FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Running a Code for Lattice Quantum Chromodynamics Efficiently on CRAY T3E Systems

Norbert Attig, Stephan Güsken[b], Pierre Lacock[a],
Thomas Lippert[b], Klaus Schilling[a,b],
Peer Ueberholz[b], Jochen Viehoff[b]

FZJ-ZAM-IB-9823

Mai 1998

(letzte Änderung: 11.05.98)

Preprint: Proceedings of High Performance Computing and Networking Europe 1998 (HPCN), Amsterdam, Niederlande, 21. - 23. April 1998, pp. 183-192

([a]) Höchstleistungsrechenzentrum (HLRZ), c/o Forschungszentrum Jülich GmbH, D-52425 Jülich, Germany
([b]) Fachbereich Physik, Universität Wuppertal, D-42097 Wuppertal, Germany

Running a Code for Lattice Quantum Chromodynamics Efficiently on CRAY T3E Systems

N. Attig[1⋆], S. Güsken[3], P. Lacock[2], Th. Lippert[3], K. Schilling[2,3], P. Ueberholz[3], and J. Viehoff[3]

[1] Zentralinstitut für Angewandte Mathematik (ZAM), Forschungszentrum Jülich GmbH, D-52425 Jülich, Germany
[2] Höchstleistungsrechenzentrum (HLRZ), c/o Forschungszentrum Jülich GmbH, D-52425 Jülich, Germany
[3] Fachbereich Physik, Universität Wuppertal, D-42097 Wuppertal, Germany

Abstract. Computing physical quantities in lattice quantum chromodynamics means solving huge systems of linear equations ($\mathcal{O}(10^7)$ equations). Efficient parallel Krylov subspace solvers play a vital role in the solution of these systems. We present a detailed analysis of the performance of the stabilized biconjugate gradient algorithm with preconditioning on massively parallel CRAY T3E systems.

1 Lattice Gauge Theory computations

The numerical investigation of quantum chromodynamics (QCD) on a four-dimensional space-time grid is one of the grand challenges in high-performance scientific computing [1, 2]. QCD is considered to be the fundamental theory of strongly interacting particles. After 20 years of research, the strong coupling regime of QCD [3] still has not been solved in a non-perturbative analytical approach, and it is by now widely believed that the numerical treatment of the theory on the lattice using very fast parallel supercomputers is the only viable scheme to extract quantitative physical results [4]. The results from lattice gauge theory (LGT) simulations are urgently needed as theoretical input for current and future accelerator experiments that attempt to observe new physics beyond the *Standard Model* of elementary particle physics [5].

LGT computes functional integrals using Monte Carlo methods known from statistical physics [6]. A representative ensemble of field configurations is generated by a Markov process (simulation phase). These configurations are subsequently analysed by composing (and averaging over) hadronic correlators from quark Green's functions (analysis phase). The correlators then serve to extract physical observables like hadron masses and decay constants.

2 The numerical problem and computational effort

The enormous amount of floating-point operations which have to be calculated in the stochastic simulation in order to achieve statistically significant physical results has

⋆ Talk presented by N. Attig

led to a concentration of activities in this field of research. Several collaborations in Japan (CP-PACS), the U.S. (MILC), United Kingdom (UKQCD), and Italy (APE) are investigating QCD systematically either on special-purpose or commercial high-end parallel supercomputer hardware performing with several hundreds of Gflops.

The large-scale projects SESAM[1] [7] and TχL[2] [8], a common effort of German and Italian high-energy physicists to study full QCD with two flavours of dynamical Wilson fermions, take place on APE100/Quadrics systems in Zeuthen and Rome and on CRAY T3E systems installed at the Research Centre Jülich [9]. In the following, the physical model of this application which is common many QCD investigations and its numerical implementation are discussed in some detail.

In the simulation phase a representative ensemble of gauge field configurations on a space-time lattice is generated using a Hybrid Monte Carlo algorithm (HMC) on the parallel supercomputer APE100/Quadrics. The SESAM project has simulated on a $16^3 \times 32$ lattice while the TχL production is still ongoing on a $24^3 \times 40$ lattice. The APE100 systems are equipped with 256 and 512 nodes respectively. Each node has a theoretical peak performance of 50 Mflops, the HMC algorithm reaches a sustained performance of 50% on a 512-node machine. The total computer time used so far adds up to 250 Teraflop hours.

The analysis of these gauge field configurations is performed on the CRAY T3E systems at the Research Centre Jülich. The most time-consuming part of this analysis phase is the frequent computation of the QCD Green's functions as solutions of huge sparse systems of linear equations [10] on each gauge field configuration

$$M^{c,c\prime}_{\alpha,\alpha\prime}(x,x\prime)\psi^{c\prime}_{\alpha\prime}(x\prime) = \phi^{c}_{\alpha}(x) \ , \tag{1}$$

where ϕ is some input vector, M is the Dirac fermion matrix and ψ the required solution. The solution (quark propagator) is then used to investigate the properties of hadrons by constructing the appropriate hadronic correlators. For the choice of lattice fermions considered here, namely the so-called Wilson fermions, (1) has the explicit form

$$\begin{aligned} \psi(x)^{c}_{\alpha} - \kappa \Bigg(& \sum_{\mu=1}^{4} U_{-\mu}(x)^{c,c\prime} m_{-\mu,\alpha,\alpha\prime} \psi(x-\mu)^{c\prime}_{\alpha\prime} \\ & + \sum_{\mu=1}^{4} U_{\mu}(x)^{c,c\prime} m_{\mu,\alpha,\alpha\prime} \psi(x+\mu)^{c\prime}_{\alpha\prime} \Bigg) = \phi(x)^{c}_{\alpha} \ , \end{aligned} \tag{2}$$

where $U_{\mu}(x)^{c,c\prime}$ is an SU(3) matrix for the gauge field (gluon), computed in the simulation phase, $\psi(x)^{c}_{\alpha}$ is a 4×3 complex matrix for the particle (quark) and m is a 4×4 matrix containing the spin components. The index x runs over all space-time lattice points (4 dimensions). In particular, M is a non-hermitean complex sparse matrix with the following structure:

– 1 in the diagonal.

[1] **S**ea Quark **E**ffects on **S**pectrum **a**nd **M**atrix Elements

[2] **T**owards the **chi**ral **L**imit

– 8 non-diagonal 12×12 matrices (nearest neighbours in space-time).
– Each 12×12 matrix is a tensor product of an $SU(3)$ matrix U times the Dirac 4×4 matrix m.

In order to solve this system of linear equations we apply Krylov subspace solvers[3], complemented by preconditioning techniques.

3 Inversion algorithms

The class of Krylov subspace iterative methods for solving (1) is characterised by the generic form

1. choose an initial guess ψ^0 and set $r^0 = \phi - M\psi^0$,
2. compute iteratively ψ^m of the form $\psi^m = \psi^0 + q_{(m-1)}(M)r^0$.

Here $q_{(m-1)}(M)$ is a polynomial of degree $< m$. Examples are the conjugate gradient or the minimal residual algorithms. At present, the state-of-the-art method for the fermion matrix is the stabilised biconjugate gradient (BiCGstab) algorithm [11].

Further improvement of the inversion algorithm can be obtained by using preconditioning techniques, which should reduce the number of iterations and the computing time necessary to achieve a given accuracy. To precondition (1) we take two non-singular matrices V_1 and V_2 which act as left and right preconditioners respectively, i.e. we replace (1) by

$$M\psi = \phi \;\; \longrightarrow \;\; V_1^{-1} M V_2^{-1} \widetilde{\psi} \; = \; \widetilde{\phi} \tag{3}$$

The matrix $V = V_1 V_2$ is called the preconditioner. The efficiency of the preconditioning method depends on how good an approximation V is of M, as well as the computational overhead entailed in the method: solving systems with V_1 and V_2 should be cheap.

Currently, the most efficient way is to use symmetric successive over-relaxation (SSOR) preconditioning (for details see [12, 13]):

$$V_1 = I - L; \qquad V_2 = I - U \tag{4}$$

with $M = I - L - U$, I the diagonal part, L the strictly lower triangular part and U the strictly upper triangular part. The convergence rate still depends on the ordering of the sites in the lattice.

In case of even-odd ordering, where all even sites are labeled before the odd ones on each processor like a checkerboard (see Fig. 1), the preconditioned matrix $V_1^{-1} M V_2^{-1}$ can be computed explicitly. For larger sub-systems we use parallel SSOR preconditioning. In particular, the sub-blocks have the same size as the partitions of the lattice assigned to a processor. Thus, in parallel SSOR, the parallelism is adapted to the parallel system. The use of the Eisenstat trick is essential:

$$V_1^{-1} M V_2^{-1} = (I - U)^{-1} + (I - L)^{-1}(I - (I - U)^{-1}) \tag{5}$$

[3] The coefficients of the regular sparse matrix are related to a stochastic background gauge field, hence multigrid methods are not efficient here!

In this way the SSOR preconditioning is about as expensive as $M\psi$, because $(I-L)x = y$ (forward solve) and $(I-U)x = y$ (backward solve) are easy to solve.

The largest vector computed in the SESAM and TχL projects is of a size of 12 Mwords. This number is given by the volume factor (e.g. $24^3 \times 40$), the four Dirac components, three colour components, and a factor 2 due to complex elements. As a consequence, we are faced in the analysis phase, where about ten thousand configurations have to be used for calculating the propagators, with a requirement of about 10 Teraflop hours computing time on the order of 1 Terabyte of data. It is therefore evident that optimization of the QCD analysis codes on parallel supercomputers is mandatory in order to carry out ambitious computations of this kind efficiently.

4 CRAY T3E architecture

The CRAY T3E, which is the second generation of Cray Research MPP systems, is the ideal machine for this kind of application. It is a fully scalable MIMD system with distributed memory and global address space. The application processing elements (application PE) are connected by a network which has the topology of a three-dimensional torus. The system is self-hosting and scalable from 8 to 2048 PEs. For every 16 application PE an operating system node (OS PE) is required.

Each processing node is equipped with a DEC Alpha EV5 microprocessor. In the CRAY T3E-600 the processor clock rate is 300 MHz, while in the T3E-900 the clock rate is 450 MHz. The instruction rate is up to 4 per clock cycle (2 floating-point, 2 integer/logic) which results in a peak performance of 600 Mflops for the T3E-600 and 900 Mflops for the T3E-900, respectively. The microprocessor uses IEEE 64-bit arithmetic. Furthermore, each microprocessor has 3 on-chip caches: one 8 KByte data cache (Dcache), segmented into 256 cache lines with 4 64-bit words each, one 8 KByte instruction cache (Icache) with the same segmentation and one 96 KByte secondary cache (Scache), separated in 3 associative sets of 512 cache lines each, serving either Dcache and Icache. The load latencies are 2 clock periods (CP) from Dcache and 8 to 10 CP from Scache, the load bandwidths are 2 loads/CP from Dcache or Scache, or 1 store/CP.

Table 1. Characteristics of the CRAY T3E systems installed at the Research Centre Jülich.

	T3E-600	T3E-900
Number of application PE	512	256
Processor clock	300 MHz	450 MHz
Mflops (peak per PE)	600	900
Main memory (per PE)	128 MByte	128 MByte
Primary data cache (per PE)	8 KB	8 KB
Secondary cache (per PE)	96 KB	96 KB
Memory bandwidth	1200 MByte/s	1200 MByte/s

Each PE is equipped with a local memory (DRAM) of 128, 256 or 512 MByte. To maximize local memory bandwidth, 6 data stream buffers are available. They do a prefetch from DRAM to cache for vector-like data references leading to a better performance. Additionally, a set of 512 off-chip memory mapped external (E) registers can be used, which directly load/store into/from the cache registers from/to the global memory.

For QCD simulations it is sometimes sufficient to perform the calculations in half-precision. On the T3E operations can only be performed in 64-bit precision due to the 64-bit arithmetic of the processor. Nevertheless, it is possible to use half-precision variables (32 bit). Before being processed, they are extented to 64 bit, then processed and finally reduced again to 32 bit. So it is not possible on the T3E to increase the number of operations per clock period but one can profit from a better cache usage.

The calculations are performed at the Research Centre Jülich on two CRAY T3E systems: on a 512-node T3E-600 and and on a 256-node T3E-900. Table 1 summarizes the main characteristics of these machines installed at Jülich.

5 Implementing the inversion algorithm on the CRAY T3E

The strategy for computing the quark propagators is as follows. First, a four-dimensional processor grid with n_{proc} processors is defined, where $p_x \cdot p_y \cdot p_z \cdot p_t = n_{proc}$. Then, the system is partitioned by dividing the four-dimensional lattice of size $N_x \cdot N_y \cdot N_z \cdot N_t = V$ into local sublattices of size $n_x \cdot n_y \cdot n_z \cdot n_t = V_{loc}$ with $n_i = N_i/p_i$.

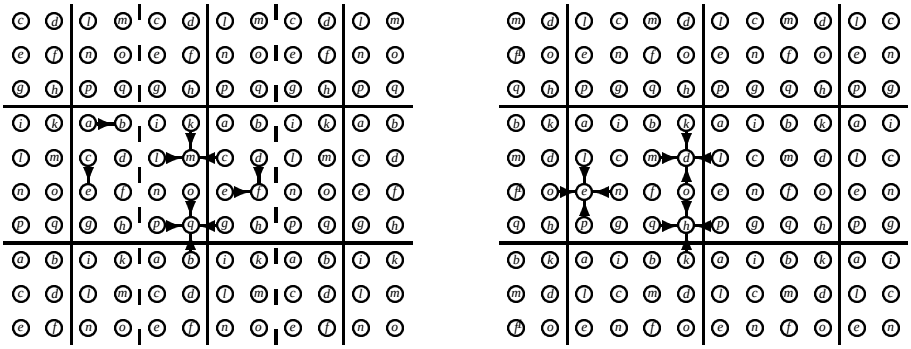

Fig. 1. Blocked local lexicographic (left) and even-odd ordering (right)

The speed of the algorithm and the convergence depends on the layout and the ordering of the fields on each processor. The cache is used best by running the space-time lattice index last. Two efficient orderings of the lattice sites are the even-odd or checkerboard ordering and the local lexicographic ordering. The first one is easy to vectorize or parallelize, because the even (odd) points are independent from each other (Fig. 1), while the second one is the most efficient way with regard to the number of iterations [12]. In the forward (backward) solves the neighbours before (after) the current site have to be considered, thus one needs to communicate the results of the points on the boundary as soon as they are computed. In order to reduce the overhead for the communication we introduce a further blocking within local lexicographic (BLL) ordering on each processor (Fig. 1), which accelerates the Mflop rate by a factor of 3.

Cutting the local sub-lattice once in order to decouple the local data elements is sufficient to achieve efficient pipelined communication. Hence, the blocking is called one-dimensional. We remark that it would not be useful to perform two-dimensional

blocking (leading to four distinct local partitions) as the efficiency of local lexicographic SSOR depends on the local lattice size and decreases for smaller local lattices.

In Fig. 1 the ordering of the sites corresponds to the alphabetic ordering $a - q$. The arrows illustrate the points, which have to be taken into account for the update of one of the sites in a two-dimensional analogue.

The program starts with an input phase. The gauge fields ($\{U\}$, 159 MByte on the $24^3 \times 40$ lattice) and sometimes the starting vectors (ϕ, ψ_o, 637 MByte each on the $24^3 \times 40$ lattice) are read in (in half-precision, because they were generated on the APE100 in 32-bit precision). The I/O is implemented in parallel by letting the p_t processors read n_t files and send the data to those processors belonging to the same time-slices. Making use of user-triggered disk-striping by distributing the files over different partitions, a performance of up to 120 MBytes/s can be reached. When the data are read in and distributed, they finally have to be converted to 64 bit. Now the inversion algorithm can start.

One iteration step for solving the linear equation $M\psi = \phi$ involves four basic operations:

1. communication of the boundary values,
2. vector inner products,
3. AXPY vector operations ($y = y + \alpha * x$),
4. matrix-vector multiplications.

For the communication, the Cray-specific shared memory routines (*shmemput* and *shmemget*) are used which transfer the data directly between the local and remote memories. For standard message-passing routines like MPI about 10% of the total computer time is spent on communication. With shared memory routines, communication is a factor of 2 to 3 faster for our code.

In case of the inner products and AXPY operations, Fortran 90, BLAS, and assembler routines can be compared. We found that for our application there is nearly no difference between the different routines, but on the Fortran level, it is worthwhile taking care that all cache entries are used as often as possible and padding the arrays to avoid cache conflicts. E.g. the operation $y = x + \beta * y$ followed by $z = x + \alpha * y$ runs with 86 Mflops (on the T3E-900) while the performance goes up to 175 Mflops if both operations are done in one step $z = x + \alpha * (z + \beta * y)$.

Around 80% of the computer time is spent for the forward/backward solve or, in case of even-odd ordering, for the matrix-vector multiplication, which from the programming point of view is essentially the same. Therefore, we spent most of our effort on optimizing this operation. Looking at the fermion matrix in more detail, the number of $SU(3)$-multiplications can be reduced by a factor of 2 by noting that the 4×4 matrix $m_{\alpha,\alpha\prime}$ can be split into the matrices p of size 4×2 and q of size 2×4. This is known as the Wilson trick. The matrix M of (2) takes the form

$$\begin{aligned} \delta^{c,c\prime}_{\alpha,\alpha\prime}\delta_{x,x\prime} - \kappa \sum_{\mu=1}^{4} \sum_{\alpha\prime\prime=1}^{2} \Big(& q_{-\mu,\alpha,\alpha\prime\prime} U^{+c,c\prime}_{\mu}(x-\mu) p_{-\mu,\alpha\prime\prime,\alpha\prime} \delta_{(x-\mu),x\prime} \\ + \quad & q_{\mu,\alpha,\alpha\prime\prime} U^{c,c\prime}_{\mu}(x) p_{\mu,\alpha\prime\prime,\alpha\prime} \delta_{(x+\mu),x\prime} \Big) = M^{c,c\prime}_{\alpha,\alpha\prime}(x,x\prime) \ . \end{aligned} \tag{6}$$

For further discussion of the parallel implementation, let us consider the last term

$$\sum_{\alpha\prime=1}^{4}\sum_{\alpha\prime\prime=1}^{2}\sum_{c\prime=1}^{3} q_{\mu,\alpha,\alpha\prime\prime}U_{\mu}^{c,c\prime}(x)p_{\mu,\alpha\prime\prime\alpha\prime}\chi(x+\mu)_{\alpha\prime}^{c\prime} = y(x)_{\alpha}^{c}\ . \tag{7}$$

A standard way to compute (7) in four steps is:

1. – Communicate the boundary of χ (done globally for the whole vector χ).
   – Compute $p_{\mu,\alpha\prime\prime\alpha\prime}\chi(x+\mu)_{\alpha\prime}^{c\prime} = \chi(x)\prime_{\alpha\prime\prime}^{c\prime}$ for a specific site x.
   – Compute $U_{\mu}(x)^{c,c\prime}\chi(x)\prime_{\alpha\prime\prime}^{c\prime} = \chi(x)\prime\prime_{\alpha\prime\prime}^{c}$ for a specific site x.
   – Compute $q_{\mu,\alpha,\alpha\prime\prime}\chi(x)\prime\prime_{\alpha\prime\prime}^{c} = y(x)$ for a specific site x.

Using even-odd ordering there is another possible way to proceed in five steps:

2. – Compute $p_{\mu,\alpha\prime\prime\alpha\prime}\chi(x)_{\alpha\prime}^{c\prime} = \omega(x)_{\alpha\prime\prime}^{c\prime}$ for all sites x.
   – Communicate $\omega(x)$ to its neighbour $x+\mu$.
   – Gather result in $\chi(x)\prime$.
   – Compute $U_{\mu}(x)^{c,c\prime}\chi(x)\prime_{\alpha\prime\prime}^{c\prime} = \chi(x)\prime\prime_{\alpha\prime\prime}^{c}$ for all sites x.
   – Compute $q_{\mu,\alpha,\alpha\prime\prime}\chi(x)\prime\prime_{\alpha\prime\prime}^{c} = y(x)$ for all sites x.

At first glance one would expect case 1 to be better suited to exploit cache reuse, while case 2 should be able to exploit the stream buffers more effectively. This is confirmed by the single-processor results listed in Tab. 2, where one clearly sees that the multiplication $U\chi\prime$ has a higher Mflop rate in case 2, while for the final multiplication case 1 has the higher rate since it has better cache usage. From this performance comparison it can be concluded that at least on a Fortran programming level, case 1 is the better choice.

Table 2. Single-processor performance in Mflops. Half-precision results are given in parentheses.

	T3E-600		T3E-900	
	case 1	case 2	case 1	case 2
$p_{\mu,\alpha\prime\prime\alpha\prime}\chi_{\alpha\prime}^{c\prime}$	12 (20)	13 (27)	14 (25)	14 (36)
$U_{\mu}^{c,c\prime}\chi\prime_{\alpha\prime\prime}^{c\prime}$	70 (73)	94 (124)	96 (100)	138 (169)
$q_{\mu,\alpha,\alpha\prime\prime}\chi\prime\prime_{\alpha\prime\prime}^{c}$	26 (30)	20 (45)	24 (40)	21 (59)
complete computation of (7)	48 (63)	39 (61)	63 (90)	48 (80)
	overall performance of BiCGstab with SSOR			
BLL ordering	59 (75)		67 (130)	
even-odd ordering	62 (80)		85 (106)	

Also, as expected, nearly all floating-point operations are generated by the complex 3×3 matrix-vector multiplication with $U_{\mu}(x)$. On the other hand, the multiplications with p and q are mainly load and store, because m has a simple form, e.g. for μ=2:

$$p_2 = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

These multiplications are hard to optimize because the performance is limited by the bandwidth between memory and cache.

Comparing the results on the T3E-600 with the ones obtained on the T3E-900 one clearly sees that the application speeds up very well for those parts which are not memory-bounded, e.g. $U_\mu^{c,c\prime}\chi\prime_{\alpha\prime\prime}^{c\prime}$. If mainly load/store operations are performed ($p_{\mu,\alpha\prime\prime\alpha\prime}\chi_{\alpha\prime}^{c\prime}$ or $q_{\mu,\alpha,\alpha\prime\prime}\chi\prime\prime_{\alpha\prime\prime}^{c}$), the code cannot gain from the increased processor speed due to the bottleneck of memory access time.

6 Single-processor optimization

The performance obtained up to now implies automatic (Fortran 90 compiler options for intrinsic vectorisation and unrolling) and manual optimizations (index permutations, loop interchange to improve cache usage, common block padding to avoid Scache conflicts and keeping the number of active data streams to 6 or less wherever possible). To improve the performance further, we decided to use assembler programming for the kernel routines. As a first step, the multiplication with U was rewritten in assembler. The performance results are listed in Tab. 3. A significant improvement can be observed in cases 1 and 2, indicating that the hand-written assembler routine makes much better use of the registers.

Table 3. Single-processor performance of assembler code in Mflops. Half-precision results are given in parentheses.

	T3E-600		T3E-900	
	case 1	case 2	case 1	case 2
$U_\mu^{c,c\prime}\chi\prime_{\alpha\prime\prime}^{c\prime}$	150 (158)	168 (240)	185 (207)	221 (320)
complete computation of (7)	67 (81)	45 (72)	89 (116)	52 (98)

Further improvements are expected by extending the assembler programming to more steps needed for the calculation of (7), especially for case 2 where the inner loop over the sites x implies a very efficient use of the streams buffers. It might be possible that a hand-written assembler program can benefit much more from this effect than the corresponding Fortran routine. Our attempt is to speed up case 2 as much as possible in order to finally obtain a BiCGstab algorithm with even-odd odering which converges faster in real time than the corresponding one with BLL ordering. For case 1 we do not expect any further major acceleration because only site after site can be calculated.

One possibility is to combine the last three steps of case 2 in one assembler program. In this case, we measured a performance of 108 Mflops (T3E-600), and 120 Mflops (T3E-900) respectively. However, the assembler code now contains the gather instruction of the nearest neighbour sites as well as the multiplication with the matrix q, which have both very low Mflop rates. Combining these steps with the matrix-vector multiplication leads to more effective use of cache resident data and eliminates some of the intermediate steps (e.g. explicitly calculating $\chi\prime\prime$). This increases the performance from 52 (98) Mflops (T3E-900) to 68 (120) Mflops in the complete computation of (7).

Furthermore, it is possible to combine all steps of case 2 in one assembler routine, except the communication, which is done globally as in case 1. In this case the single-processor performance of the assembler routine reached 85 (140) Mflops on the T3E-600 and 92 (190) Mflops on the T3E-900. This disapointing performance is caused

by the extra load and store instructions that have been added to the assembler routine. To understand why this happens one has to consider the architecture of the EV5 RISC chip: even though four instructions are performed per clock period, some instructions cannot be issued simultaneously in the same clock period (e.g. loads and stores). In addition, only one floating-point multiply or add may be issued in the same clock period. These factors make it very difficult to balance the requirements of the processor with the practical nature of the problem, when the assembler code becomes saturated with load and store instructions, while the number of floating-point operations only increases marginally. On the other hand, a performance improvement of around 20% for the complete computation of (7) can be realized. Our currently best performance values for the BiCGstab algorithms are summarized in Tab. 4.

Table 4. Improved overall performance (Mflops per processor) of the BiCGstab algorithm with SSOR preconditioning. Half-precision results are given in parentheses.

	T3E-600		T3E-900	
	case 1	case 2	case 1	case 2
BLL ordering	76 (95)		88 (127)	
even-odd ordering	65 (80)	87 (115)	86 (110)	103 (148)

These data clearly show that the BiCGstab algorithm with even-odd ordering in the implementation of case 2 is only slightly faster than the one with BLL ordering in the implementation of case 1. As mentioned in Chap. 5, the number of iterations with BLL ordering is roughly one half compared to even-odd ordering. Therefore, in real time BLL ordering is still about 80% faster than even-odd ordering.

7 Summary

We have presented a discussion of optimized codes for lattice QCD on CRAY T3E systems. To improve the state-of-the-art BiCGstab inversion algorithm, we implemented SSOR blocked lexicographic preconditioning which can be efficiently implemented on parallel machines. Our optimization efforts were concentrated on the matrix-vector multiplications since most of the processing time is spent in these routines. It was demonstrated that assembler programming of kernel routines improves the overall performance substantially. In assembler programming, we found that one of the main difficulties in optimizing codes on the CRAY T3E processing nodes is the efficient implemention of instruction scheduling with respect to the complex memory system (on-chip caches, stream buffers, E-registers), and the processor's capability to perform four instructions per clock period.

The final performance of 20% of the peak speed on the T3E-600 with half-precision data is consistent with the well known limitations of direct mapped cache architectures. Also on other popular machines used by the QCD community like the APE100/Quadrics or CP-PACS, assembler programming and register optimization is mandatory in order to achieve relative performances between 50 and 60% of their peak speed. However, these systems profit from hardware optimizations relevant for efficient QCD matrix-vector multiplications.

Acknowledgements

The authors gratefully acknowledge the computer time granted by the HLRZ on the CRAY T3E systems of the Research Centre Jülich. They would like to thank E. Anderson of Silicon Graphics / Cray Research for his advice and efforts with respect to the assembler programming and R. Vogelsang from Silicon Graphics GmbH / Cray Research for his continuous support.

References

1. Wilson, K.G.: Grand Challenges to computational science, Future Generation Computer Systems **5** (1989) 171.
2. Aoki, S.: The QCD Teraflops Project, Int. J. Mod. Phys. **C2** (1991) 829.
3. Zerwas, P.M., Kastrup, H.A. (eds.), *QCD 20 years later* (World Scientific, Singapore, 1992).
4. DeGrand, T.: Lattice Gauge Theory for QCD, COLO-HEP-378 (hep-ph/9610391), October 1996.
5. Particle data group, Phys. Rev. **D54** (1996) 1.
6. Creutz, M. in *Quarks, Gluons and Lattices* (Cambridge University Press, Cambridge, 1983).
7. Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Luo, X., Ritzenhöfer, G., Schilling, K., Siegert, G.: QCD with dynamical Wilson fermions - first results from SESAM, Nucl. Phys. **B** (Proc. Suppl.) **47** (1996) 386;
   Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Ritzenhöfer, G., Schilling, K., Siegart, G., Wachter, A.: First Evidence of N_f-Dependence in the QCD Interquark Potential, Phys. Lett. **B383** (1996) 98;
   Eicker, N., Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Ritzenhöfer, G., Schilling, K., Siegart, G., Spitz, A., Ueberholz, P., Viehoff, J.: Evaluating sea quark contributions to flavour singlet operators in lattice QCD, Phys. Lett. **B389** (1996) 720.
8. Conti, L., Eicker, N., Giusti, L., Glässner, U., Güsken, S., Hoeber, H., Lippert, Th., Martinelli, G., Rapuano, F., Ritzenhöfer, G., Schilling, K., Siegert, G., Spitz, A., Viehoff, J.: Full QCD with dynamical Wilson fermions on a $24^3 \times 40$ lattice - A feasibility study, Nucl. Phys. **B** (Proc. Suppl.) **53** (1997) 222;
   Lippert, Th., Bali, G., Eicker, N., Giusti, L., Glässner, U., Güsken, S., Hoeber, H., Lacock, P., Martinelli, G., Rapuano, F., Ritzenhöfer, G., Schilling, K., Siegert, G., Spitz, A., Ueberholz, P., Viehoff, J.: SESAM and TχL Results for Wilson Action: A Status Report, Nucl. Phys. **B** (Proc. Suppl.) **60A** (1998) 311.
9. Attig, N.: QCD on Parallel Computers at the HLRZ Supercomputing Center, Proceedings of Physics Computing '96 (PC'96), Krakow, Poland, (1996) 536.
10. Lippert, Th., Schilling, K. and Petkov, N.: Quark Propagator on the Connection Machine, Parallel Computing **18** (1992) 1291.
11. Frommer, A., Hannemann, V., Lippert, Th., Nöckel, B., Schilling, K.: Accelerating Wilson Fermion Matrix Inversions by Means of the Stabilized Biconjugate Gradient Algorithm, Int. J. Mod. Phys. **C5** (1994) 1073.
12. Fischer, S., Frommer, A., Glässner, U., Lippert, Th., Ritzenhöfer, G., Schilling, K.: A Parallel SSOR Preconditioner for Lattice QCD, Comp. Phys. Comm. **98** (1996) 20.
13. Eicker, N., Frommer, A., Hoeber, H., Lippert, Th., Medecke, B., Schilling, K., Weufen, G.: Parallel SSOR preconditioners for Improved Actions in Lattice Field Theory, to appear in Proceedings of PARCO'97, Bonn, Germany, (1997).