Quark propagator on the Connection Machine *,†

T. Lippert a, K. Schilling a and N. Petkov b

a Fachbereich Physik, Universität Wuppertal, Gaussstrasse 20, 5600 Wuppertal 1, Germany
b Department of Computer Science, Rijksuniversiteit Groningen, Postbus 800, NL-9700 AV Groningen,
The Netherlands

Received 26 September 1991

Abstract


The computation of the fermion propagator in lattice Quantum Chromodynamics requires the solution of a large system of linear equations. We discuss and compare the structure, implementation and performance of two linear equation solvers, the Jacobi algorithm and the Conjugate Gradient algorithm, on the Connection Machine CM-2. We investigate the computer time needed for next neighbor communication versus the time required for floating point operations on 84 and 164 lattices. We compare the convergence behavior of Conjugate Gradient and Jacobi as applied to gauge configurations at β = 0.0 and 6.0.

Keywords. Lattice gauge theory; quark propagator; Connection Machine; relaxation methods.

1. Introduction – Physical aspects

Quantum Chromodynamics (QCD) is the theory of the strong interactions which bind together quarks, the conceptual basic constituents of matter, to form hadrons as they are observed in nature [1]. QCD cannot be solved analytically. A very promising way to evaluate such theories of elementary particles is offered by their discretization and computation on the lattice [2,3]. This method is called lattice gauge theory (LGT). LGT requires very large-scale scientific computing, with distributed memory in the many Gigabyte range and performance in the many Gigaflop domain. There is a general hope that the powerful parallel Teraflop machines of the nineties will lead to realistic results in this field of computational physics [4,5]. By now parallel machines have just come to dominate large-scale lattice gauge calculations [6–8]. In this context parallel computers like the Connection Machine 1 CM-2 [9] represent an important intermediate stage towards Teraflop machines because they allow to develop tools adapted to this coming era of scientific computing.

A key ingredient of LGT is the computation of quark propagation in a gluon field. The problem involves the frequent solution of a large but sparse system of linear equations.

Correspondence to: T. Lippert, Fachbereich Physik, Universität Wuppertal, Gaussstrasse 20, 5600 Wuppertal 1, Germany.

* Work supported by Deutsche Forschungsgemeinschaft grant Schi 257/1–4.
† Poster presented by Thomas Lippert at Parallel Computing 91.
1 The Connection Machine is a registered trademark of Thinking Machines Corporation.

0167-8191/92/$05.00 © 1992 – Elsevier Science Publishers B.V. All rights reserved
Relaxation methods are appropriate in this case. The solution vector, the propagator, is the starting point of hadron mass calculations in LGT, see [10]. A lot of work has been done in the past years to investigate and accelerate relaxation methods in the context of LGT [11]. Anew we address here a problem that will stay with us for many coming years namely the efficient implementation and application of linear equation solvers on parallel machines. We discuss the implementation of the simple Jacobi algorithm and the Conjugate Gradient algorithm on the Connection machine CM-2. We work on an 8K CM-2 machine, which has been installed at the University of Wuppertal in September 1990. This CM-2 has a configuration of 8k processors.

The paper is organized as follows: in Section 2 we begin with an illustration of the computational problem and introduce the algorithms to be considered, in Section 3 we describe the implementation of the linear equation solvers on the Connection machine and deal with the geometry of the problem and the communication patterns of the machine. Section 4 is devoted to performance measurements and bandwidth considerations of interprocessor communication and a comparison of the application of the two algorithms on real life LGT problems.

2. The computational problem

2.1. Quark propagator

The basic problem we have to solve is the numerical solution of the Euclidean Dirac equation [2]

\[ MX = \Phi. \] (1)

\( \Phi^a(x) \) is a vector, the space components of which reside on the discrete points \( x \) of a 4-dimensional lattice. \( x \) has the components \( (x_1, x_2, x_3, x_4) \). Therefore we deal with a 4-dimensional lattice gauge problem on a lattice of \( L^4 \) points. Figure 1 shows a 2-dimensional cross-section of the 4-dimensional lattice. On each lattice site there are attached 12 components of \( \Phi^a(x) \), which represent internal degrees of freedom. The ‘color’ index \( a \) ranges from 1 to 3, the Dirac index \( \alpha \) ranges from 1 to 4. Color is the strong interaction pendant to the electric charge of electromagnetism, the Dirac index respects the spin structure of the quarks. Thus we deal with a vector of \( 12 \times L^4 \) components. On each site \( x \) there are attached 4 additional objects called links \( U^{a,b}_\mu(x) \), \( \mu = 1, \ldots, 4 \), which are unitary \( 3 \times 3 \) matrices, acting in the color space. They describe the gluons which mediate the strong interactions. As these matrices represent a vector field they point in the four space directions to the points \( x + \mu, \mu = 1, \ldots, 4 \). The ‘Wilson fermion’ matrix in Eq. 1 is then given by

\[ M^{a,b}_\mu(x, y) = 1 - \kappa \sum_{\mu_1}^4 \left( (1 - \gamma^a_{\mu_1}) U^{a,b}_\mu(x) \delta_{x,y,\mu} \right. \]

\[ + \left. (1 + \gamma^a_{\mu_1}) U^{a,b}_{\mu_1}(x - \mu) \delta_{x,y,\mu} \right). \] (2)

The Dirac matrices \( \gamma^{a,\beta}_\mu \) act in the internal \( 4 \times 4 \) Dirac spin space. The parameter \( \kappa \) is a function of the quark mass.

2 Typical values of \( L \) in the case of QCD nowadays range from 8 up to 32. In the near future one intends to treat linear lattice extensions up to 128.
2.2. Jacobi algorithm

The Jacobi [12] algorithm is a simple linear equation solver. Since the matrix $M$ of Eq. 2 can be written as $M = 1 - \kappa D$, where $D$ contains only non-diagonal elements, the basic Jacobi iteration step is directly given by

$$X^{(i+1)} = \Phi + \kappa DX^{(i)}.$$  \hspace{1cm} (3)

The features of Jacobi are:

- In each iteration step one multiplication by the matrix $D$ is required,
- Two vectors have to be stored.

2.3. Conjugate Gradient

The Conjugate Gradient algorithm cannot be applied directly to our computational problem Eq. 1, because the matrix $M$ is non-hermitean. But one can instead solve the following system of linear equations:

$$M^\dagger MX = M^\dagger \Phi,$$  \hspace{1cm} (4)

where $M^\dagger$ is the hermitean conjugate of $M$. The Conjugate Gradient algorithm [12] consists of 3 basic steps:

$$X^{(i+1)} = X^{(i)} + \frac{(R^{(i)}, R^{(i)})}{(R^{(i)}, MX^{(i)})} P^{(i)}$$  \hspace{1cm} (5)

$$R^{(i+1)} = R^{(i)} - \frac{(R^{(i)}, R^{(i)})}{(R^{(i)}, MX^{(i)})} MX^{(i)}$$

$$P^{(i+1)} = R^{(i+1)} + \frac{(R^{(i+1)}, R^{(i+1)})}{(R^{(i)}, R^{(i)})} P^{(i)}.$$
The vectors $R^{(i)}$ and $P^{(i)}$ are auxiliary for the iteration and $P^{(0)} = R^{(0)} = \Phi - MX^{(0)}$. The features of Conjugate Gradient are:

- In each iteration step two multiplications by the matrix $M$ have to be performed,
- A minimum of four vectors has to be stored.

3. Implementation on the Connection Machine CM-2

3.1. Matrix-vector multiplication

The Jacobi and the Conjugate Gradient algorithms presented above require the multiplication of the vector $X$ by the matrix $M$ and by $M$ and $M^t$, respectively. Since QCD is a 4-dimensional theory with next neighbor interactions the matrix $M$ is sparse. To store and multiply the full matrix would be utterly inefficient. An efficient way to store $M$ and implement the iterations is to use the geometrical structure of the computational problem as presented in Section 2.1, i.e. to consider the problem on a 4-dimensional lattice. The interconnection network of the CM-2 processors is of 12 dimensional hypercube type. It allows the realization of a number of different commonly used communication patterns. In particular the processors can be connected in form of a 4-dimensional processor array, where each processing element communicates with two other processing elements, its nearest neighbors, in each of the four dimensions. This configuration fits perfectly the communication requirements of the problem considered. The implementation of the matrix-vector multiplication $MX$ on the Connection Machine is straightforward. Here we briefly comment on the implementation of this computation in CM-FORTRAN:

- The required geometry, a 4-dimensional grid is set by declaring data arrays with *four distributed* subscripts (dimensions). These dimensions correspond to the spatial dimensions of the problem. The other three dimensions of the 7-dimensional data array refer to the internal degrees of freedom attached to each site of the lattice (color and spin). These dimensions are realized as local within the memory of each processor. The difference between the distributed and the local dimensions is specified by a compiler directive (*layout*) where the former dimensions are referred to as 'news' and the latter as 'serial'. Here is a small part of the declarations of our code for the matrix-vector multiplication (16$^4$ lattice):

```fortran
parameter(icolor=3, icompl=2, idirac=4, ispace=4)
palrameter(nx=16, ny=16, nz=16, nt=16)
parameter(mcolor : icolor*icolor)
real, array(icolor, icompl, idirac, nx, ny, nz, nt)::phi, x
real, array(mcolor, icompl, ispace, nx, ny, nz, nt)::u
```

- The computation of $MX$ involves only interprocessor communication when done on a 4-dimensional grid. CM-FORTRAN provides a function 'shift' (circular shift) for this purpose. The following statement, for instance, describes the transport of the variable $\Phi$, located at $x + \mu$, to the site $x$:

```fortran
phi_s(icol, icom, idir, ::, ::, ::)=
cshift(phi(icol, icom, idir, ::, ::, ::), mu, 1)
```
3.2. Global operations

The Conjugate Gradient algorithm as described in Section 2.3 requires the computation of scalar products of vectors. Microprogrammed routines on the Connection Machine allow for fast global operations such as summing up data distributed among the processor elements to perform scalar products (the full interconnection pattern of the hypercube is used for this purpose):

\[ a = \text{sum}(x \times x) \]

![8^4 LATTICE](image)
![16^4 LATTICE](image)

Fig. 2. (a) Time per iteration for Jacobi and Conjugate Gradient on the $8^4$ lattice. In the case of Conjugate Gradient the matrix-vector multiplication dominates the global operations. (b) Time per iteration for Jacobi and Conjugate Gradient on the $16^4$ lattice.
4. Results and conclusions

4.1. Results of time measurements

In both relaxation algorithms, Jacobi and Conjugate Gradient, nearly identical matrix multiplications have to be carried out. The basic Jacobi step, Eq. 3, requires one multiplication by the matrix \( D, M = 1 - \kappa D \), whereas Conjugate Gradient requires 2 multiplications, one by the matrix \( M \) and one by \( M^T \). For that reason it is not surprising that the time amount the Conjugate Gradient algorithm spends on matrix multiplication is twice the time amount spent by Jacobi, on both 8\(^4\) and 16\(^4\) lattices, see Figs. 2 (a) and (b). The basic Jacobi step does not require global operations, whereas this is the case for Conjugate Gradient. As can be seen in Figs. 2(a) and (b) the matrix-vector multiplication dominates the rest of the program.

In order to speedup the relaxation methods further one has to concentrate on matrix-vector multiplication. The Connection Machine CM-2 as installed in Wuppertal has 8K bit processors and 256 64-bit floating-point accelerating units. For numerical computations, the latter are more important so that memory is allocated and code is compiled in such a way that the floating-point accelerators are considered as the primary system units. This is done in the so called slice-wise mode of using the CM-2. Because of the vector register structure of the floating-point units, it is more advantageous to realize at least four identical copies of each operation to be executed. For this reason each of the floating-point units realizes four ‘physical’ processors. As a result, in a slice-wise mode an 8K CM-2 looks to the user like a parallel computer of 1K physical processors. The size of this machine is evidently smaller than the size of our problem, if only one physical processor is to be provided for each lattice site. In this case, each physical processor realizes a corresponding number of virtual processors so that each lattice site can be assigned to a different virtual processor. In our case the VP(= virtual processor) ratio was 4 on the 8\(^4\) lattice and 64 on the 16\(^4\) lattice. This is the reason why (for both algorithms) the total amount of time per iteration is approximately 10 times larger on the 16\(^4\) lattice as compared to the 8\(^4\) lattice, see Figs. 3(a) and (b), and not 16 times as expected naively. At higher VP ratio the communication takes place predominantly within the processors. An important issue of Figs. 3(c) and (b) is the fact that considerable time is spent on communication: in the matrix-vector multiplication routine 14 floating point operations come to one cshift communication operation. However, on the 8\(^4\) lattice more than half the time spent on one iteration is used for nearest-neighbor communication. This time amount is reduced on the 16\(^4\) lattice to 37%. We reached a performance of 400 Mflops on our 8k machine for the 16\(^4\) lattice.

4.2. Real-life convergence behavior

The hopping parameter \( \kappa \) appearing in Eq. 2 is related to the bare quark mass. A reasonable definition is given by

\[
m_q \approx \frac{1}{2\kappa} - \frac{1}{2\kappa_c}.
\]

The physically interesting case is the limit of small quark mass at some critical \( \kappa = \kappa_c \). On any given gauge configuration this parameter can be found very accurately as shown in [13]. Figure 4 presents the number of iterations for both algorithms, Jacobi and Conjugate Gradient, which is necessary to achieve a certain accuracy for the solution, as function of the hopping parameter \( \kappa \). Here the background field \( (U_b, x) \) was a ‘hot’ configuration at \( \beta = 1/T = 0 \). \( T \) is the temperature of the system. Throughout the \( \kappa \) range Jacobi shows a faster convergence
behavior than Conjugate Gradient down to very small quark masses of $m_q = 0.01$. At $\kappa_c$ the matrix acquires a zero eigenvalue reflected by the fact that both iteration numbers diverge there. We imposed the same condition on both algorithms. The iteration stopped when a residual of

$$r = \frac{\| \Phi - MX \|^2}{\| X \|^2}$$

was reached. A residual of $r = 10^{-9}$ has turned out to be accurate enough in practical lattice gauge applications. By going to ‘realistic’ configurations at $\beta = 6.0$ the picture changes. There is a crossover, where the number of Conjugate Gradient iterations becomes smaller than the number of Jacobi iterations going to smaller quark mass. In our case this occurs at a quark mass of $m_q = 0.035$ as shown in Fig. 5. At a quark mass of $m_q = 0.01$ we see in Fig. 6 that the residual $r$ for the Conjugate Gradient algorithm goes monotonically down as function of the...
iteration number, whereas the Jacobi algorithm is not able to reduce $r$ to a smaller value than $10^{-3}$.

4.3. Concluding remarks

Conjugate Gradient turns out to become substantially superior to Jacobi at very small quark masses, but the question, which algorithm should be applied in lattice gauge computations, cannot be answered uniquely, since at quark masses used currently in most applications the Jacobi algorithm is still faster than the Conjugate Gradient algorithm. Attempts to improve the implementation of such relaxation algorithms on the Connection Machine should concen-
brate on an improvement of the matrix-vector multiplication in order to reduce the time amount spent on communication. Successful ideas in this direction have been presented in [14].

References