BQCD Manual

Yoshifumi Nakamura and Hinnerk Stüben

October 2011

Yoshifumi Nakamura
RIKEN Advanced Institute for Computational Science
Kobe, Hyogo 650-0047
Japan
<nakamura@riken.jp>

Hinnerk Stüben
Universität Hamburg
Regionales Rechenzentrum
20146 Hamburg
Germany
<hinnerk.stueben@uni-hamburg.de>

Contents

Preface

Berlin quantum chromodynamics program (BQCD) is a Hybrid Monte-Carlo program for simulating lattice QCD with dynamical Wilson fermions. The development of BQCD started in 1998 by H.S. for the two flavour case and the original Wilson action. It was written for a study of parallel tempering [2, 3] (these roots are still visible at some places). At that time the whole parallelisation framework was completed.

Two years later the program was extended in two different directions. The first direction was the implementation of clover $O(a)$ improvement of the fermionic action. With the availability of clover improvement BQCD became one of the main production codes of the QCDSF collaboration [4]. The second direction was the addition of an external field to the standard Wilson action in order to study the Aoki phase [5, 6]. The next milestone was the implementation of the Hasenbusch trick [7, 8].

Since 2006 the main code development has been made by Y.N. The code was largely extended and improved to enable simulations including a third fermion flavour [9, 10, 11, 12].

The code is also being used by the DIK Collaboration for simulations at finite temperature [13, 14]. Several people took BQCD as a starting point for adding their own code for measurements. The plan of the QPACE project [15] to port BQCD to their machine has triggered the publication of the code as free software under the *GNU General Public License*. We hope that it will be useful for others and kindly ask to cite our contribution to the proceedings of *Lattice 2010* [1] in case the code is used to prepare a publication.

June 2010 *Yoshifumi Nakamura*
Hinnerk Stüben

1 Overview

BQCD is a program to generate configurations with clover type action.

Implemented functions for dynamical simulations:

- N_f=2 clover action
- N_f=2+1 clover action
- Improved gauge action
- Fat link clover action with stout link smearing
- with imaginary θ

Implemented functions for tuning:

- RHMC algorithm
- Multi mass Preconditioning
- Multi pseudoscalar Fermions
- Multi time Scale Integration
- Minimal norm integrator (Omelyan)
- Minimum and maximum eigenvalues calculation of $\tilde{M}^\dagger \tilde{M}$
- User Assistance for determination u_0 on tadpole improvement

Many of these items will only be sketched in this manual. As a starting point we suggest to look at the test cases we have described in section 2.5 for $N_f = 2$ and $N_f = 2 + 1$ simulations.

1.1 Summary of changes

- version : 4.0.0 (June 2010)
  - first public version
- version : 4.1.0 (October 2011)
  - GCRO-DR solver
  - replay trick
  - Schrödinger functional method to determine c_{SW}
  - further RHMC tuning (see section 5.2)
  - SSE implementation of the hopping and clover matrix multiplications (set `libd = 103` in `Makefile.var`)
  - Check return value of functions for reading/writing ILDG format
  - Minor changes for printing and function interface

2 Installation

2.1 Prerequisites

The preferred directories for installing prerequisite packages are

```
$HOME/opt/package .
```

Prerequiste directories can be changed in the `Makefile.in`. It is recommended to use the same compiler for building packages and BQCD.

2.1.1 lime

It can be downloaded from:

```
http://usqcd.jlab.org/usqcd-software/c-lime/lime-1.3.2.tar.gz
```

Installation:

```
cd ~/opt
tar zxvf lime-1.3.2.tar.gz
cd lime-1.3.2
export CC=non-default compiler            # optional
export CFLAGS=non-default compiler flags  # optional
./configure --prefix=$PWD
make
```

2.1.2 LAPACK

LAPACK installed personally must be put as

```
LAPACK = $(HOME)/lib/$(COMPILER)/lapack.a $(HOME)/lib/$(COMPILER)/blas.a
(see Makefile.in)
```

2.2 Download

The sources of BQCD and this manual can be downloaded from

```
https://www.rrz.uni-hamburg.de/fileadmin/forschung/bqcd .
```

2.3 License

2.4 Configuration

2.4.1 Supported platforms

Platform dependent parts are kept in two files which are symbolic links to corresponding files in the `platform` directory

```
Makefile.var -> platform/Makefile-platform.var
service.F90 -> platform/service-platform.F90
```

for example:

```
Makefile.var -> platform/Makefile-gnu.var
service.F90 -> platform/service-gnu.F90
```

One can prepare working on a particular platform by entering the command

```
make prep-platform
```

which creates symbolics links, for example:

```
make prep-gnu
```

In the `platform` directory one can find files for machines that were used in the past but that are not necessarily up-to-date. Currently you can expect that compilation works in these cases:

`altix2`	SGI Altix 4700 at LRZ Munich (Intel compiler, MPI from SGI)
`gnu`	GNU compiler, single processor version
`hlrn2`	SGI ICE at HLRN (Intel compiler, various MPI libraries)
`jugene`	IBM BlueGene/P at JSC Jülich
`linux`	GNU compiler, MPICH/openMPI

2.4.2 Settings in `Makefile.var`

In `Makefile.var` one can make the following high level settings:

`timing`	= *empty* or `1`	switch on profiling
`mpi`	= *empty* or `1`	single processor program or MPI
`omp`	= *empty* or `1`	compile with OpenMP
`debug`	= *empty* or `1`	compile with debug flags
`libd`	= `100`	which hopping matrix multiplication
`random`	= `ranlux-3.2`	which random number generator

On platform `hlrn2` `mpi` must not be set to `1` but rather to a word describing an MPI library: `mpt` (SGI Massage Passing Toolkit), `mpich` (MVAPICH2) or `impi` (Intel MPI).

When switching on OpenMP for the GNU compiler one should use *gfortran* version 4.4.1 or higher.

These settings are relevant only on systems that support the *shmem* communication library:

`shmem`	= *empty* or `1`	use *shmem* instead of MPI
`shmempi`	= *empty* or `1`	use *shmem* only in hopping matrix multiplication

Based on theses high level settings low level settings are made. This includes compiler flags, preprocessor flags and selection of libraries.

The `make` procedure is sometimes not straightforward. Instead of `make` `make fast` builds the binary. The binary is called `bqcd4`.

2.5 Testing

For testing reference data can be found in directory `data`. There are test cases for $N_f = 2$ and $N_f = 2 + 1$. Due to rounding differences the output will not be identical to the reference output but should be very close to the reference output.

2.5.1 Testing $N_f = 2$

```
cd data
../bqcd4 bqcd.200.input bqcd.200.res
diff bqcd.200.output bqcd.200.res
```

2.5.2 Testing $N_f = 2 + 1$

```
cd data
../bqcd4 bqcd.300.input bqcd.300.res
../bqcd4 -c bqcd.300.input bqcd.300.res
diff bqcd.300.output bqcd.300.res
```

2.5.3 Testing parallel runs

- Set the number of processes to be used for each lattice dimension in the input file, for example

  ```
  processes  1 1 2 4
  ```

  to decompose the z- into 2 domains and the t-direction into 4 domains.

- Run BQCD on the appropriate number on processes, which is 8 in the above example:

  ```
  mpirun -np 8 ../bqcd4 bqcd.200.output bqcd.200.res
  ```

  In principle the output (`bqcd.200.res`) is identical to the output from the sequential run (again up to rounding differences). This holds true for any decomposition.

3 Usage

3.1 Command line

BQCD takes the following arguments:

```
bqcd [-c] [-I] [-V] input [output]
```

3.1.1 `-c` – continuation job

If the `-c` parameter is present the start configuration is being read from file. Otherwise a start configuration is being generated. The setup is such that `input` does not have to be modified from the first to the second job in a job chain.

3.1.2 `-I` – print default input

If `-I` is given the program prints all possible input parameters with their default values and exits. Most of them are decribed in section 3.2.

3.1.3 `-V` – print program version

The program prints version information and exits. The output looks like this:

```
 This is bqcd 3.9.0 (revision 244)
    input format:              4
    conf info format:          3
    MAX_TEMPER:               50
    REAL kind:                 8
    Version of D:            100
    Communication:   single_pe
    RandomNumbers:   ranlux-3.2 level 2
```

3.1.4 `input`

Name of input parameter file.

3.1.5 `output`

Name of log- and results file. If not given data will be written to *stdout*. If the file does not exist it will be created. If `-c` is not set an existing file will be overwritten. If it is set new output will be appended to the file.

3.2 Input parameters and syntax of `input` file

The syntax of the input file is one *keyword value(s)* pair (or tuple) per line. Empty lines and lines beginning with a `#` character are ignored. Keywords are checked for validity but the number of values and the types of values are not. It is a good idea to enclose character string parameters in double quotes `"..."` (in particular Fortran will scramble filenames containing slashes).

3.2.1 General parameters

`run`	*integer* `from 0 to 999`
`comment`	*string*

3.2.2 Lattice and its decomposition

`lattice`	`Lx Ly Lz Lt`	lattice size
`processes`	`Px Py Pz Pt`	process grid (`L`$_{\mathtt{i}}$ must be divisible by `P`$_{\mathtt{i}}$)
`boundary_conditions_fermions`	±1 ±1 ±1 ±1	
`boundary_sf`	`0`	if !=0, Schödinger functional boundary for c_{SW} (`Pt` must be `> 1`)

3.2.3 Physical parameters

`gauge_action`	`WILSON`	*or* `TREE` *or* `IWASAKI` *or* `(TREETAD)`
`fermi_action`	`NON`	*or* `WILSON` *or* `CLOVER` *or* `SLW` *or* `SLIC` *or* `SLRC`
`beta`	`0.0`	gauge coupling
`kappa`	`0.0`	hopping parameter for 2 degenerate fermions
`kappa_strange`	`0.0`	hopping parameter for $+1$ fermion
`csw`	`0.0`	coefficient of clover term
`n_stout`	`0`	number of stout link smearing steps
`alpha`	`0.0`	parameter for stout link smearing
`h`	`0.0`	coefficient to break parity-flavour symmetry
`theta`	`0.0`	coefficient to break CP symmetry

3.2.4 Start parameters

These parameters take only effect if `-c` (continuation) is not given on the command line. The start configuration can be cold, hot or be read from file. The start configuration file can be in `bqcd/info` format or in `ildg/lime` format. The random number generator can be initialised with a default seed or a user defined seed.

`start_configuration`	`cold` *or* `hot` *or* `file`	
`start_info_file`	*`filename`*`.info`	(if set, do not specify `start_ildg_file`)
`start_ildg_file`	*`filename`*`.lime`	(if set, do not specify `start_info_file`)
`start_random`	`default` *or* *`integer`*	

3.2.5 Monte-Carlo parameters

`mc_total_steps`	*`integer`*	total number of trajectories in this run
`mc_steps`	*`integer`*	number of trajectories per job
`mc_save_frequency`	*`integer`*	configuration save frequency

A *run* consists of several *jobs*. If the trajectory counter is at `mc_total_steps` the program generates a stop file and exits. The stop file is named

```
progname.run.STOP
```

for example: `bqcd.042.STOP`. Configurations are saved if

```
mod(trajectory_counter, mc_save_frequency) == 0 .
```

3.2.6 Hybrid Monte-Carlo parameters

`hmc_test`	`0`	if !=0 reversibility is checked
`hmc_model`	`E` *or* `F`	for 2 or 2+1 flavours
`hmc_accept_first`	`0`	this number of trajectories are accepted forcedly
`hmc_trajectory_length`	`1`	is twice as long as *Chroma*'s `tau`
`hmc_integrator1`	`LPFSTS, LPFTST, 2MNSTS, 2MNTST, 4MN4FP` *or* `4MN5FV`	
`hmc_integrator2`	`NON` *or others*	
`hmc_integrator3`	`NON` *or others*	
`hmc_integrator4`	`NON` *or others*	

`hmc_integrator5`	`NON`	*or others*
`hmc_integrator6`	`NON`	*or others*
`hmc_steps`	0	number of steps for first level of integrator
`hmc_m_scale`	1	inverse time scale ratio to 1 level higher
`hmc_m_scale2`	1	inverse time scale ratio to 1 level higher
`hmc_m_scale3`	1	inverse time scale ratio to 1 level higher
`hmc_m_scale4`	1	inverse time scale ratio to 1 level higher
`hmc_m_scale5`	1	inverse time scale ratio to 1 level higher
`hmc_hkappa`	0	way of mass preconditioning 0: M1=M+rho otherwise: kappa is replaced by rho
`hmc_rho`	`0.0`	parameter for mass preconditioning
`hmc_rho2`	`0.0`	parameter for mass preconditioning
`hmc_rho3`	`0.0`	parameter for mass preconditioning
`hmc_rho4`	`0.0`	parameter for mass preconditioning
`hmc_mpf_mass`	0	number of pseudofermions treated by mass prec.
`hmc_dsf1_mtmp`	0	time scale level, M1 is integrated
`hmc_dsf2_mtmp`	0	time scale level, M1/M2 is integrated
`hmc_dsf3_mtmp`	0	time scale level, M2/M3 is integrated
`hmc_dsf4_mtmp`	0	time scale level, M3/M4 is integrated
`hmc_dsf5_mtmp`	0	time scale level, M4/M5 is integrated
`hmc_dsd`	0	time scale level, LogDetClover is integrated
`hmc_dsg`	0	time scale level, Plaq of gauge action is integrated
`hmc_dsig`	0	time scale level, Lect of gauge action is integrated

3.2.7 Rational Hybrid-Monte Carlo parameters

`hmc_mpf_rhmc_s`	0	number of pseudo-fermions in rational approximation
`hmc_dsf_eor`	0	time scale level, RationalFractions is integrated

3.2.8 Solver parameters

`solver_rest`	1e-8	tolerance of solver for action evaluation
`solver_rest_md`	1e-8	tolerance of solver for molecular dynamics
`solver_rest_cg_ritz`	1e-8	tolerance of cg˙rtiz to compute eigenvalues
`solver_maxiter`	100	maximum iteration for solvers
`solver_ignore_no_convergence`	0	action at when solver reaches max. iter. 2: ignore

		otherwise: abort
`solver_check_solution`	0	if !=0, residual is printed
`solver_mre_vectors`	0	num. of past solutions to get initial guess for CG

`solver_stopping_criterion`	1	if =0, `‖Ax-b‖^2<tol`
		if !=0, `|Ax-b|/|b|<tol`

`solver_outer_solver`	`cg`	`[cg|bicgstab|gmres|cg_mix|bicgstab_mix|gcrodr]`
`solver_inner_solver`	`cg`	`[cg|bicgstab]`
`solver_outer_steps`	20	num. of outer solver iterations
`solver_gcrodr_numarstep`	10	num. of Arnoldi steps when outer solver is gcrodr
`solver_gcrodr_numdefvec`	5	num. of deflation vectors when outer solver is gcrodr

3.2.9 Measurement switches

`measure_cooling_list`	`""`	path of file to specify cooling steps
`measure_polyakov_loop`	0	-th trajectory, is measured
`measure_traces`	0	-th trajectory, is measured
`measure_schrpcac`	0	-th trajectory, f_A and f_P are measured
`measure_minmax`	0	num. of measured min/max eigenvalues
`measure_rhmc_forces`	0	if !=0, forces for rational fractions are printed
`measurement_only`	0	if !=0, HMC is skipped

3.2.10 Tuning parameters

`replay_trick_ntau`	0	trial ntau
`replay_trick_threshold`	1.0	if `< |dH|,` replay
`tuning_approx_range`	0	if !=0, tune rational approx. (see [24])
`tuning_approx_range_list`	`""`	path of file for rational approx.
`tuning_fraction_tolerance`	`""`	path of file for their tolerance

3.2.11 Miscellaneous

`ran_ranlux_level`	`1` *or* `2`
`io_restart_format`	`ildg` *or* `bqcd`

3.3 File naming conventions

File names have these components (see examples given below):

```
progname.run
progname.run.extension
progname.run.trajectory.extension
progname.run.ensemble.trajectory.timeslice.extension
progname.run.ensemble.ensemble2.trajectory.timeslice.extension
```

By default the value of `progname` is `bqcd` (can be changed in `modules/module_bqcd.F90`). `run` is a three digit run number that is set in the input file. Its value cannot be changed in a job chain (there is a consistency check). `trajectory` is a five digit trajectory counter. `timeslice` is a two digit time coordinate of a time slice (it will be extended automatically to three digits if $L_t > 99$).

`ensemble` and `ensemble2` are single digit values of the ensemble index in parallel tempering simulations. Without tempering both values are 1.

3.3.1 `input`, `output` and batch log files

These file names are not automatically being generated by the program. We choose names that fit to the naming scheme:

`bqcd.042`	`input` (command line parameter)
`bqcd.042.res`	`output` (command line parameter)
`bqcd.042.log`	log file from batch system

3.3.2 Restart files in `bqcd` format

`bqcd.042.count`	counters: `run, job, trajectory`
`bqcd.042.pinfo`	plaquette info
`bqcd.042.ran`	state of random number generator
`bqcd.042.1.info`	configuration metadata
`bqcd.042.1.00.u`	timeslice 0 of SU(3) configuration
`bqcd.042.1.01.u`	...
`bqcd.042.1.02.u`	
`bqcd.042.1.03.u`	

3.3.3 Configuration files in `bqcd` format

`bqcd.042.1.1.00015.info`	configuration metadata
`bqcd.042.1.1.00015.00.u`	configuration at trajectory 15, timeslice 0
`bqcd.042.1.1.00015.01.u`	...
`bqcd.042.1.1.00015.02.u`	
`bqcd.042.1.1.00015.03.u`	

The format of these files is identical to the format of `bqcd` restart files.

3.3.4 Restart files in `ildg` format

`bqcd.042.count`	counters: `run, job, trajectory`
`bqcd.042.pinfo`	plaquette info
`bqcd.042.ran`	state of random number generator
`bqcd.042.lime`	configuration

`count`, `pinfo` and `ran` files are the same as in `bqcd` format. The `lime` file contains additional data for consistency checks. This information is not contained in `ildg` configuration files.

3.3.5 Configuration files in `ildg` format

`bqcd.042.xml`	ensemble metadata
`bqcd.042.00010.xml`	metadata of configuration at trajectory 10
`bqcd.042.00010.lime`	binary data of configuration at trajectory 10

3.4 Working with data in ILDG format

3.4.1 Restart files

By default the program works with restart files in its own `bqcd` format. To work with restart files in `ildg` format one has to set

```
io_restart_format ildg
```

in the input parameter file.

3.4.2 SU(3) configuration files and metadata

In order to work with the `ildg` data format one has to set:

```
   io_conf_format ildg
```

The program will then write binary data in *lime* format as well as ensemble and configuration metadata. Currently *lime* I/O is sequential (the `bqcd` format allows for parallel I/O).

Generation of metadata works with templates. On has to provide template files containing placeholders. The syntax for placeholders is `#placeholder#`, for example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gaugeConfiguration xmlns="http://www.lqcd.org/ildg/QCDml/config1.3">
    <management>
        <crcCheckSum>#crc_check_sum#</crcCheckSum>
        <archiveHistory>
            <elem>
                <revisionAction>generate</revisionAction>
                <participant>
                    <name>#participant_name#</name>
                    <institution>#participant_institution#</institution>
                </participant>
                <date>#today#</date>
            </elem>
        </archiveHistory>
    </management>
    ...
```

The following placeholders are available:

```
   #participant_name#
   #participant_institution#
   #machine_name#
   #machine_institution#
   #machine_type#
   #code_name#
```

```
#code_version#
#code_date#
#para_number_steps#
#para_step_size#
#para_time_scale_ratio#
#para_solver_residuum#
#para_rho#
#markov_chain_uri#
#markov_series#
#markov_update#
#Lx#
#Ly#
#Lz#
#Lt#
#beta#
#kappa#
#csw#
#today#
#average_plaquette#
#precision#
#crc_check_sum#
#data_lfn#
```

Placeholders are defined in `ildg/ildg_meta.h`. Some placeholders can be determined from the normal input, for others there are special input parameters:

```
ildg_markov_chain_uri            "mc://UNDEFINED"
ildg_data_lfn_path               "lfn://UNDEFINED"
ildg_participant_name            "UNDEFINED"
ildg_participant_institution     "UNDEFINED"
ildg_machine_name                "UNDEFINED"
ildg_machine_institution         "UNDEFINED"
ildg_machine_type                "UNDEFINED"
```

In any case file names for the templates have to be given:

```
ildg_template_ensemble           "ensemble_template.xml"
ildg_template_conf               "config_template.xml"
```

One can also set prefix and extension of the `ildg` files:

```
ildg_filename_prefix            "qcdsf"
ildg_filename_extension         "lime"
```

This setting would, for example, generate these configuration files:

```
qcdsf.042.xml
qcdsf.042.00010.xml
qcdsf.042.00010.lime
```

3.4.3 Precision

The program can handle `ildg` files in 32- and 64-bit precision. When reading the precision is taken from the `ildg` file. The precision of writing can be set to 32-bit by:

```
ildg_precision         32   SU(3) configurations
ildg_precision_restart 32   restart files (for testing)
```

3.4.4 Example of a complete set of `ildg` settings

```
io_conf_format                "ildg"
ildg_filename_prefix          "qcdsf"
ildg_filename_extension       "lime"
ildg_precision                 64
ildg_template_ensemble        "../data/qcdsf-ensemble-05.xml"
ildg_template_conf            "../data/qcdsf-configuration-04.xml"
ildg_markov_chain_uri         "mc://ldg/qcdsf/clover_nf2/b5p00kp13000-04x04"
ildg_data_lfn_path            "lfn://ldg/qcdsf/clover_nf2/b5p00kp13000-04x04"
ildg_participant_name         "Hinnerk Stueben"
ildg_participant_institution  "ZIB"
ildg_machine_name             "HLRN-II"
ildg_machine_institution      "HLRN"
ildg_machine_type             "SGI ICE 8200"
```

3.5 Output – structure of `res`(ults) file

The output was structured in such a way that it is humanly readable and can be easily processed with *awk*. As a consequence each line begins with a keyword which is followed by data. In addition there are sections. The sections are:

```
>BeginJob
   >BeginHeader
   >EndHeader

   >BeginILDGread
   >EndILDGread

   >BeginForceAcceptance
   >EndForceAcceptance

   >BeginMC
      >BeginILDGwrite
      >EndILDGwrite
      >BeginCooling
      >EndCooling
   >EndMC

   >BeginHMCtest
   >EndHMCtest

   >BeginILDGwrite
   >EndILDGwrite

   >BeginFooter
      >BeginTiming
      >EndTiming
      >Begintiming2
      >Endtiming2
   >EndFooter
>EndJob
```

The Monte-Carlo sections contain tables embedded. A single table can be extracted from the output using *grep* or *awk*, For example:

```
$ grep %mc bqcd.032.res
 T%mc  traj  e  f     PlaqEnergy  exp(-Delta_H) Acc CGcalls CGitTot CGitMax
  %mc     1  1  1   0.4546852837   0.9239828462   0      11     381      37
  %mc     2  1  1   0.4743147510   0.6786574618   1      11     360      34
  %mc     3  1  1   0.4725616177   1.1718570939   1      11     369      34
  %mc     4  1  1   0.4716513122   1.0527896694   1      11     390      36
  %mc     5  1  1   0.4607034796   1.0829331352   1      11     385      35
  %mc     6  1  1   0.4705402342   0.8272890510   1      11     386      36
  %mc     7  1  1   0.4808335455   1.0465054341   1      11     386      36
  %mc     8  1  1   0.4808963557   1.2053582280   1      11     390      36
  %mc     9  1  1   0.4808963557   0.7580344698   0      11     417      39
  %mc    10  1  1   0.4820214435   0.8795287474   1      11     403      38
```

If one is interested in the values only (without the table header)

```
   $ awk '$1 == "%mc"' bqcd.032.res
```

does the job. The general format of a table is:

```
   key  trajectory_counter  value(s)
```

Tables introduced at an early stage contain ensemble indices `e` and `f` in addition. This is because a measurement can always belong to two ensembles when working with parallel tempering.

3.5.1 Header section

The header section contains compile (e.g. program version), input (from parameter file) and runtime (e.g. date) parameters. For historic reasons the keywords used for parameters are different from the input file (previous input files only used positional parameters). Another pecularity are `_1` endings. This ending indicates the ensemble index (what is only necessary for parallel tempering).

3.5.2 ILDG read and write sections

The ILDG sections contain information about the file being read or written, e.g. the filename and the ILDG *logical filename (LFN)*. If a restart file is written in ILDG format the LFN line contains other data, namely the filename, its CRC check-sum, its size in bytes, run-, job- and trajectory counters.

3.5.3 Monte-Carlo sections (`ForceAcceptance`, `MC`, `HMCtest`)

`ForceAcceptance` reports on trajectories that were generated without acceptance test which is sometimes needed at the beginning of a simulation.

`MC` reports on usual Hybrid Monte-Carlo trajectories including optional measurements when a new trajectory is finished.

`HMCtest` reports on a reversibility test. One trajectory is integrated forward and backward. Afterwards energies and fields are compared.

3.5.4 Cooling section

This section contains a cooling history with the measurement of the topological charge.

3.5.5 Footer section

In the footer one finds the end of execution date, the elapsed run time and how many CPUs were used (#CPUs = #cores = MPI processes × OpenMP threads).

3.5.6 Timing sections

Output from profiling. Results are shown in a table that list how often a region was called and how much time was spent in that region:

```
                                          Performance
 region      #calls      time         mean       min       max       Total
                            s      Mflop/s   Mflop/s   Mflop/s     Gflop/s
```

For the most interesting regions the number of floating point were counted. In these cases the table lists the average, minimal and maximal performance per CPU (core) as well as the overall performance.

3.5.7 List of embedded tables

key	meaning
`%fa`	forced acceptance logs
`%mc`	main Hybrid Monte-Carlo logs
`%pr`	rectangular plaquettes
`%Hold`	energies at start of trajectory
`%Hnew`	energies at end of trajectory
`%Hdif`	energy differences (ΔH)
`%Favg`	average forces in molecular dynamics integration
`%Fmax`	maximal forces in molecular dynamics integration
`%Frat`	force ratios: maximal forces / minimal forces
`%Frac`	forces in RHMC molecular dynamics integration
`%Qc`	topological charge from cooling (cooling history)
`%pl`	Polyakov loop
`%tr`	fermionic bulk quantities (traces)
`%it`	counters of cg iterations
`%it4`	counters of $\text{cg}_{\text{32 bit}}$ iterations
`%cgChk`	check of cg solution
`%bicgstabChk`	check of Bicgstab solution
`%bicgstabmixChk`	check of mixed precision Bicgstab solution
`%cg_ritz`	check of Ritz cg solution
`%cg_ritz_dd`	check of Ritz cg solution
`%shift`	check of multi shift solutions
`%egnv`	eigenvalues

3.6 Measurements

Measurements are made at the end of every trajectory. Traces are measured if

```
   measure_traces > 0 .and. mod(trajectory_counter, measure_traces) == 0 .
```

3.6.1 Topological charge

To measure the topological charge with the cooling method a file must be provided in which the cooling steps are defined. One has to set

```
   measure_cooling_list  "filename"
```

in the input file. The file contains the coolings steps after which the topological charge and the plaquette are measured. For example,

```
   1
   2
   5
   10
   20
```

In this case 20 cooling iterations are made and measurements are performed after cooling iteration 1, 2, 5, 10 and 20. The corresponding output looks like this:

```
   >BeginCooling
   T%Qc  traj  e  f  i_cool        Q_cool     PlaqEnergy
    %Qc     1  1  1       1     -0.137916   0.3552673573
    %Qc     1  1  1       2     -0.347026   0.1878487312
    %Qc     1  1  1       5     -0.650364   0.0555456917
    %Qc     1  1  1      10     -0.533735   0.0181235629
    %Qc     1  1  1      20      0.000154   0.0051610597
   >EndCooling
```

3.6.2 Polyakov loop

To measure the Polyakov loop set

```
   measure_polyakov_loop  1
```

in the input file.

3.6.3 Fermionic bulk quantities

To measure fermionic bulk quantities set

```
   measure_traces  1
```

in the input file.

3.6.4 f_A and f_P

To measure f_A and f_P for non-perturbative determination of c_{SW}

```
   measure_schrpcac  2
```

in the input file.
The corresponding output, fAfPhist, looks like this:

```
traj t+1 fA(t)           fP(t)          fA'(T-t)         fP'(T-t)
   2   2 -0.10939226E+02  0.32266553E+02 -0.56909470E+01  0.17781391E+02
   2   3 -0.34826553E+01  0.87987951E+01 -0.19998378E+01  0.40562673E+01
   2   4 -0.10307930E+01  0.19997526E+01 -0.87475663E+00  0.11811545E+01
   4   2 -0.10427659E+02  0.36927566E+02 -0.57629684E+01  0.18859736E+02
   4   3 -0.49157290E+01  0.13352503E+02 -0.31013816E+01  0.54608054E+01
   4   4 -0.19897425E+01  0.38296835E+01 -0.18940843E+01  0.23258797E+01
```

4 Physics

4.1 Gauge actions

The gauge action can be:

- the Wilson action

$$S = S_G^{\text{Wilson}} = \sum_{\text{plaquette}} \frac{1}{3} \operatorname{Re} \operatorname{Tr} \left(1 - U_{\text{plaquette}}\right) \tag{1}$$

- an improved gauge action

$$S_G = \frac{6}{g^2} \left[c_0 \sum_{\text{plaquette}} \frac{1}{3} \operatorname{Re} \operatorname{Tr} \left(1 - U_{\text{plaquette}}\right) + c_1 \sum_{\text{rectangle}} \frac{1}{3} \operatorname{Re} \operatorname{Tr} \left(1 - U_{\text{rectangle}}\right) \right] , \tag{2}$$

  with $c_0 + 8c_1 = 1$. Note that $\beta = 10/g^2$, if one sets

```
gauge_action TREE .
```

4.2 Fermionic actions

The fermionic action can be:

- the Wilson action

$$S_F^{\text{Wilson}} = \sum_x \left\{ \bar{\psi}(x)\psi(x) - \kappa \left[\bar{\psi}(x) U_\mu^\dagger(x - \hat{\mu})(1 + \gamma_\mu) + \bar{\psi}(x) U_\mu(x)(1 - \gamma_\mu) \right] \right\} \tag{3}$$

- the Wilson action plus an explicitly parity-flavour symmetry breaking source term, where τ^3 is the third Pauli matrix

$$S_F = S_F^{\text{Wilson}} + h \sum_x \bar{\psi}(x) i\gamma_5 \tau^3 \psi(x) \tag{4}$$

- the clover $O(a)$ improved Wilson action

$$S_F = S_F^{\text{Wilson}} - \frac{i}{2} \kappa\, c_{\text{SW}} \sum_x \bar{\psi}(x) \sigma_{\mu\nu} F_{\mu\nu}(x) \psi(x) \tag{5}$$

- the clover $O(a)$ improved Wilson action + CP breaking term

$$S_F = S_F^{\rm Wilson} - \frac{i}{2}\kappa\, c_{\rm SW} \sum_x \bar\psi(x)\sigma_{\mu\nu}F_{\mu\nu}(x)\psi(x) + \theta\bar\psi(x)\gamma_5\psi(x) \qquad (6)$$

- fat link fermions

$$S_F = \sum_x \Big\{ \bar\psi(x)\psi(x) - \kappa\,\bar\psi(x)U_\mu^\dagger(x-\hat\mu)[1+\gamma_\mu]\psi(x-\hat\mu) - \kappa\,\bar\psi(x)U_\mu(x)[1-\gamma_\mu]\psi(x+\hat\mu) + \frac{i}{2}\kappa\, c_{\rm SW}\,\bar\psi(x)\sigma_{\mu\nu}F_{\mu\nu}(x)\psi(x)\Big\}\,, \qquad (7)$$

where the gauge links U_μ are replaced by stout links [18]

$$U_\mu \to \tilde U_\mu(x) = e^{iQ_\mu(x)}\,U_\mu(x)\,, \qquad (8)$$

with

$$Q_\mu(x) = \frac{\alpha}{2i}\left[V_\mu(x)U_\mu^\dagger(x) - U_\mu(x)V_\mu^\dagger(x) - \frac{1}{3}{\rm Tr}\,\big(V_\mu(x)U_\mu^\dagger(x) - U_\mu(x)V_\mu^\dagger(x)\big)\right]\,, \qquad (9)$$

where $V_\mu(x)$ is the sum over all staples associated with the link. For SLiNC fermions,

```
fermi_action SLRC
csw          2.65   see [11]
n_stout      1
alpha        0.1
```

4.3 Observables

4.3.1 Gluonic observables

The following gluonic observables can be measured:

- Average plaquette and average rectangular plaquette (both overall, space-like and time-like).

- Topological charge. The topological charge is measured with the field theoretic method after cooling the gauge field configuration.
- Polyakov loop.

4.3.2 Fermionic observables

Some fermionic bulk quantities can be measured (from stochastic estimators):

$$\begin{aligned}
\langle \bar\psi\psi \rangle &= \frac{1}{12V}\langle \mathrm{Tr}(M^{-1})\rangle \qquad \text{('chiral condensate')} \\
\langle \bar\psi\gamma_5\psi \rangle &= \frac{1}{12V}\langle \mathrm{Tr}(\gamma_5 M^{-1})\rangle \\
\langle \Pi^2 \rangle &= \frac{1}{12V}\langle \mathrm{Tr}(M^\dagger M)^{-1}\rangle \qquad \text{('pion norm')}
\end{aligned}$$

5 Algorithms

5.1 Multi timescale integration

In order to explain multi timescale integration we look at the partition function for N_f=2+1 improved Wilson fermions

$$\begin{aligned}
Z &= \int DUD\bar\psi D\psi e^{-S}\,, \\
S &= S_g(\beta) + S_l(\kappa_l, c_{\mathrm{SW}}) + S_s(\kappa_s, c_{\mathrm{SW}})\,,
\end{aligned} \tag{10}$$

where S_g is a gluonic action, S_l is an action for the degenerate u- and d-quarks and S_s is an action for the strange quark. After integrating out fermions

$$S = S_g(\beta) - \ln[\det M_l^\dagger M_l][\det M_s^\dagger M_s]^{\frac{1}{2}}\,. \tag{11}$$

We first apply even-odd preconditioning:

$$\det M_l^\dagger M_l \propto \det(1{+}T^l_{oo})^2 \det Q_l^\dagger Q_l\,, \quad [\det M_s^\dagger M_s]^{\frac{1}{2}} \propto \det(1{+}T^s_{oo})[\det Q_s^\dagger Q_s]^{\frac{1}{2}}\,, \tag{12}$$

where

$$Q = (1+T)_{\mathrm{ee}} - M_{\mathrm{eo}}(1+T)^{-1}_{\mathrm{oo}} M_{\mathrm{oe}}\,, \quad T = \frac{\mathrm{i}}{2} c_{\mathrm{SW}}\,\kappa\,\sigma_{\mu\nu}F_{\mu\nu}\,. \tag{13}$$

We then separate $\det Q_l^\dagger Q_l$ following Hasenbusch [23]

$$\det Q_l^\dagger Q_l = \det W_l^\dagger W_l \det \frac{Q_l^\dagger Q_l}{W_l W_l^\dagger}\,, \qquad W = Q + \rho\,. \tag{14}$$

Finally we modify the standard action to

$$S = S_g + S_{det}^l + S_{det}^s + S_{f1}^l + S_{f2}^l + S_{fr}^s\,, \tag{15}$$

where

$$\begin{aligned}
&S_{det}^l = -2\,\mathrm{Tr}\,\log[1 + T_{oo}(\kappa^l)]\,, \quad S_{det}^s = -\,\mathrm{Tr}\,\log[1 + T_{oo}(\kappa^s)]\,,\\
&S_{f1}^l = \phi_1^\dagger [W(\kappa^l)^\dagger W(\kappa^l)]^{-1}\phi_1\,, \quad S_{f2}^l = \phi_2^\dagger W(\kappa^l)[Q(\kappa^l)^\dagger Q(\kappa^l)]^{-1} W(\kappa^l)^\dagger \phi_2\,,\\
&S_{fr}^s = \sum_{i=1}^{n} \phi_{2+i}^\dagger [Q(\kappa^s)^\dagger Q(\kappa^s)]^{-\frac{1}{2n}} \phi_{2+i}\,.
\end{aligned} \tag{16}$$

We calculate S_{fr} using the RHMC algorithm [24] with optimised values for n and the number of fractions. We now split each term of the action into one ultraviolet and two infrared parts,

$$S_{\mathrm{UV}} = S_g\,, \quad S_{\mathrm{IR}-1} = S_{det}^l + S_{det}^s + S_{f1}^l\,, \quad S_{\mathrm{IR}-2} = S_{f2}^l + S_{fr}^s\,. \tag{17}$$

In [7] we have introduced two different time scales [25] for the ultraviolet and infrared parts of the action in the leap-frog integrator. Here we shall go a step further and put S_{UV}, $S_{\mathrm{IR}-1}$ and $S_{\mathrm{IR}-2}$ on *three separate* time scales,

$$\begin{aligned}
V(\tau) &= \left[V_{\mathrm{IR}-2}\left(\frac{\delta\tau}{2}\right) \;\; A^{m_1} \;\; V_{\mathrm{IR}-2}\left(\frac{\delta\tau}{2}\right) \right]^{n_\tau}\,,\\
A &= V_{\mathrm{IR}-1}\left(\frac{\delta\tau}{2m_1}\right) \;\; B^{m_2} \;\; V_{\mathrm{IR}-1}\left(\frac{\delta\tau}{2m_1}\right)\,,\\
B &= V_{\mathrm{UV}}\left(\frac{\delta\tau}{2m_1m_2}\right) V_Q\left(\frac{\delta\tau}{m_1m_2}\right) V_{\mathrm{UV}}\left(\frac{\delta\tau}{2m_1m_2}\right)\,,
\end{aligned} \tag{18}$$

where $\mathrm{n}_\tau = \tau/(\delta\tau)$ and the Vs are evolution operators of the Hamiltonian. The length of the trajectory τ is taken to be equal to one in our simulations.

5.2 Tuning the rational fraction part

BQCD is able to avoid generating coefficients for the rational approximation every time. Specific sets of coefficients are implemented in advance (see code in `fermi/rhmc`). If the approximation range of generated coefficients is wider than the condition number of X, BQCD automatically shifts as follows, when range does not cover actual [min, max] of X

$$\begin{aligned} X^{\alpha} &= \beta^{-\alpha}(\beta X)^{\alpha} \\ &\approx \beta^{-\alpha}\big[c_0 + \sum_{i=1} \frac{c_i}{\beta X + d_i}\big] \end{aligned} \tag{19}$$

where β is the inverse of the minimum eigenvalue of X and c_i, d_i is generated by Remez algorithm with range for one to the condition number of X, $[1, C(X)]$.

BQCD also supports to tune the rational approximation by given range and approximation degree in a file

```
tuning_approx_range_list  "rangelist"
```

Example ***`rangelist`*** file:

```
1 11 15 2
2 10 14 2
```

1st column is ID for rational approximation. It starts from 1 and is consistent with `rid` in `check para` region printed to stderr. 2nd column is degree of approximation used to approximate $1/X^{-n}$ which is used at MD steps. 3rd column is degree of approximation used to approximate $1/X^{-2n}$ and $1/X^{+2n}$ which are used at action calculation. 4th column is a margin factor of appoximation range.

To relax the solver tolerance one can specify

```
tuning_fraction_tolerance  "fractiontolerance"
```

Example ***fractiontolerance*** file:

```
0.0011
0.55
2.2
0.11
```

In this case, toleraces for 1st, 2nd, 3rd and 4th shift are relaxed by factors of 2000, 4, 1 and 20. This tuning works only if

```
tuning_approx_range  !=0
```

and BQCD is compiled with FMLIB.

6 Implementation issues

In this section we explain why things in BQCD are the way they are.

6.1 Programming language

BQCD is mainly written in Fortran. The reasons for this decision were the following. First of all the programmer was a Fortran programmer. C was not chosen because it was not plausible to write a program that used complex arithmetic almost throughout in a language that had no support for that (or better to say only had added complex arithmetic recently). C++ was considered but the feeling was that at least a first implementation would have been completed before it would be understood how to use C++ effectively.

The main disadvantages of this decision became visible when the program became more and more complex. Fortran90 eventually supported dynamical memory management but there was no support for dynamical algorithms, i.e., there were no function pointers (which became available in Fortran2003, but still it is not clear which parts of Fortran2003 are supported by various compilers). The second disadvantage is that working with classes instead of arrays would offer new possibilities like introducing different orderings of the lattice sites which is an interesting optimisation option (of course this can be done in the current approach but the program will become less readable).

One design goal was to write readable code. We have to leave to the reader to check how well this was achieved.

6.2 Preprocessing

6.2.1 C preprocessor

The C preprocessor was employed from the beginning. It is used for conditional compilation and for macro processing. By convention all BQCD source files have the suffix `.F90`. The suffix of the preprocessed file is `.f90`. In some cases several `.f90` files are being generated from a `.F90` file. The `.f90` file are being compiled. They are always kept such that one can always check the result of preprocessing.

Macro names are all uppercase (sometimes mixed case). Fortran code is always lowercase.

There are macros with and without arguments. Macros without arguments are used for defining constants and datatypes. The motivation for this was mainly readability (and aesthetics we have to admit), compare 'BQCD style'

```
# include "defs.h"

GAUGE_FIELD :: u
COMPLEX :: x, y

...
x = TWO * y + u(...)
```

with a pure Fortran style:

```
use defs

type(gauge_field) :: u
complex(rkind) :: x, y

...
x = const%two * y + u%u(...)
```

Macros with arguments are used as tools, e.g. the `ASSERT` and `ALLOCATE` macros and for simplifying programming. When marcos are used for the latter purpose it is a good idea to look at the `.F90` and the generated `.f90` files when studying the source code.

Over the years C preprocessors became more picky. For example, the GNU preprocessor now refuses to process general Fortran90 code. In some situations C preprocessors complain about the Fortran `%` character (which is a separator in Fortran but an operator in C) and the dots in Fortran operators like `.or.` (because dots are separators in C).

6.2.2 m4 macro preprocessor

The *m4* macro processor is also used. In this case there are two preprocessing steps: first a `.F90` file is generated and then a `.f90` file.

6.3 Fortran modules

Modules are used for storing global data, type definitions and a few interface definitions. Global data is always readonly except for its initialisation (there are a few exceptions to this rule). In general modules do not contain functions or subroutines. The idea behind this is that it should always be possible to call functions or subroutines from C/C++ if it should become necessary to do so. Modules that are only used within the same file are put into that file. Modules that are used by more than one file are put into the `modules` subdirectory.

6.4 Precision

Also from the beginning it was foreseen that one might be interested in multi-precision code. In principle one can compile any version of BQCD using single precision arithmetic if one defines:

```
#define RKIND 4
#define BQCD_REAL mpi_real4
```

This feature was used much later to generate multi-precision code. The recipe is the following.

- The original source file, `foo.F90` say, is compiled as usual with double precision arithmetic.

- A single precision version `foo_r4.F90` is generated. It contains only four lines:

  ```
  #define PRECISION_R4
  #include "defs.h"
  #include "defs_r4.h"
  #include "foo.F90"
  ```

- `defs_r4.h` contains macros for renaming all subroutines and functions, e.g.:

  ```
  #define fun1 fun1_r4
  #define fun2 fun2_r4
  ```

Again there is also a Fortran way of handling the multi-precision problem. One can use interfaces and overloading (see `su3sc/module_sc.F90`).

6.5 Parallelisation

The early versions of BQCD were parallelised by using the *shmem* library from Cray. Later an MPI version was added and also a single processor version that can be compiled without any message passing library. Currently only the MPI and single processor version work (the routines using *shmem* are still contained in the distribution). All files that use calls to message passing routines are located in directory `comm`. Which message passing library to use can be selected in `Makefile.var`.

BQCD is parallelised with OpenMP in addition. On the Hitachi SR8000 this lead to great performance by overlapping communication and computation. In order to facilitate OpenMP programming, BQCD routines contain typically only one loop. It is then straightforward to add OpenMP private and reduction declarations: the candidates can be found in the type declarations of the routine (`implicit none` is used throughout).

The parallel design is such that results are independent of the numbers of processes used. This is true up to rounding errors introduced by global summations. In a job chain one can change the number of processes in every job. This feature was implemented, of course, in order to be able to adapt to changing job mixes at computer centres.

6.6 Random numbers

The first random number generator used was *ranf* by Cray because it has the ability to jump to an arbitrary position in the sequence of random numbers and this operation is not much more expensive than generating the next random number. This skipping of random numbers was reversely engineered such that it was also available on other computers. Skipping is used to generate distributed random numbers in such a way that results become independent of the number of processes.

For running production *ranlux* [26, 27] is the recommended random number generator. Currently the same skipping mechanism as for *ranf* is used. As a consequence generation of random numbers is not parallelised (random numbers do have to be communicated, but every process generates all random numbers, it only picks the ones that belong to its local lattice). Up to now this is not a severe restriction in practice but random number generation with *ranlux* should become truly parallel in future.

6.7 Saving and reading configurations

BQCD's file format for configuration was designed to enable simple parallel I/O. Metadata are kept separate (`.info` files) from binary data (`.u` files). To enable parallel I/O one binary file is written for each timeslice. Again the design is such that everything works on any number of processes. All binary data is written to disk in big endian format. BQCD automatically converts to little endian if necessary. Only two columns of SU(3) matrices are stored. Checksums are calculated on the fly and added to the metadata. The checksums can be verified with standard *cksum* command.

Internally BQCD differentiates between restart files and files that are supposed to be saved. However, the file structures are the same.

Alternatively *lime* files can be written conforming to the ILDG standard [16]. Restart files are always written in 64 bit precision. Configurations can also be saved in 32 bit precision. This kind of I/O is not parallelised.

6.8 Performance measurements and profiling

A simple profiling mechanism was built in. It can be switched on by defining the `TIMING` macro. If it is switched off there is no overhead. In the most important routines operations were counted manually in order to get performance figures.

6.9 Fermionic boundary conditions

BQCD started with having only one copy of the gauge field. Fermionic boundary conditions were imposed by multiplying SU(3) links with -1 accordingly. Flipping boundary conditions between gluonic and fermionic is handled by subroutine `flip_bc()`. It has to be called before and after fermionic operations.

One can optimise the hopping matrix multiplication by introducing a copy of the gauge field that has an optimised storage ordering. This copy then has fermionic boundary conditions (and might also include factors 2 from the $(1 \pm \gamma_4)$ projection).

6.10 C interface

C routines have to be called here and there. Examples are checksum calculations, *ranlux* random numbers and I/O of *lime* files. The Fortran name mangling scheme is selected by defining `NamesToLower`, `NamesToLOwer_` or `NamesToLower__`, respectively.

6.11 Input parsing

The input parser checks whether keywords are known but does not check the rest of the line! It is easy to add a new keyword. New keywords can be introduced by adding them to `modules/module_input.h`. A similar mechanism was used to introduce placeholders for ILDG metadata files (see `ildg/ildg_meta.F90`).

A γ-matrix definitions

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & +i \\ 0 & 0 & +i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_2 = \begin{pmatrix} 0 & 0 & 0 & +1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ +1 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_3 = \begin{pmatrix} 0 & 0 & +i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & +i & 0 & 0 \end{pmatrix}$$

$$\gamma_4 = \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$\gamma_5 = \begin{pmatrix} 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \\ +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \end{pmatrix}$$

B Preprocessor flags – `MYFLAGS`

This appendix we list preprocessor flags that are set via `MYFLAGS` in the `Makefile`s.

`ALTIX`

Settings for SGI Altix, in particular for the *shmem* communication library.

`ARPACK`

Settings for ARPACK.

`BAGEL`

Use Bagel code for the hopping matrix multiplication.

`CRAY`

Settings for the Cray compiler.

`DOEDEO`

If not defined (default) the even/odd ordered hopping matrices appear in the preconditioned matrix as $D_{\mathrm{eo}}D_{\mathrm{oe}}$ and as $D_{\mathrm{oe}}D_{\mathrm{eo}}$.

`GAMMA_NOTATION_CHROMA`, `GAMMA_NOTATION_CHIRAL`, `GAMMA_NOTATION_DDHMC`, `GAMMAC`

Alternative conventions for γ-matrices.

`IBM`

Additions for IBM, in particular data alignment on Blue Gene.

`INTEL`

Settings for the Intel compiler.

`I_TIMES_MACRO`

At many places a statement function `i_times(z)` is used. It returns `z` multiplied by the imaginary unit. (No floating point operations are necessary here.) Because statement functions are outmoded one can achieve the same by using a preprocessor macro.

`LAPACK`

Settings for LAPACK.

`LIBDI`

Use *i*mproved *libd* (often fastest Fortran hopping matrix multiplication).

`LongLong`

For C source files: 8 byte integers are `long long`.

`MPI_1`

Replace some calls to MPI-2 routines by equivalent calls to MPI-1 routines. (Necessary for older versions of the SGI Message Passing Toolkit).

`NamesToLower`, `NamesToLower_`, `NamesToLower__`

For C source files: set Fortran name mangling scheme, lower case with no, one or two underscores appended.

`OMTDTD`

If not defined the preconditioned clover improved fermion matrix reads $M = T_{\text{ee}} - D_{\text{eo}} T_{\text{oo}}^{-1} D_{\text{oe}}$ and $M = 1 - T_{\text{ee}}^{-1} D_{\text{eo}} T_{\text{oo}}^{-1} D_{\text{oe}}$ otherwise.

`PATHSCALE`

Settings for the PathScale compiler.

`TIMING`

Switch profiling on (time and performance measurement).

`_OPENMP`

This is the standard C preprocessor variable that is set when compiling with OpenMP enabled. However, it has to be defined explicitly here, because preprocessing and compilation are two separate steps.

C Running on Blue Gene/P

For Blue Gene/P a very fast assembler version of the hopping matrix multiplication was implemented by Thomas Streuer. To use it, one has to set

```
libd = 520
```

in `Makefile.var`. In addition the following arguments must be given to `mpirun`:

```
mpirun -np ... -mode VN \
               -mapfile XYZT \
               -env DCMF_INJCOUNTER=7 \
               -env DCMF_RECCOUNTER=7 \
               ...
```

On Blue Gene/P BQCD decomposes the lattices automatically according to the physical dimension of the torus network. The input parameters `processes` and `process_mapping` are being ignored.

The assembler code requires that there is a torus network, i.e. one has to work at least on a midplane. For testing on small lattices one has to chose another version of the hopping parameter multiplication. In any case the same automatic lattice decomposition will occur.

References

[1] Y. Nakamura and H. Stüben, `PoS(Lattice 2010)040`.

[2] E. M. Ilgenfritz, W. Kerler and H. Stüben, Nucl. Phys. Proc. Suppl. **83** (2000) 831, [arXiv:hep-lat/9908022].

[3] E. M. Ilgenfritz, W. Kerler, M. Müller-Preussker and H. Stüben, Phys. Rev. D **65** (2002) 094506 [arXiv:hep-lat/0111038].

[4] H. Stüben [QCDSF-UKQCD Collaboration], Nucl. Phys. Proc. Suppl. **94** (2001) 273, [arXiv:hep-lat/0011045].

[5] E. M. Ilgenfritz, W. Kerler, M. Müller-Preussker, A. Sternbeck and H. Stüben, Phys. Rev. D **69** (2004) 074511, [arXiv:hep-lat/0309057].

[6] A. Sternbeck, E. M. Ilgenfritz, W. Kerler, M. Müller-Preussker and H. Stüben, Nucl. Phys. Proc. Suppl. **129** (2004) 898 [arXiv:hep-lat/0309059].

[7] A. Ali Khan, T. Bakeyev, M. Göckeler, R. Horsley, D. Pleiter, P. Rakow, A. Schäfer, G. Schierholz, H. Stüben [QCDSF Collaboration], Phys. Lett. B **564** (2003) 235 [arXiv:hep-lat/0303026].

[8] A. Ali Khan, T. Bakeyev, M. Göckeler, R. Horsley, D. Pleiter, P.E.L. Rakow, A. Schäfer, G. Schierholz, H. Stüben [QCDSF Collaboration], Nucl. Phys. Proc. Suppl. **129** (2004) 853 [arXiv:hep-lat/0309078].

[9] M. Göckeler *et al.* [QCDSF Collaboration], PoS **LAT2007** (2007) 041 [arXiv:0712.3525 [hep-lat]].

[10] N. Cundy *et al.* [QCDSF-UKQCD Collaborations], PoS **LATTICE2008** (2008) 132 [arXiv:0811.2355 [hep-lat]].

[11] N. Cundy *et al.*, Phys. Rev. D **79** (2009) 094507 [arXiv:0901.3302 [hep-lat]].

[12] W. Bietenholz *et al.* [QCDSF-UKQCD Collaborations], PoS **LAT2009** (2009) 102 [arXiv:0910.2963 [hep-lat]].

[13] Y. Nakamura *et al.*, AIP Conf. Proc. **756** (2005) 242 [Nucl. Phys. Proc. Suppl. **140** (2005) 535] [arXiv:hep-lat/0409153].

[14] V. G. Bornyakov *et al.*, arXiv:0910.2392 [hep-lat].

[15] H. Baier *et al.*, arXiv:0911.2174 [hep-lat].

[16] `http://ildg.sasr.edu.au/Plone`

[17] K. Symanzik, Nucl. Phys. **B226** (1983) 187.

[18] C. Morningstar and M. J. Peardon, Phys. Rev. **D69** (2004) 054501 [hep-lat/0311018].

[19] S. Capitani, S. Dürr and C. Hoelbling, JHEP 0611 (2006) 028 [hep-lat/0607006].

[20] H. Perlt *et al.* [QCDSF Collaboration], PoS(LATTICE 2007)250 [arXiv:0710.0990].

[21] S. Boinepalli *et al.*, Phys. Lett. **B616** (2005) 196 [hep-lat/0405026].

[22] J.M.Zanotti *et al.*, Phys. Rev. **D71** (2005) 034510 [hep-lat/0405015].

[23] M. Hasenbusch, Phys. Lett. **B519** (2001) 177 [hep-lat/0107019].

[24] M. A. Clark and A. D. Kennedy, Nucl. Phys. Proc. Suppl., **129** (2004) 850 [hep-lat/0309084].

[25] J. C. Sexton and D. H. Weingarten, Nucl. Phys. **B380** (1992) 665.

[26] M. Lüscher, Comput. Phys. Commun. **79** (1994) 100 [arXiv:hep-lat/9309020].

[27] `http://luscher.web.cern.ch/luscher/ranlux/index.html`