.01 \leqslant m \leqslant 0.5$ are given in Table II. We apply the above conditions on the last timeslice; closer to the origin the baryon propagators are stable in the 4th and 5th decimal places.

The CG algorithm performs well, but is very expensive in terms of storage. We need to store 3 vectors per colour, each of size $3N_s^3N$ words, which together with the $27N_s^3N$ words of links must be paged through the machine on each iteration, as the system is too big to be held in memory. This disk-to-DAP paging of timeslices is done asynchronously, the links for timeslice $(t+1)$ being paged on while those for timeslice t are being used. The DAP data expansion software, DDX, enables variables held in COMMON areas to be transferred between DAP and disk while the program is running.

The CG vectors can only be updated when calculations of the scalars α and β are complete. Barkai *et al.* [17] have proposed a modified CG algorithm which avoids some of the synchronisation problems at the expense of an extra vector. However, because of the DAPs low *I/O* rate (approximately 250–300 Kbytes/sec compared to a floating-point performance of around 15 Mflops) the matrix-vector multiply step $q_k = \mathrm{A}x_k$ is *I/O* bound by itself, and so we use the standard algorithm for a hermitian matrix.

TABLE II

Stopping Conditions for CG Algorithm with Typical Measurements of (r_k, r_k)

Number of iterations \ Mass	0.01	0.04	0.09	0.16	0.5
At $g^2 = 0.95$	$500 + 200$	300	150	120	40
At $g^2 = 1.00$	$500 + 200$	250	120	100	40
Typical (r_k, r_k)	$0.85E-08$	$0.41E-09$	$0.84E-12$	$0.20E-15$	$0.38E-15$

TABLE III

Approximate Timings (in sec) for CG Algorithm on the DAP for a 16^4 Lattice

Step[a]	*CPU*	*I/O*	*I/O* overhead
Calculate $\mathbf{A}p_k$	7.2	15	10
Update vectors	0.6	30	30
Scalar products	0.2	0	0
Totals	8	45	40
Total time per iteration = 48 sec			

[a] See Eq. (2.1).

The total connect time for a propagator calculation running on the DAP is a factor of 6 longer than the processor time used. Similar *I/O* stretch factors are reported [17] for a Cyber 205. Details of the timings are given in Table III. Almost all the DAP CPU time is used applying the **D** operators. CPU time for updating the vectors is insignificant. The 10 sec of *I/O* overhead that occur while the links are being paged in and the **D** operators applied is not available. But during the longer (30 sec) overhead while the vectors are paged on and off the machine, we could do useful work. In the next section we describe a preconditioned Conjugate Gradient algorithm. The work done modifying the residual vector (from r_k to d_k, see below) in this algorithm can be performed while the CG vectors are being moved.

The standard Conjugate Gradient method (without preconditioning) is only really practical for propagator calculations when either (i) the system of equations is small enough to fit within the machine's fast memory or (ii) the slow-to-fast memory bandwidth is sufficiently high to keep the processor(s) going all of the time. Systems of equations satisfying (i) are not usually large enough to be of physical significance and machines satisfying (ii) are not widely available. (The Cray-2 and ETA-10 may achieve the necessary data rates.) Since most applications require as large a grid as possible, we must look for new algorithms that do not require large numbers of vectors, and that use many more floating-point operations per word of *I/O* transfer.

3. Preconditioning

We began to consider the problem of introducing a preconditioning step into our Conjugate Gradient algorithm with two questions in mind, (i) could we find a suitable preconditioning matrix **N**, and (ii) if so, could the preconditioning steps be performed while the CG vectors were being paged on and off the machine, prior to, and after updating. Previous studies [18] of preconditioned CG algorithms have concluded that significant reductions in the number of CG iterations can be achieved, but at the cost of much more CPU time per sweep. Our aim is to utilise the CPU time we have available (because of the *I/O* overhead in the CG code) to offset this disadvantage.

The motivation for preconditioning is to speed the convergence of an iterative solution of the system of equations $\mathbf{A}x = b$ by premultiplying by $\mathbf{N}^{-1}$, where $\mathbf{N}^{-1}$ is an approximation to $\mathbf{A}^{-1}$, and then solving

$$\mathbf{N}^{-1}\mathbf{A}x = \mathbf{N}^{-1}b, \tag{3.1}$$

where $K(\mathbf{N}^{-1}\mathbf{A}) \ll K(\mathbf{A})$. If $\mathbf{N}^{-1}$ is a good approximation to $\mathbf{A}^{-1}$ then $\mathbf{N}^{-1}\mathbf{A} \sim \mathbf{I}$, $K(\mathbf{N}^{-1}\mathbf{A}) \sim 1$ and our iterative scheme will converge rapidly.

The choice of preconditioner $\mathbf{N}$ is crucial. We are restricted by the requirement that it must use little or no extra storage space, and by the necessity of having $\mathbf{N}$ in operator form. We cannot, for example, consider using block *LU* decomposition or block-diagonal scaling, as the calculation of the block-diagonal inverses for $\Delta = \text{blockdiag}(\mathbf{D} + \mathbf{M})$ would require $3N_s^2$ column inversions, and some 150 Mbytes of storage for each timeslice, as all such inverses are dense (like $(\mathbf{D} + \mathbf{M})^{-1}$ itself).

Instead we use a preconditioned CG algorithm of the form [13],

```
initial guess $x_0$
$r_0 = b - \mathbf{A}x_0$
solve $\mathbf{N}d_0 = r_0$
$p_0 = d_0$
loop while $(r_k, d_k) > \varepsilon$ for $k = 0, 1, 2, \ldots$
    $\alpha_k = (r_k, d_k)/(p_k, \mathbf{A}p_k)$
    $x_{k+1} = x_k + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k \mathbf{A}p_k$                         (3.2)
    solve $\mathbf{N}d_{k+1} = r_{k+1}$
    $\beta_k = (r_{k+1}, d_{k+1})/(r_k, d_k)$
    $p_{k+1} = d_{k+1} + \beta_k p_k$
end loop
```

where $\mathbf{N}$ is an approximation to $\mathbf{A}$ and the system of equations $\mathbf{N}d_{i+1} = r_{i+1}$ is well conditioned and its iterative solution converges rapidly.

We have been experimenting with preconditioning using free fermions. We use the free fermion operator $\mathbf{D}_\mathrm{F}$, Eq. (1.9) with all the U's set equal to the identity, to approximate $\mathbf{D}$ and on each iteration of the CG method we solve the system

[illegible]

where $\mathbf{M}_\mathrm{F} = m_\mathrm{F}\mathbf{I}$, for each of the 3 complex colour components of d_{i+1} $(a = 1, 2, 3)$. The free fermion systems are well conditioned $\Lambda_{\max} \leqslant m_\mathrm{F}^2 + 16$ and $\Lambda_{\min} \geqslant m_\mathrm{F}^2 +$

$3\sin^2 p_{\min}$, where $p_{\min} = \pi/N_S$ for anti-periodic boundary conditions in all 3 spatial directions, so in our case $K(\mathbf{N}) \leqslant 140$. The operator $\mathbf{D}_F$ can be applied without using any extra storage space and the systems can each be solved quickly; the residual squared for the system of equations (3.3) being $\sim 10^{-16}$ after 15 iterations of an inner CG loop. The mass m_F is a free parameter which should be optimised, having a significant dependence only on the gauge coupling and the quark mass.

Our choice of a free fermion preconditioner is motivated by the fact that at short distances (within a proton, e.g.) quarks behave as free particles since QCD is an asymptotically free theory. Consequently, free propagation becomes a better approximation as the lattice spacing decreases, which will be the case for simulations using larger lattices. So we would expect this preconditioner to work better the larger and computationally more demanding the system. There is a problem, however, in that within a gauge theory there is no clear distinction between high and low momentum modes unless we fix a gauge. The quark propagator is gauge dependent and will only resemble a free propagator in a gauge where the link variables are chosen to be as close to the unit matrix as possible. We began by using axial gauge in which all the timelike links (this is possible with Dirichlet b.c. on the fermions) and as many of the spacelike links as possible are rotated to the identity. We employ temporal gauge (timelike links only rotated to the identity) in any case to save on storage.

We have not succeeded in reducing the total time necessary to solve the system with this choice of gauge. We have recently learned that the Cornell group [19] have had some success using covariant gauges, and by storing an approximation to the diagonal of $(\mathbf{D}+\mathbf{M})$ in momentum space as a precontitioner, translating between momentum space and configuration space using fast fourier transforms (FFTs).

The group working at Tsukuba University [20] have presented an incomplete LDU decomposition algorithm which they use in the context of dynamical fermion simulations (on $9^3 \times 18$ lattices).

4. Iterative Block Gauss Elimination

Following the method of Bowler *et al.* [11], we perform Gauss Elimination on the blocks of the matrix $(\mathbf{D}+\mathbf{M})$ partitioned according to (1.16), i.e., omitting even–odd partitioning for simplicity. Consider the first two rows of the operator $(\mathbf{D}+\mathbf{M})$. Let $\mathbf{P}_0 = \mathbf{I}$ and $\mathbf{P}_1 = 4(\mathbf{D}_1 + \mathbf{M})\mathbf{T}$, add row one to $\mathbf{P}_1$ (row two) and we get

$$\left|\begin{matrix} \mathbf{P}_1\mathbf{T} & \mathbf{P}_0\mathbf{T} & 0 & \\ 0 & \mathbf{P}_1(\mathbf{D}_2+\mathbf{M})+\mathbf{P}_0\mathbf{T} & \mathbf{P}_1\mathbf{T} & 0 \end{matrix}\right|. \tag{4.1}$$

Let $\mathbf{P}_2\mathbf{T} = \mathbf{P}_1(\mathbf{D}_2 + \mathbf{M}) + \mathbf{P}_0\mathbf{T}$, then $\mathbf{P}_2 = 4\mathbf{P}_1(\mathbf{D}_2 + \mathbf{M})\mathbf{T} + \mathbf{P}_0$ as $4\mathbf{T}^2 = \mathbf{I}$. Continue this process until all the blocks in the lower triangle have been eliminated.

$$\left|\begin{array}{cccccc} \mathbf{P}_1\mathbf{T} & \mathbf{P}_0\mathbf{T} & 0 & & & \\ 0 & \mathbf{P}_2\mathbf{T} & \mathbf{P}_1\mathbf{T} & 0 & & \\ \ddots & \ddots & \ddots & \ddots & & \\ & & & 0 & \mathbf{P}_{N-1}\mathbf{T} & \mathbf{P}_{N-2}\mathbf{T} \\ & & & & 0 & \mathbf{P}_N\mathbf{T} \end{array}\right| \left|\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{array}\right| = \left|\begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_{N-1} \\ c_N \end{array}\right|, \tag{4.2}$$

where

$$\mathbf{P}_t = \mathbf{P}_{t-2} + 4\mathbf{P}_{t-1}(\mathbf{D}_t + \mathbf{M})\mathbf{T} \tag{4.3}$$

and

$$c_N = \delta_1 + \mathbf{P}_1\,\delta_2 + \cdots + \mathbf{P}_{N-1}\,\delta_N. \tag{4.4}$$

For a delta function source on timeslice t,

$$\delta_{t'} = \begin{cases} \delta_{n0} & \text{for } t = t' \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad c_N = \mathbf{P}_{t-1}\,\delta_{t'}. \tag{4.5}$$

x_N is the propagator from the origin to timeslice N, and its calculation requires solution of a system of 12,288 complex linear equations rather than 294,912,

$$\mathbf{P}_N\mathbf{T}x_N = c_N \tag{4.6}$$

(down by a factor of N on x in (1.12)). The matrix operator $\mathbf{P}_N$ is not hermitian, instead we solve the system

$$\mathbf{P}_N^\dagger\mathbf{P}_N\mathbf{T}x_N = \mathbf{P}_N^\dagger c_N \tag{4.7}$$

using a CG algorithm. Having solved (4.7) for x_N, we obtain x_t for $1 \leqslant t < N$ by back substitution

$$x_t = -4\mathbf{T}\,\delta_{t'+1} + 4\mathbf{T}(\mathbf{D}_{t+1} + \mathbf{M})\,x_{t+1} + x_{t+2}, \tag{4.8}$$

setting $x_{N+1} = 0$. However, $\mathbf{P}_N$ involves all N timeslices, so we must bring the complete set of links through the machine (twice) per iteration of the CG algorithm. These transfers are done asynchronously. The stretch factor is of order 2.

We began testing this scheme for the interacting theory at high quark mass ($m = 0.5$), and obtained convergence rates in line with those predicted by Bowler *et al.* [11], and pion propagators consistent with previous results. But when the quark mass was lowered, convergence rates dropped dramatically and pion masses calculated from apparently converged quark propagators were clearly incorrect. At a quark mass of 0.01 we failed to obtain a converged propagator after 10,000 sweeps on a $16^3 \times 8$ system (see Fig. 4.1).

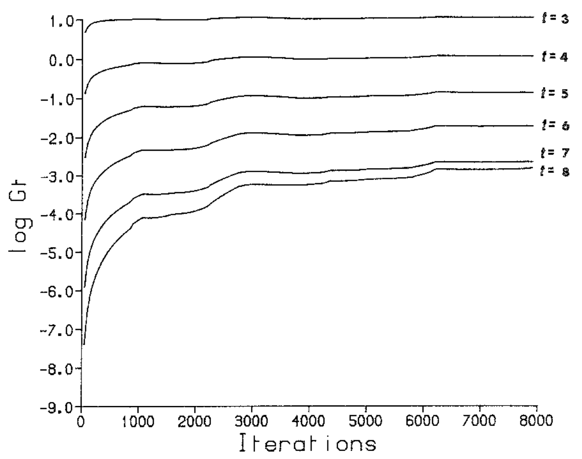

FIG. 4.1. Convergence of the pion propagator signal G_t using Iterative Block Gauss Elimination algorithm; 8000 sweeps on a $16^3 \times 8$ lattice at $m = 0.01$ with origin on timeslice $t = 3$.

We can explain the failure of this method to converge in two ways:

(i) The operator $\mathbf{P}_N$, although of dimension 12,288 rather than 294,912, is a fully dense matrix, containing terms linking each site of the 16^3 spatial lattice to every other site. It is only diagonally dominant at very high m.

(ii) The projection mechanism (4.8) for obtaining x_t on timeslices 1 to $N-1$ amplifies exponentially any errors present in the solution x_N. Later studies showed that the propagator on timeslice N is always the last to converge and so using it as a basis for obtaining x_t for $1 \leqslant t < N$ is unsound.

Having reached these conclusions we examined the free fermion data of Bowler *et al.* for a 16^4 lattice and found the convergence of the propagators to be qualitatively the same, but on a much shorter timescale (see Fig. 4.2). Pion propagators, given by

$$\langle \pi(0)\, \bar{\pi}(n_t) \rangle = \sum_{n_1, n_2, n_3} |\langle \chi(n)\, \bar{\chi}(n) \rangle|^2 \tag{4.9}$$

measured using the quark propagators of Bowler *et al.*, are stable and apparently converged to five or more decimal places for large numbers of iterations.[1] They then undergo changes of several orders of magnitude before converging to the analytic result. This seems to indicate that the algorithm is losing track of the long range structure in time of the propagators.

We therefore conclude that this Iterative Block Gauss Elimination algorithm is

[1] Fifty out of a total of 100 iterations for free fermions, 500 out of 1500 for a pure gauge configuration, i.e., a random gauge transformation of the unit configuration. This disparity in the rates of convergence for gauge equivalent configurations was not observed for the other algorithms considered here.

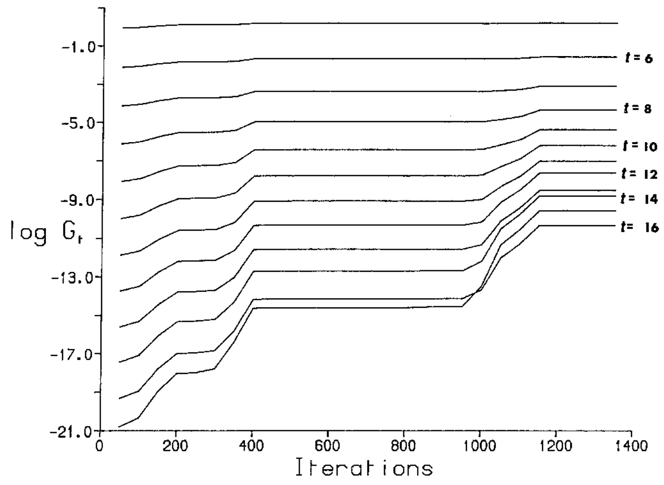

FIG. 4.2. Convergence of the free pion propagator signal G_t for Iterative Block Gauss Elimination algorithm; 1400 sweeps on a 16^4 lattice at $m = 0.01$.

intrinsically unreliable; without knowing the answer in advance, it is difficult to know when to stop iterating when the convergence pattern is like that in Fig. 4.2.

We would like to be able to retain some of the features of this algorithm—in particular, solution of small systems of linear equations (preferably for diagonally dominant matrix operators). But we require a scheme in which an approximation to the correct long range structure of the propagator is obtained quickly. Such a method, which works well, is discussed in the next section.

5. Block Iterative Methods

Consider the partitioning scheme for $(\mathbf{D}+\mathbf{M})$ in temporal gauge (1.16). We can define a class of block-iterative algorithms by regarding each $3N_s^3 \times 3N_s^3$ complex block as a component of an $N \times N$ linear system of equations, applying an established iterative algorithm (e.g., Jacobi, Gauss–Seidel or successive over-relaxation (SOR)) to the $N \times N$ block equation and using an inner CG loop (where applicable) to solve the reduced rank systems of equations.

We use an SOR scheme to solve the $N \times N$ system of block equations (5.1),

$$\left|\begin{array}{ccccc} (\mathbf{D}_1+\mathbf{M}) & \mathbf{T} & & & \\ -\mathbf{T} & (\mathbf{D}_2+\mathbf{M}) & \mathbf{T} & & \\ & -\mathbf{T} & (\mathbf{D}_3+\mathbf{M}) & \mathbf{T} & \\ \ddots & \ddots & \ddots & & \\ & -\mathbf{T} & (\mathbf{D}_{N-1}+\mathbf{M}) & \mathbf{T} & \\ & & -\mathbf{T} & (\mathbf{D}_N+\mathbf{M}) & \end{array}\right| \left|\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \\ x_N \end{array}\right| = \left|\begin{array}{c} \delta_1 \\ \delta_2 \\ \delta_3 \\ \vdots \\ \delta_{N-1} \\ \delta_N \end{array}\right| \qquad (5.1)$$

and an inner iterative scheme to solve the system of equations

$$(\mathbf{D}_t+\mathbf{M})\,x_t^{(k+1)}=\omega(\delta_t+\mathbf{T}x_{t-1}^{(k+1)}-\mathbf{T}x_{t+1}^{(k)})+(1-\omega)(\mathbf{D}_t+\mathbf{M})\,x_t^{(k)} \qquad (5.2)$$

for each x_t on each iteration k of the outer SOR scheme, where the parameter ω is selected for optimum convergence. We can generalise this to the solution of $(-\mathbf{D}^2+\mathbf{M}^2)\,x^{\text{even}}=\delta^{\text{even}}$ (1.15) by analogy with (1.20),

$$\begin{aligned}(\mathbf{C}-\mathbf{D}_t^2+\mathbf{M}^2)\,&y_t^{(k+1)}\\ &=\beta_t\\ &=\omega(\delta_t+\tfrac{1}{4}y_{t-2}^{(k+1)}+(\mathbf{D}_{t-1}-\mathbf{D}_t)\,\mathbf{T}y_{t-1}^{(k+1)}+(\mathbf{D}_t-\mathbf{D}_{t+1})\,\mathbf{T}y_{t+1}^{(k)}+\tfrac{1}{4}y_{t+2}^{(k)})\\ &\quad+(1-\omega)(\mathbf{C}-\mathbf{D}_t^2+\mathbf{M}^2)\,y_t^{(k)}, \qquad (5.3)\end{aligned}$$

where $\mathbf{C}=c\mathbf{I}$ and $y_t=x^{\text{even}}$.

The method can be used for any choice of temporal fermion boundary conditions, e.g.,

for periodic $c=\frac{1}{2}$ and y_1 is identified with y_{N+1},

for Dirichlet $c=\frac{1}{4}$ for y_1 and y_N and $c=\frac{1}{2}$ for y_t $1<t<N$; terms on the r.h.s. of Eq. (5.3) with $t<1$ or $t>N$ are dropped.

The operator $\mathbf{D}_t$ (defined by (1.13) for $\mu=1,\ldots,3$ rather than $1,\ldots,4$) is anti-hermitian like $\mathbf{D}$. So by analogy with $\mathbf{D}^2$, $\mathbf{D}_t^2$ is hermitian, c and m are real, and so a standard CG algorithm can be used to solve the inner systems of equations. The matrix operator $(\mathbf{C}-\mathbf{D}_t^2+\mathbf{M}^2)$ is diagonally dominant (its diagonal elements being $\frac{3}{2}+c+m^2$). Further, this diagonal dominance remains as m goes to 0, and so convergence of the inner CG algorithm is very rapid for all m. Denote this scheme IBSOR (iterative block SOR).

It would appear that the IBSOR scheme would require a huge amount of work, to exactly invert N systems of size $3N_s^3$ per sweep, but this is not the case. β_t (the r.h.s. of Eq. (5.3)) is accumulated from terms on 5 timeslices, and if we assume that the algorithm converges then the error on 3 of these timeslices y_t, y_{t+1}, y_{t+2} is greater than that on the other two y_{t-2}, y_{t-1}. So we should aim to converge the $(k+1)$th iterate on timeslice t to a level where the residual on this timeslice is some factor lower than that on $(t-1)$. We should not run the inversion so long that we do work which will be wasted on the next sweep, when y_{t+1} and y_{t+2} have also been upgraded. This fits in with our aim of producing a balanced or CPU-dominated program, if we can do the necessary iterations in the time taken to page out y_{t-2} and page in y_{t+3} and the next set of links. This proves to be the case.

5.1. *Tuning the IBSOR Algorithm*

The SOR algorithm does not converge for all values of its parameter ω, the range of acceptable values being determined by the eigenvalue spectrum of the matrix. We must determine whether there exists a set of values for which IBSOR converges, and if so what the optimum values of ω are. We must also determine whether the tuning of ω depends upon our choice of gauge configuration—if it does then the algorithm will be useless as we must calculate propagators on large numbers of configurations.

Calculation of optimum values for ω would require a detailed knowledge of the eigenvalue spectra of the $\mathbf{D}_t$, which we do not have. Instead we began by setting $\omega = 1$ (the Gauss–Seidel limit of IBSOR) and m to 0.50 (a high value) and running the code to obtain a benchmark with which to compare IBSOR. To tune ω for a given m we ran the code for a fixed number of iterations (from a $y_0 = 0$ start), measuring the residual on each sweep for 4 values of ω and noting the rate of fall of the norm of the residual. We then repeated this process using a second configuration. The optimum values of ω were found to differ from one configuration to the next (as one would expect given the known variation in the lowest eigenvalues) but not by an unacceptable degree (see Fig. 5.1). We continued refining our values for ω until the difference between the upper and lower bounds on the optimum value was approximately equal to the variation between the two test configurations.

We have tuned the parameter ω for mass values between 0.01 and 0.50 (see Table IV). We find that quark mass is the only significant independent variable. ω is independent of N (for $8 \leqslant N \leqslant 32$) and does not need retuning from one configuration to the next. When the coupling constant g^2 in Eq. (1.5) is changed (in the small range we have explored) only slight adjustments of ω are necessary. The range of acceptable ω values narrows with decreasing mass.

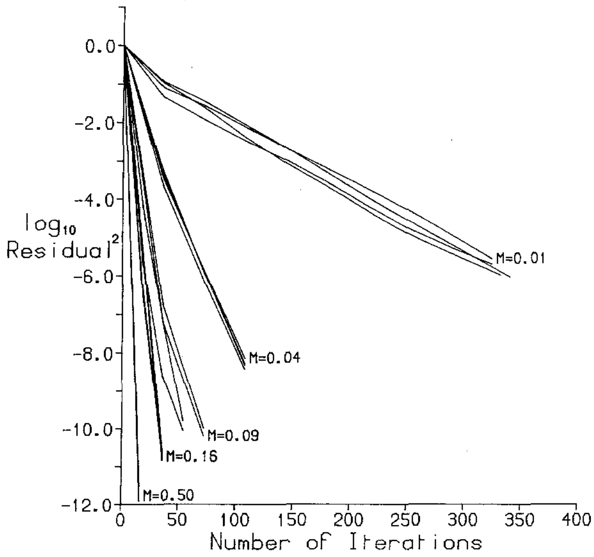

FIG. 5.1. Rate of convergence of IBSOR algorithm, showing the dependence on mass for 4 different gauge configurations on a $16^3 \times 24$ lattice.

TABLE IV

Optimum Values of ω for a $16^3 \times 24$ Lattice at $g^2 = 1.0$

m	0.50	0.16	0.09	0.04	0.01
ω	1.25	1.55	1.70	1.88	1.955

5.2. *Convergence Rates and Roundoff Errors*

Figure 5.1 illustrates the variation of the rate of convergence with mass and the slight dependence on configuration (at the lightest mass). The attainable values of (r_k, r_k), the residual squared, are limited by the precision to which we store the propagator as shown in Table V. But because the calculation of β_t is limited to a range of 5 timeslices and there are no global scalars to be accumulated, the timesliced (r_k, r_k) falls off exponentially away from the source, with the propagator. This is demonstrated in Table V and Fig. 5.2. The source term is seen by the algorithm on every sweep, so there is no roundoff-error-induced cumulative drift of the propagator from the correct solution.

The variation of convergence rate with the number of CG iterations used in the inner loop is shown in Fig. 5.3 for a mass of 0.04. Eight iterations of the CG inverter are more than sufficient at all our mass values. This can be traced to $c \geqslant \frac{1}{4}$ in the l.h.s. of Eq. (5.3) which guarantees diagonal dominance of the sub-blocks.

We find that 32-bit arithmetic is sufficient to converge all our propagators $M = 0.01, ..., 0.50$ on a $16^3 \times 24$ lattice. The convergence rate is independent of N (for $8 \leqslant N \leqslant 32$) and the behaviour of the timesliced residual shown in Fig. 5.2 is a feature of all mass values.

TABLE V

Limiting Squared Residuals (r_k, r_k) for IBSOR Algorithm

		Limiting (r_k, r_k)	Timesliced (r_k, r_k)		
Free fermions			t_0	$t_0 + 10$	
16^4	32 bit	0.14×10^{-13}	0.13×10^{-13}	0.92×10^{-18}	
	64 bit	0.20×10^{-30}	0.18×10^{-30}	0.15×10^{-35}	
Interacting fermions			$t_0 + 5$	$t_0 + 10$	$t_0 + 15$
$16^3 \times 24$	$m = 0.01$	0.1×10^{-8}	0.68×10^{-10}	0.35×10^{-10}	0.19×10^{-10}
32 bit	$m = 0.09$	0.3×10^{-10}	0.54×10^{-12}	0.22×10^{-13}	0.10×10^{-14}
	$m = 0.50$	0.1×10^{-12}	0.38×10^{-15}	0.19×10^{-18}	0.86×10^{-22}

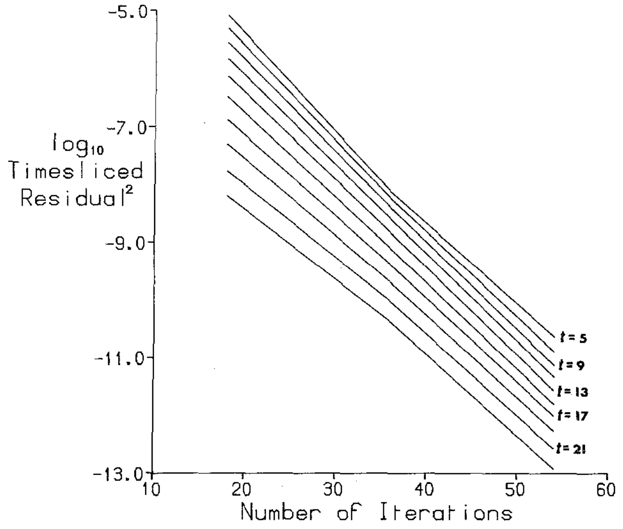

FIG. 5.2. Timesliced squared residuals $(r_k, r_k)_t$ for IBSOR algorithm on a $16^3 \times 24$ lattice at $m = 0.09$, timeslices $t_0, t_0 + 2,..., t_0 + 18$, for an origin on timeslice 5.

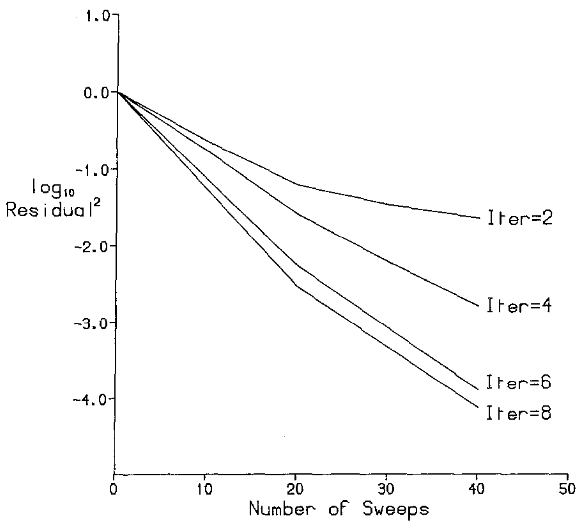

FIG. 5.3. Dependence of the rate of convergence of IBSOR on the number of CG inversion steps, for a $16^3 \times 24$ lattice at $m = 0.04$.

5.3. *Performance*

We tested our algorithm by measuring the number of sweeps necessary to obtain agreement between hadron propagators calculated using our CG algorithm and those obtained using the IBSOR method. In both cases we required convergence of the propagators on the last timeslice to 3 significant figures (this gives 4 or 5 significant figures near the source) and the results agreed to this accuracy.

When we use 6 or 8 iterations of the inner CG inverter per sweep we find that our stretch factor is 1.02 (compared with 6 for our full CG algorithm). The time taken to produce a set of 16^4 propagators drops by a factor of around 3. For a $16^3 \times 24$ or larger lattice we gain a factor of between 5 and 7 on the CG method, increasing with mass.

5.4. *Implementation of the IBSOR Algorithm on Other Computer Systems*

Our work to date has been on the DAP, whose performance peaks at 15 Mflops and whose asynchronous *I/O* rate is around 250–300 Kbytes/sec. Our conclusions,

present form has one third of the vector *I/O* per sweep, and requires significantly fewer sweeps. Consequently, performance is determined by the CPU speed, not the slow-to-fast-memory transfer rate.

Further, the removal of synchronising scalars means that IBSOR can run at very close to 100% efficiency on multiprocessor systems, such as the Cray XMP/4. If processor 1 is inverting the $(k+1)$th block equation for y_t then processor 2 can work in parallel on the $(k+2)$th equation for y_{t-3} without any danger of write conflicts arising. Extending this idea we can fully utilise an n processor system when our lattice has temporal extent $3n$ or larger.

For a larger array of less powerful processors, e.g., a few hundred Transputers, other partitioning schemes may be more appropriate. In this case we expect an iterative block SOR scheme based on 2- rather than 3-dimensional partitioning would perform very well. Hypercube architectures might make use of a 4-dimensional partitioning scheme.

6. Conclusions

We conclude that when the systems are small enough to fit in fast memory, the conjugate gradient algorithm is a good way of obtaining a few columns of the inverse (Susskind) fermion matrix, but for large systems the IBSOR algorithm is more efficient. At present IBSOR requires slightly more CPU time than CG at low masses, but there is no *I/O* overhead. Thus on the DAP it requires less than $\frac{1}{5}$ of the connect time of CG. In addition roundoff and synchronisation problems in accumulating the CG scalars have been avoided, and so bigger systems can be studied in 32-bit arithmetic than is possible with CG.

Using the IBSOR algorithm we have been able to compute quark propagators at

5 quark mass values for 32 configurations at $g^2 = 1.05$, 1.0, and 0.95. We will continue to explore the range of couplings within which our $16^4 \times 24$ lattice should be a good approximation to the spacetime continuum, making high statistics measurements of hadron propagators and matrix elements.

Acknowledgments

We would like to thank the Edinburgh Regional Computer Centre for their continued DAP support and, in particular M. Brown. A. McKendrick, and K. Yarwood for writing the DDX data transfer software. The DAPs are supported by SERC Grants NG 11849 and NG 15908. We are grateful to K. C. Bowler and D. J. Wallace for many helpful discussions during this work. D.R. is supported by SERC and Inmos U.K. Ltd., C.B.C. acknowledges the award of a Vans Dunlop scholarship from the University of Edinburgh.

References

1. K. C. Bowler, C. B. Chalmers, R. D. Kenway, G. S. Pawley, and D. Roweth, *Nucl. Phys. B* **284**, 299 (1987).
2. K. C. Bowler and G. S. Pawley, *Proc. IEEE* **72**, 42 (1984); G. S. Pawley and G. W. Thomas, *J. Comput. Phys.* **47**, 165 (1982).
3. K. G. Wilson, *Phys. Rev. D* **10**, 2445 (1974).
4. K. G. Wilson in *New Phenomena in Subnuclear Physics, Erice 1975*, edited by A. Zichichi (Plenum, New York, 1977).
5. L. Susskind, *Phys. Rev. D* **16**, 3031 (1977).
6. N. Kawamoto and J. Smit, *Nucl. Phys. B* **192**, 100 (1981).
7. J. B. Kogut, *Rev. Mod. Phys.* **55**, 775 (1983); M. Creutz, L. Jacobs, and C. Rebbi, *Phys. Rep.* **95**, 201 (1983).
8. K. C. Bowler, D. L. Chalmers, A. Kenway, R. D. Kenway, G. S. Pawley, and D. J. Wallace, *Nucl. Phys. B* **240** [FS12], 213 (1984).
9. C. Bernard, T. Draper, and K. Olynyk, *Phys. Rev. Lett.* **49**, 1076 (1982); *Phys. Rev. D* **27**, 227 (1983).
10. R. D. Kenway, *Phys. Lett. B* **158**, 327 (1985).
11. K. C. Bowler, R. D. Kenway, G. S. Pawley, and D. J. Wallace, *Phys. Lett. B* **145**, 88 (1984).
12. M. R. Hestenes and E. Stiefel, *J. Res. Nat. Bur. Standards* **49**, 409 (1952); Golub & Van Loan, *Matrix Computations* (John Hopkins Univ. Press, Baltimore, 1983), pp. 352–379; J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* (Springer-Verlag, Berlin, 1980), p. 572; I. S. Berezin and N. P. Zhidkov, *Computing Methods* (Pergamon, Oxford, 1965), p. 34.
13. J. K. Reid, in *Proc. Conference on Large Sparse Sets of Linear Equations* (Academic Press, New York, 1971); P. Concus, G. H. Golub, and D. P. O'Leary in *Sparse Matrix Computations*, edited by J. R. Bunch and D. J. Rose (Academic Press, New York, 1976).
14. I. M. Barbour, N. E. Behilil, P. Gibbs, G. Schierholz, and M. Teper, in *The Recursion Method and Its Applications*, edited by D. G. Pettifor and D. L. Weaire (Springer-Verlag, Berlin, 1985), pp. 149–164.
15. A. N. Burkitt, Liverpool preprint, 1986 (unpublished).
16. I. M. Barbour, P. Gibbs, K. C. Bowler, and D. Roweth, *Phys. Lett. B* **158**, 61 (1985).
17. D. Barkai, K. J. M. Moriarty and C. Rebbi, *Phys. Lett. B* **156**, 385 (1985); *Comput. Phys. Commun.* **36**, 1 (1985).
18. G. Meurant, *BIT* **24**, 623 (1984).
19. G. C. Batrouni, G. R. Katz, A. S. Kronfeld, G. P. Lepage, B. Svetitsky, and K. G. Wilson, *Phys. Rev. D* **32**, 2736 (1985).
20. Y. Oyangi, University of Tsukuba preprint ISE-TR-86-57, 1986 (unpublished).