Fortran MPI Checkerboard Code for SU(3) Lattice Gauge Theory I

Bernd A. Berg

Department of Physics, Florida State University, Tallahassee, FL 32306-4350, USA

Abstract

We document Fortran MPI checkerboard code for Markov Chain Monte Carlo simulations of pure SU(3) lattice gauge theory with the Wilson action on a D-dimensional double-layered torus. This includes the usual torus with periodic boundary conditions as an optional case. We use Cabibbo-Marinari heatbath checkerboard updating. Parallelization on sublattices is implemented in all D directions and can be restricted to less than D directions. The parallelization techniques of this paper can be used for any model with interactions of link variables defined on plaquettes.

Program Summary

Program title: STMC2LSU3MPI.
Program identifier: Not yet available.
Program summary URL: Not yet available.
Program available from: Temporarily from URL http://www.hep.fsu.edu/~berg/research
Programming language: Fortran 77 with MPI extensions.
Computer: Any capable of compiling and executing Fortran 77 code with MPI extensions.

Key words: Markov Chain Monte Carlo, Parallelization, MPI, Fortran, Checkerboard updating, Lattice gauge theory, SU(3) gauge group.
PACS: 02.70.-c, 11.15.Ha

1. Introduction

Moore’s law [1] appears to be dead. Certainly we have not seen CPU processor speed going up by a factor of ten in the last five years. Instead, we get now ten times as many processors (more precisely cores) for the price of one five years ago. PCs with 8 cores have become commodities and soon one may expect 64 or more. The usefulness of parallelization is no longer limited to large scale supercomputer applications, but becomes relevant for everyday calculations.

This motivates the present paper, which documents Fortran 77 MPI checkerboard [2] code for Markov Chain Monte Carlo (MCMC) simulations of pure SU(3) Lattice Gauge Theory (LGT) with the Wilson action on D-dimensional lattices. Sublattices are updated in parallel after collecting boundary variables from other sublattices. The introduced parallelization techniques apply to any model with dynamical variables defined on links.
and their interactions on plaquettes.

The code of this paper implements the Cabibbo-Marinari (CM) SU(3) updating [3] using for the SU(2) subgroups the heatbath method of Fabricius-Haan [4] and Kennedy-Pendleton [5] (FHKP), which is more efficient than the older Creutz heatbath [6]. CM with FHKP SU(2) updating is also about three times more efficient than Pietarinen’s [7] full SU(3) heatbath [8].

To synchronize the simulations on all processes, we use FHKP updating in the multi-hit accept/reject version [9]. Overrelaxation moves [10] are presently not implemented, but would fit seamlessly into the code. In extension of the usual periodic boundary conditions (PBC), which define the gauge system on a torus, our code allows for a double-layered torus (DLT). These are two identical lattices, each using the other as boundary, a geometry expected to be of relevance for studies of the deconfining phase transition.

The next section gives an overview of the code and explains Web access. Section 3 provides a number of verifications. Summary and conclusions follow in section 4. Runs are setup in the code, which reproduce the examples of this and a companion paper [11]. Running on up to 1 296 CPU cores, the companion paper studies performance as function of the number of MPI processes. It also discusses and resolves problems, which were encountered with MPI send and receive instructions for large arrays.

2. Overview of the Code

The code for this paper is freely available as a gzipped archive

STMC2LSU3MPI.tgz

that can be downloaded from the website of the author

http://www.hep.fsu.edu/~berg/research .

With

tar -zxfv STMC2LSU3MPI.tgz

the folder structure of Fig. 1 is created. Main programs are located in ForProg. The LIBS folder contains a number of libraries with plain Fortran 77 and Fortran 77 MPI code. Test and verification runs are setup in subfolders of several Project folders. Non-MPI SU(3) code and runs are in the project tree STMC_SU3.

We use checkerboard labeling [2] to divide lattice sites into two sets of colors $i_c = 1, 2$. Moving one step in any direction changes the color. For spin models with nearest neighbor interactions this allows one to update spins at half of the sites in parallel. For SU(3) LGT the matrices are located on lattice links and one can update at half of the sites one of the link directions in parallel. This is employed to update sublattices in parallel after collecting from other sublattices boundary variables, which need no updating because they belong to another checkerboard or link direction. For efficient performance the sublattice volume to surface ratio, each measured in numbers of variables, has to be sufficiently large. Examples are discussed in [11].

To arrange storage of our SU(3) matrices and other physical variables, we label lattice sites and links following the book of the author [12]. Corresponding routines from ForLib of this reference are taken over into Libs/Fortran of Fig. 1. In our approach a lattice site is specified by a single integer $i_s$, which we call site number. The dimension of the lattice is given by D. The Cartesian coordinates of a site are chosen to be

$$x^i = 0, \ldots, n^i - 1 \quad \text{for} \quad i = 1, \ldots, D. \quad (1)$$

The site number is defined by the formula

$$i_s = \sum_{i=1}^{D} n^{i-1} x^i. \quad (2)$$

Fig. 1. Structure of our program package.
\[ i_s = 1 + \sum_{i=1}^{D} x_i n_a^i, \quad n_a^i = \begin{cases} 1 \text{ for } i = 1, \\ \prod_{j=1}^{i-1} n_j^i \text{ for } i > 1 \end{cases} \] (2)

and calculated by the Fortran function \texttt{isfun.f}.

Vice versa, the coordinates for a given site number \( i_s \) are obtained by an iteration procedure, which relies on Fortran integer division (\( i.e.\), \( 1 = [5/3] = [5/5], \( 0 = [4/5], \) etc.). Let \( n_s = \prod_{i=1}^{D} n_i \) be the number of lattice sites. Then,

\[ x^D = \left[ \frac{(i_s - 1)}{(n_s / n_D)} \right] = \left[ \frac{(i_s - 1)}{n_a^D} \right] \] (3)

and for \( i = D - 1, \ldots, 1 \)

\[ x^i = \left[ \frac{(j^i_s - 1)}{n_a^i} \right], \quad j^i_s = i_s - \sum_{j=D}^{i+1} x^j n_a^j. \] (4)

The Fortran subroutine \texttt{ixcor.f} computes coordinates from the site number, though somewhat differently than by the formulas written down here.

The site number \( i_s \) allows one to store variables at sites in 1D arrays \( A_1(n_s) \), independently of the lattice dimension \( D \). Variables on links are located in 2D arrays \( A_2(n_s, nd) \), where the integer \( nd \) is the lattice dimension \( D \), \( nd \geq 2 \) for LGT. One more label is required to store SU(3) matrix elements in a 3D array. For checkerboard labeling we arrange the lattice variables in two arrays, corresponding to the colors \( i_c = 1, 2 \). The formula returning the color assignment of a lattice site is

\[ i_c = 1 + \text{mod} \left( \sum_{i=1}^{D} x^i, 2 \right). \] (5)

To update variables in array 1, neighbor variables are collected from array 2, which remains unchanged, and vice versa. LGT requires also to collect variables from the same checkerboard, which are not updated, because they are on links in other directions than the one updated. The checkerboard algorithm requires even lattice extensions. Otherwise PBC destroy the pattern.

2.1. Updating

Our code implements CM [3] SU(3) updating using for the SU(2) subgroups the FHKP [4,5] heat-bath algorithm. In the original FHKP version proposals are repeated until one is accepted, which is by construction from the desired probability distribution. For parallelization this is inconvenient, because all MPI processes have to wait until the last one finished. As pointed out by Fredenhagen and Marcu [9], one can terminate the inner loop after a finite number of hits and keep the link matrix at hand when none of the proposal has been accepted. The new configuration is still proposed with the local heatbath distribution. What changes is the average stay time of the old configuration. This time depends on the configuration at hand, but drops out in the detailed balance equation. By using CM heatbath in this Metropolis-like fashion the MPI processes get synchronized. The 1-hit acceptance rate depends on \( \beta \) and is in the scaling region of SU(3) LGT around 97%. Lower 1-hit acceptance rates are encountered for smaller \( \beta \) values. One may then increase the number of hits.

As usual, the updating step keeps track of the total action. Due to parallelization action differences have to be added by the MPI process of the sublattice on which the update is carried out. Then, action fluctuation across boundaries can be created, which lead in course of time to absurd sublattice contributions, while the total action (their sum) is still correct. To elaborate on this point, we first need to define sublattice actions. Due to links crossing boundaries, there is some amount of freedom in that. We simply attribute the action of a plaquette to the sublattice, which contains the site from which two forward links of the plaquette emerge. Now, updating one of the other links of a plaquette, the action change is recorded in a wrong sublattice, if the updated link emerges there.

As long as one is only interested in the total action, it is sufficient to recalculate the sublattice action once in a while directly to prevent an amplification of rounding errors due to differences of large numbers (sublattice contributions can fluctuate to negative values). However, if one wants to attribute physical significance to sublattice actions, it is mandatory to recalculate them before a measurement is recorded.

Our updating subroutine is \texttt{cbsu3_2hbnhit.f} located in \texttt{Libs/MPISU3}. A call to this routine performs one sweep, which is here defined by update-
ing each SU(3) matrix once in sequential order (see section 2.4 for more details). Updating in sequential order fulfills balance and is more efficient than updating link matrices in random order. This observation holds also for spin models [12].

2.2. Double-Layered Torus

This section can be skipped by readers, who are only interested in simulations with PBC. The DL T is, for instance, of interest for simulations on $N^3$ lattices if one likes to have boundaries at a different temperature than the interior of the lattice [13], as it is the case for deconfined volumes created in relativistic heavy ion collisions. The DL T is defined by two lattices of identical size, each using the other as boundary in all or just volume directions. In the latter case distinct $\beta$ values in the lattices lead to different physical temperatures $T$ through the usual definition of $T = 1/(\alpha N_\tau)$.

Even with identical $\beta$ values in both lattices the DLT has some intriguing properties as illustrated in Fig. 2 for a 2D DLT of size $(N_s)^2$. The boundaries are glued together as indicated by the arrows. While for PBC the shortest connection of a point with itself through the boundary is of length $N_s$, it is now of length $\sqrt{2}N_s$ along the diagonal. The two arrows in diagonal direction give an example of a line, which is closed by DLT boundary conditions.

1 Note that interchanging the labels 3 and 4 on one of the lattices of the figure leads to an undesirable situation in which some sites pairs are connected by two links

Compared to a torus of size $(N_s)^D$, the effective extension of a DLT with DLT boundary conditions in all directions is

$$N_s^{\text{eff}} = 2^{1/D} N_s,$$

so that $(N_s^{\text{eff}})^D$ is the size of the DLT. One may argue that finite length corrections are exponentially suppressed by $\sqrt{2}N_s$, which is for $D > 2$ larger than $N_s^{\text{eff}}$. Then one would for $D \geq 3$ gain with respect to the suppression of finite size effect compared to the usual torus. However, simulations of the 3D and 4D Ising model on a DLT [14] showed an exponential suppression of finite size corrections with $N_s^{\text{eff}}$ and not with $\sqrt{2}N_s$. The reason for that has remained unclear.

When using two different $\beta$ values, $\beta_0 \neq \beta_1$, we assign a unique $\beta_i$, $i = 0, 1$ to each plaquette in a slightly asymmetrical way: If any link of a plaquette is from the second torus, we take $\beta_1$, otherwise $\beta_0$. Technically this is done by tagging all links in the first torus by 0 and in the second by 1. When considering a plaquette, all these tags are added up. If the sum is zero, $\beta_0$ is used, otherwise $\beta_1$. So the $\beta_1$ lattice becomes slightly larger than the $\beta_0$ lattice.

2.3. Parameter files

As indicated in Fig. 1 runs are kept in subfolders of project folders, one run per subfolder. The relevant parameters are set in two files: \texttt{latmpi.par} and \texttt{mc.par}. Before the compile step the parameters are transferred by a simple preprocessing procedure into subroutines and, in particular, used to dimension common blocks properly (see section 2.4). Due to this procedure it is mandatory that runs and their parameter files are kept two levels down from the \texttt{STMC2LSU3MPI} root directory.

As an example the parameter files of the run in

\texttt{1MPICH/08x08y08z04t5p65b2f3d}

are given below.

\begin{verbatim}
latmpi.par
# Bernd Berg Jan 11, 2009, MPI Checkerboard lattice:
# c fln Part of data file name.
# c lbvex Boundary exchange T/F.
# c ldmpi .true. distinct random number seeds on each process,
c .false. identical random number seeds (for tests).
# c nd Dimension of the lattice space
\end{verbatim}
Central are the sublattice extensions, \( n_{l1}, n_{l2}, n_{l3}, n_{l4} \) in 4D, and the MPI parameters \( ndmpi, mpifactor \). For \( mpifactor = 1 \) there is only one process and the entire lattice agrees with the sublattice. For \( mpifactor > 1 \) the extensions of the entire lattice agree with those of the sublattice in directions larger than \( ndmpi \), which exist for \( ndmpi < nd \), and there are \( mpifactor \) sublattices in each of the \( ndmpi \) directions. The sublattices themselves form a lattice of dimension \( ndmpi \) with

\[
msmpi = mpifactor * ndmpi
\]

points, which we refer to as \( MPI \) lattice. To allow for variable extensions, the sublattice values are stored in an array \( nla \). To make use of the same routines, the MPI lattice extension \( mpifactor \) is similarly stored in an array \( nla_{mpi} \). Both arrays are initialized in the file \( latmpi.dat \):

```
data nla/nl1,nl2,nl3,nl4/,nla_mpi/ndmpi*mpifactor/
```

Usual PBC are simulated for \( nl = 1 \), the DLT for \( nl = 2 \). For PBC the number of MPI processes in \( msmpi \), while for the DLT it is \( 2*msmpi \). We will get familiar with choices of other \( latmpi.par \) parameters when we perform verification and test runs in the next section.

The example file for \( mc.par \) is:

```
c Bernd Berg, Jan 11, 2009.
c Definition of parameters for SU(3) LDT simulations.
c Output units io, iud, iud2 (off/on with iud2), iud3.
c Job number nj and seeds iseed1, iseed2.
parameter(iu=6, iud=11, iud2=12, iud2=0, ltrue=false, iu3d=13)
parameter(njob=1, iseed=0, job=0)
c Parameters for equilibrium and data production sweeps:
c beta0,1: beta_g=2/g^2 inverse bare coupling.
c istart: 1 ordered start all matrices 1; 2 disordered start.
c nhit: Number FHP proposals made.
c nrep: Number of repetitions of nsweep sweeps.
c nequi: Number of sweeps between measurements.
c nreq: Number of sweeps between measurements.
c nmeas: Number of measurement sweeps per repetition.
```

The purpose of most parameters should be obvious from the comments. The MCCM run structure is that defined by \( nequi, nreq \) and \( nmeas \) in Ref. [12]. After equilibrium measurements are saved in \( nreq \) blocks to allow for a conveniently binned analysis, employing jackknife methods when suitable. There are \( nmeas \) sweeps done between measurements.

### 2.4. Program structure

We trace the code structure and that of a typical run from the main program

```
cbsu3_dlt{a,b,c}.f . (8)
```

The program comes in three versions \{a, b, c\}, where \( b \) is obtained from \( a \) by simply replacing everywhere in the code \( mpi = mpib \), and similarly for \( c \). As discussed in [11], differences lie in the coding of MPI send and receive instructions. We were unable to find a single solution which works on all MPI platforms on which we performed tests. The \( a \) version, which uses the simplest (plain) subroutines for boundary transfers, is listed in the following.

```
program cbsu3_dlt ! Berg Jan 11 2009.
C MPI checkerboard for SU(3) lattice. Perodic
C boundary conditions and double-layered torus with two beta values.
C Version a: Plain send/receive (mpi extensions).
C Version b: Buffered send/receive (mpi extensions).
```
C Measurements of action, spacelike, timelike plaquettes.
C Recalculation of action before each measurement is only
C needed for nlat=2, lat2=.true..

C LGT Checkerboard:
C For test purposes only:
C include '../../Libs/MPI_par/cbsu3_act_mpi.f' ! Calculat e action.
C include '../../Libs/MPI_par/cbsu3_actdif_mpi.f'! Check action.
C include '../../Libs/MPI_par/cbsu3_addb_m_m.f'! Addition.
C include '../../Libs/MPI_par/cbsu3_add_m_m.f'! Addition.
C include '../../Libs/MPI_par/cbsu3_bnd_mpi.f' ! Gather bo undary.
C include '../../Libs/MPI_par/cbsu3_act_MPI.f' ! Calculate action.
C include '../../Libs/MPI_par/cbsu3_bnd_MPI.f' ! Gather boundary.
In the first lines of the program, after the comments, the general structure is defined. Variables are declared throughout the entire code by including the `implicit.08` file of the Fortran library folder:

```
implicit real*8 (a-h,o-z)
implicit logical (1)
```

This has the advantage that the type of a variable follows from its first letter. An exception to this rule are character variables, which are explicitly declared, though their first letter is always `c`. No complex variables are used. MPI is set up by including the system provided file `mpif.h` and a number of constants are defined by including the file `constants.08` (see inside the file).

The program is compiled by a file `mpimake`, or similar, of which a copy is located in each project folder and listed here as used for Open MPI[2].

```
mpif77 -O -Wall $1
cp *.par ../../Libs/Fortran_par/.  
rm ../../../Libs/Fortran_par/*.par
```

The `mpimake` command transfers the parameter files into the `Fortran_par` folder and removes them from there after the compile step. This creates a hyperstructure, which transfers to all subroutines identical parameter values and dimensions common blocks properly. As already mentioned, to keep this structure intact runs `must` be carried out in subfolders, which are two levels down from `STMC2LSU3MPI`. Job submission is subsequently done by `run*` executables, which are kept in the run subfolders.

All library routines needed by the program are explicitly included at the end of the main program. So their source code can be easily located. Exceptions are calls to MPI routines (all routines with names starting with `mpi`), which have to be looked up in MPI manuals or tutorials (for instance [12], see [11] for subtle points with send and receive).

Step by step the execution of a run is explained in the following.

(i) MPI initialization by `mpi_init`.
(ii) Calculation of the rank (identity `my_id` of the MPI process) by `mpi_comm_rank`.
(iii) Some printout from MPI process zero, setup of printout for each process if `1ud2` is true.
(iv) A call to `cbsu3_2init_mpi` initializes the run, setting up many important features:

(a) A call to `rmaset` initializes Marsaglia’s (pseudo) random number generators [15,12] used throughout this code. For `1sd2mpi` true the process rank is invoked in the seed, so that a different generator is used for each MPI process.
(b) Definition of pointer arrays for checkerboard labeling and exchange of boundary values by calls to `lat_init`, for `ni=2` also to `lat2a_init` and `lat2b_init`, then to `cblat_init`, for `lbcex` true (means MPI boundary conditions exchange) to `cblgtpointer` and, finally, for `ndmpi` ≥2 to `cblgtpt2`.
(c) For the DLT a call to `cbsu3_iba_mpi` assigns a unique β to each plaquette. This routine has to be called before the start configuration is initialized, because it uses the SU(3) matrix array for temporary storage. Tags are finally stored in the arrays `iba1` and `iba2` of the common block `common_cbsu3.f` and pointers to the β values in the array `ba(0:4)`.
(d) A call to `csu3_start` generates a SU(3) start configuration.
(e) A call to `cbsu3_action_mpi` calculates the initial action.
(v) Calls to `cbsu3_action_mpi` check whether the action kept on record during the updating process agrees with the one obtained by direct calculation. Process 0 writes action information to the formatted output file.
Calls to `mpi_barrier` are supposed to synchronize the MPI processes, but may indeed have no effect.

Calls to `write_progress` by MPI process 0 write information to a file `progress.d`, which is opened and closed, so that the user can look up the file during run time.

For equilibration a double loop (`nreq` and `nequi`) of calls to the updating routine `cbsu3_2hbnhit_mpi` is performed. The purpose of a double loop is that the run can be interrupted when the total equilibration time exceeds the CPU time allowed for a single run. The updating routine relies on a number of subroutines:

(a) `cbsu3_bstaple1` calculates the staple for updating a link matrix on checkerboard 1 (`cbsu3_bstaple2` correspondingly on checkerboard 2). These routines use various matrix manipulation routines from `Libs/SU3`.

(b) `su3mult` multiplies SU(3) matrices of the first two arguments and returns the result in the third argument.

(c) `su3reunit` reunitarizes a SU(3) matrix.

(d) `cbsu3_bnd1a_mpi` collects boundaries (no corners) from a sublattice checkerboard 1 and sends them to other sublattices (`cbsu3_bnd2a_mpi` for collection from checkerboard 2). A subtle point is in gauge systems that one needs for \( n_{mpi} > 1 \) corner links from two neighboring sublattices like the links emerging from sites 2 and 3 in Fig. 3. This is handled by one more routine:

(e) `cbsu3_bnd1b_mpi` collects for \( n_{mpi} > 1 \) boundary corners from checkerboard 1 and sends them to other sublattices (`cbsu3_bnd2b_mpi` for collection from checkerboard 2).

Updating sweeps with measurements are carried out in a triple loop (`nrpt`, `nmeas` and `nsw`). Measurements are done every `nsw` sweeps and kept in time series arrays of length `nmeas`. To write reasonably sized unformatted arrays to disk is considerably faster than writing after each measurement step. Increasing `nsw` prevents strongly correlated measurements. A good choice for `nsw` is between 1% and 10% of the expected integrated autocorrelation time \( \tau_{\text{int}} \) [12], which depends not only on \( \beta \) and the lattice size, but also on the observable. Using a too large value for `nsw` destroys the possibility to estimate \( \tau_{\text{int}} \) from the run data.

Measurements are temporarily stored in arrays of the common block `common_cbsu3`. For spacelike and timelike plaquettes they are done by `cbsu3_wloops_mpi` and kept in the times series (ts) arrays `tsws` and `tswt`.

### 3. Verifications

Although our code is written for a variable lattice dimension \( D \), tests have so far been limited to 4D. The programs and routines are only moderately cleaned up. Many parts have disabled (\( ltest \Rightarrow .false. \)) or commented out test options. They are presently left in the code, because they could come into use again.

This section deals with verifications, which were performed on a 2 GHz AMD Athlon 64 XMX Dual Core Processor 3600+ at Leipzig University. MPI runs with `mpifactor` > 1 use both processors, single processor runs one of them. Fortran 77 compilation was done with the g77 compiler based on
gcc version 4.3.2 (Debian 4.3.2-1.1). MPI runs were performed with MPICH version 1.27p1. Compiler warnings about slow initialization of large aggregate areas have been ignored as the produced code works just fine. More MPI runs using up to 16 cores on a PC cluster and up to 1296 on a Cray are documented in [11].

A strong test for the correct implementation of exchange of boundaries is provided by using identical random numbers on each sublattice. Then results from all sublattices have to agree and be identical with a run on a single lattice of this size with PBC. After such tests, real production runs were performed to compare action expectation values with results from the literature [16] and from our conventional (non-MPI) SU(3) Fortran program: The average over the $Q$ values of Gaussian difference tests [12] between these six runs was close to 0.5 as it should.

We document here only the

$$\beta = 5.6 \text{ runs with ordered starts}.$$  

The analysis is kept on `ana2.txt` files. We obtained for the mean action per plaquette with MPI code (error bars are given in parenthesis and always rounded upwards in their second digit)

$$\text{act} = 0.53811 \pm 0.00019,$$  

versus with single processor code

$$\text{act} = 0.53770 \pm 0.00018,$$  

leading to an acceptable $Q = 0.12$ in the Gaussian difference test. The integrated autocorrelation time of these runs is estimated to be $\tau_{\text{int}} = 49.5 \pm 3.4$. So an error bar calculation with respect to 32 bins is appropriate.

MPI runs were repeated for the pairs (1,1), (2,1), (2,2), (2,3), (2,4), (3,1), (3,2) of the parameters

$$(\text{mpifactor, ndmpi})$$  

(13)
giving (due to `lsd2mpi` false) always to the same average action (11). The corresponding numbers of MPI lattice points (MPI processes) $\text{msmpi}$ (7) are 1, 2, 4, 8, 16, 3 and 9.

Parameters of the runs of this section are kept in

{\{F\}nnxnnynznntnbnfnd}  

(14)
subfolders of `1MPICH`, where F indicates `lsd2mpi` = .false. and is omitted for `lsd2mpi` = .true.. The letters n indicate numbers, which can be different. Lattice extensions are given by nxx, nxy, nzz and nnt. This is followed by npb for $\beta_0 = \beta_1 = n.n$, by n from nf for mpifactor = n, and n from nd for ndmpi = n. In the folder names extensions of
the full lattice are used, whereas data set names created by the program (8) are of the form

\[ \text{SU3LGTnndnnnxnnntnnnn}\ D, \] (15)

showing sublattice extensions of the x and t directions. The program calculates also the extensions of the full lattice from \text{latmpi.par} and prints them in the readable output file. Another way to find sublattice extensions is from the folder name by dividing the full lattice extensions by \text{mpifactor} for the ndmpi directions. The results have to be integers without rest term. As folder names are created by hand, the output file from the run is authoritative in case of a discrepancy.

The other acronyms in the data set name (15) are the lattice dimension \( D = n \) of the first nd, then \text{mpifactor} = n from \text{nf}, and the MPI lattice dimension \( ndmpi = n \) from \text{nd}. The extension \( nnnn.D \) labels data files by their process number. Each of the created data files corresponds to one of the sublattices. After data production, the 1-processor MPI program \text{su3datcollect.f} (16)

condenses these data files into a single one for which the extensions tnnn.D are reduced to t.D. When disk space fills up it is sufficient to keep only the *t.D files.

To give an example, the subfolder name

\[ \text{F08x08y08z04t5p6b2f2d} \]

corresponds to \text{lsd2mpi = .false.}, runs on \( 4^3 \) sublattices at \( \beta = 5.6 \) with a full lattice size \( 8^3 4^2 \). The MPI run produces four sublattice data sets

\[ \text{SU3LGT4d2f2d004x004tnnnn.D} \]

with nnnn from 0000 to 0003. After data collection with (16) the file

\[ \text{SU3LGT4d2f2d004x004t.D} \]

results, which can be analyzed further.

3.3. Different random numbers on sublattices

We set

\[ \text{lsd2mpi = .true. and mpifactor = 2} \]

in \text{latmpi.par} and produce data at \( \beta = 5.65 \) from simulations of \( 8^3 4 \) lattices, which are partitioned into different numbers of sublattices. Average action densities are compiled in table 1. The statistics of each run is the same as before (9). Each \( Q \) value in the table corresponds to the Gaussian difference test with the action value in the row above. The number of MPI processes agrees with the number of sublattices given by (7). The \text{time} column contains the CPU time measured on the Athlon PC. The non-MPI run for the first row uses the FHKP heatbath in the repeat until accepted version. Whether it is slower of faster than the MPI program run for one process (\( nf = 1 \)) depends on the Fortran compiler and the MPI installation. Here it turns out to be faster, but that is not the case on the PCs used in [11]. For \( nf = 1 \) one can turn off the boundary exchange in the MPI program by setting \text{lbcex = .false.}, indicated by 1F in the \text{nf} column. Results stay number by number identical and the small speedup is negligible for practical purposes, indicating that MPI send and receive is very efficient as long as communication stays within the same computer node. The decrease of CPU time from \text{msmpi = 1} to \text{mspi = 2} reflects the gain in real time due to using both cores of the PC. It is given by a factor slightly larger than 1/2 due to communication overhead. The parameters of these runs are setup in the

\[ \text{08x08y08z04t5p65bnfnd} \]

subfolders of \text{1MPICH}, where the notation is as introduced in (14).

Results from \( 16^3 4 \) lattices, which allow for comparison with spacelike and timelike plaquette aver-
Table 2
Spacelike and timelike plaquette expectation values for comparison with Ref. [16] (8 in the np column): Runs on a 16\^3 lattice at β = 5.65 using our non-MPI program and MPI code with np processes (sublattices).

| np | spacelike   | Q         | timelike   | Q         |
|----|-------------|-----------|------------|-----------|
| 2  | 0.537638 (17) | -         | 0.537692 (19) | -         |
| 1  | 0.537647 (17) | 0.890000 | 0.537701 (17) | 0.650000 |
| 2  | 0.537650 (16) | 0.900000 | 0.537704 (17) | 0.900000 |
| 4  | 0.537648 (16) | 0.930000 | 0.537708 (16) | 0.860000 |
| 8  | 0.537661 (18) | 0.590000 | 0.537714 (17) | 0.880000 |

*ages of the literature [16], are compiled in table 2. For our runs the statistics (9) was used again. The estimates of [16] rely on 20,000 to 40,000 sweeps after thermalization. After collection of our data by running (16), the analysis program anaw_cbsu3.f estimates the expectation values for spacelike and timelike plaquettes. Again, Q values correspond to Gaussian difference tests with the row above. The slight increase of all values with increasing numbers of processes is accidental and not reproduced when using different random number generator seeds.

3.4. **Double-layered torus**

This section deals with verifications for simulations with

```
  nl14 = 0
```

and folders of the runs are kept in

2MPICH.

For `lat2 = .false.`, the exchange of boundaries is turned off and one performs independent runs on two lattices with PBC. This is not very interesting and we discuss only runs with `lat2 = .true.`.

With exchange of boundaries turned on, a strong verification test is to run with different random numbers on the sublattices of a torus, but identical β values and random numbers on each torus. These are the parameter options

```
  lsd2mpi = .true. and lat2test = .true.,
```
designed to reproduce the action values of the run with PBC for which statistics and sublattices match. For tori of size 8\^3 and β = 5.65 these are values of table 1. These test runs worked out as required and are setup in the

```
T08x08y08z04t5p65bnfnd
```
folders of 2MPICH, where the initial T indicates that `lat2test` is set to true.

Table 3
Runs with np MPI processes, `mpifactor=nf, ndmpi=n` on a double-layered 8\^3 lattice at β = 5.55, disordered starts.

| np | nf1 | n11 | n12 | n13 | n14 | actm | Q |
|----|-----|-----|-----|-----|-----|-----|---|
| 2  | 2   | 8   | 8   | 4   | 4   | 0.510608 (33) | - |
| 4  | 2   | 4   | 8   | 8   | 4   | 0.510671 (28) | 0.15 |
| 8  | 2   | 4   | 4   | 8   | 4   | 0.510597 (24) | 0.04 |
| 16 | 2   | 4   | 4   | 4   | 4   | 0.510676 (32) | 0.05 |

Proper simulations on a DLT are performed with `lat2test = .false.`. Table 3 gives reference values for the action at β = 5.55 (at β = 5.65 one needs more statistics due to autocorrelations that become important for small error bars). The Q values refer to Gaussian difference tests as in previous cases. They are a bit on the low side, but it is clear that simply reordering the comparison of data would change that. Also runs with a different compiler gave (in the same order as listed in the table) Q = 0.97, 0.09, and 0.30.

4. Summary and Conclusions

The code of this paper allows Markov chain Monte Carlo calculations of pure SU(3) lattice gauge theory on computers which have MPI installed. Besides the usual periodic boundary conditions, the geometry of a double-layered torus is implemented, which allows for distinct inside and outside environments (the "inside" of one lattice is the other's "outside" and vice versa). A considerable number of non-trivial verification are included in this paper. The CPU time performance of the code as function of the number of processors (more precisely CPU cores) is documented in a companion paper [11].
Should the a-variant of the program “hang-up” without producing an error message, the cause are likely MPI send and receive problems which are discussed and partially resolved in [11]. Although designed for arbitrary $D \geq 2$ dimensions, the code has presently only been tested in 4D. Hence, it is unlikely to work straight away for other $D$ values (nd in lat.par), though required fix-ups are expected to be minor. Of course, it is in the responsibility of the final user to perform stringent tests before applying the provided code to any purpose.

Acknowledgments: The non-MPI part of this code is based on joint work with Alexei Bazavov [13]. Contributions are indicated in the code. I would like to thank Alexei Bazavov, Urs Heller and Wolfhard Janke for useful discussions. This work was in part supported by the U.S. Department of Energy under contract DE-FG02-97ER41022 and by the German Humboldt Foundation. I am indebted to Wolfhard Janke, Elmar Bittner and other members of the Institute for Theoretical Physics of Leipzig University for their kind hospitality.

References
[1] G.E. Moore, Electronics 38 (1965) 114.
[2] D. Barkai and K.J.M. Moriarty, Comp. Phys. Commun. 27 (1982) 105.
[3] N. Cabibbo and E. Marinari, Phys. Lett. B 119 (1982) 387.
[4] K. Fabricius and O. Haan, Phys. Lett. B 143 (1984) 459.
[5] A.D. Kennedy and B.J. Pendleton, Phys. Lett. B 156 (1985) 393.
[6] M. Creutz, Phys. Rev. D 21 (1980) 2308.
[7] E. Pietarinen, Nucl. Phys. B 190 (1981) 349.
[8] Estimate by A. Bazavov (unpublished).
[9] K. Fredenhagen and M. Marcu, Phys. Lett. B (1987) 486.
[10] S.L. Adler, Phys. Rev. D 37 (1988) 458.
[11] B.A. Berg and H. Wu, arXiv, to appear.
[12] B.A. Berg, Markov Chain Monte Carlo Simulations and Their Statistical Analysis, World Scientific, 2004.