We publish BQCD as free software under the GNU General Public License. BQCD is a Hybrid Monte-Carlo program that simulates lattice QCD with dynamical Wilson fermions. It is one of the main production programs of the QCDSF collaboration. The program can simulate 2 and 2 + 1 fermion flavours with pure, clover improved, and stout smeared fat link Wilson fermions as well as standard plaquette, and an improved (rectangle) gauge action. The single flavour is simulated with the Rational Hybrid Monte-Carlo algorithm.
1. Introduction

Berlin quantum chromodynamics program (BQCD) is a Hybrid Monte-Carlo [1] program for simulating lattice QCD with dynamical Wilson fermions. The development of BQCD started in 1998 for the two flavour case and the standard Wilson action. It was written for a study of parallel tempering [2]. At that time the whole parallelisation framework was completed. Soon the program was extended in two different directions. The first direction was the implementation of clover \(O(a)\) improvement of the fermion action. With the availability of clover improvement BQCD became one of the main production codes of the QCDSF collaboration [3]. The second direction was the addition of an external field to the standard Wilson action in order to study the Aoki phase [4].

The next milestone was the implementation of the Hasenbusch trick [5, 6]. Starting in 2006 the code has been largely extended to enable simulations including a third fermion flavour [7, 8, 9, 10]. This extension includes the implementation of Rational Hybrid Monte-Carlo (RHMC) [11] for the simulation of the third quark flavour as well as many algorithmic and performance improvements.

The code is also being used by the DIK Collaboration for simulations at finite temperature [12, 13]. Several people took BQCD as a starting point for adding their own code for measurements. The plan of the QPACE project [14] to port BQCD, in particular the fermion matrix multiplication and solvers [15], to their machine has triggered the publication of the code as free software under the GNU General Public License on the occasion of this Lattice conference. The source and a manual can be downloaded from [16]. A description on building and testing binaries can be found in the manual.

2. Actions

The program can simulate the QCD with the following actions. The gauge action can be the Wilson action

\[
S = S_G^{\text{Wilson}} = \sum_{\text{plaquette}} \frac{1}{3} \text{Re} \, \text{Tr} (1 - U_{\text{plaquette}}) \tag{2.1}
\]

or a Symanzik improved gauge action

\[
S_G = 6 g^2 \left[ c_0 \sum_{\text{plaquette}} \frac{1}{3} \text{Re} \, \text{Tr} (1 - U_{\text{plaquette}}) + c_1 \sum_{\text{rectangle}} \frac{1}{3} \text{Re} \, \text{Tr} (1 - U_{\text{rectangle}}) \right] , \tag{2.2}
\]

with \(c_0 + 8c_1 = 1\). The fermion action can be the Wilson action

\[
S_F^{\text{Wilson}} = \sum_x \{ \bar{\psi}(x) \psi(x) - \kappa \left[ \bar{\psi}(x) U_{\mu}(x - \hat{\mu})(1 + \gamma_\mu) \psi(x - \hat{\mu}) + \bar{\psi}(x) U_{\mu}(x)(1 - \gamma_\mu) \psi(x + \hat{\mu}) \right] \} , \tag{2.3}
\]

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

\[
S_F = S_F^{\text{Wilson}} + h \sum_x \bar{\psi}(x) i\gamma_\tau^3 \psi(x) , \tag{2.4}
\]

the clover action

\[
S_F = S_F^{\text{Wilson}} - \frac{i}{2} \kappa c_{SW} \sum_x \bar{\psi}(x) \sigma_{\mu \nu} F_{\mu \nu}(x) \psi(x) , \tag{2.5}
\]
the clover action plus a CP breaking term

\[ S_F = S_F^{\text{Wilson}} - \frac{i}{2} \kappa c_{SW} \sum_x \bar{\psi}(x) \sigma_{\mu \nu} F_{\mu \nu}(x) \psi(x) + \theta \bar{\psi}(x) \gamma_5 \psi(x) \]  

(2.6)

or a stout smeared fat link action (any term containing gauge links can be smeared), in particular the SLiNC fermion action \[ S_F = \sum_x \left\{ \bar{\psi}(x) \psi(x) - \kappa \bar{\psi}(x) U_\mu(x) [1 + \gamma_\mu] \psi(x) \right\}, \]  

where the gauge links \( U_\mu \) are replaced by stout links \[ U_\mu \rightarrow \tilde{U}_\mu(x) = e^{i Q(x) \mu(x)} U_\mu(x), \]  

(2.8)

with \[ Q_\mu(x) = \frac{\alpha}{2l} \left[ V_\mu(x) U_\mu^\dagger(x) - U_\mu(x) V_\mu^\dagger(x) - \frac{1}{3} \text{Tr} \left( V_\mu(x) U_\mu^\dagger(x) - U_\mu(x) V_\mu^\dagger(x) \right) \right], \]  

(2.9)

where \( V_\mu(x) \) is the sum over all staples associated with the link. Boundary condition for the gauge field are periodic in all directions. For the fermions boundary conditions can be chosen to be anti-periodic or periodic for each dimension.

3. Observables

The following gluonic observables can be measured: the average plaquette and average rectangular plaquette, the topological charge (the topological charge is measured with the field theoretic method after cooling the gauge field configuration), the Polyakov loop. In addition some fermionic bulk quantities can be measured (from stochastic estimators):

\[ \langle \bar{\psi} \psi \rangle = \frac{1}{12V} \langle \text{Tr}(M^{-1}) \rangle \quad (\text{‘chiral condensate’}) \]

\[ \langle \bar{\psi} \gamma_5 \psi \rangle = \frac{1}{12V} \langle \text{Tr}(\gamma_5 M^{-1}) \rangle \]

\[ \langle \Pi^2 \rangle = \frac{1}{12V} \langle \text{Tr}(M^2 M)^{-1} \rangle \quad (\text{‘pion norm’}) \]

4. Algorithmic improvements

4.1 Integrators

HMC trajectories can be integrated with leapfrog or Omelyan [18] integrators. Multi timescale integration is possible with up to six time scales. In the following we explain a multi timescale setup that is used in production for \( N_f = 2 + 1 \) improved Wilson fermions. Starting point is the partition function

\[ Z = \int DU D \bar{\psi} D \psi e^{-S} \]  

(4.1)

\[ S = S_g(\beta) + S_l(\kappa_l, c_{SW}) + S_s(\kappa_s, c_{SW}) \]  

(4.2)
where $S_g$ is a gluon 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

$$S = S_g(\beta) - \ln[\det M^l_j M_l][\det M^s_j M_s]^{1/2}. \tag{4.3}$$

First even-odd preconditioning is applied

$$\det M^l_j M_l \propto \det(1 + T_{oo}^l)^2 \det Q^l_j Q_l \tag{4.4}$$

$$[\det M^s_j M_s]^{1/2} \propto \det(1 + T_{oo}^s)[\det Q^s_j Q_s]^{1/2} \tag{4.5}$$

where

$$Q = (1 + T)_{oo} - M_{oo}(1 + T)^{-1}_{oo} M_{oe} \tag{4.6}$$

$$T = \frac{i}{2} c_{SW} \kappa \sigma_{\mu \nu} F_{\mu \nu}. \tag{4.7}$$

Then $\det Q^l_j Q_l$ is separated following Hasenbusch [5]

$$\det Q^l_j Q_l = \det W^l_j W_l \det \frac{Q^l_j Q_l}{W^l_j W^l_l}, \quad W = Q + \rho. \tag{4.8}$$

Finally the standard action is modified to

$$S = S_g + S^l_{det} + S^s_{det} + S^l_{f1} + S^l_{f2} + S_s^{fr}, \tag{4.9}$$

where

$$S^l_{det} = -2 \text{Tr} \log[1 + T_{oo}(\kappa^l)] \tag{4.10}$$

$$S^s_{det} = -2 \text{Tr} \log[1 + T_{oo}(\kappa^s)] \tag{4.11}$$

$$S^l_{f1} = \phi^l_1 [W(\kappa^l)]^l W(\kappa^l)^{-\dagger} \phi_1 \tag{4.12}$$

$$S^l_{f2} = \phi^l_2 W(\kappa^l)[Q(\kappa^l)^l Q(\kappa^l)]^{-1} W(\kappa^l)^{\dagger} \phi_2 \tag{4.13}$$

$$S^{fr}_s = \sum_{i=1}^n \phi^{fr}_{2+i} [Q(\kappa^s)^l Q(\kappa^s)]^{-1} \phi^{fr}_{2+i} \tag{4.14}$$

We calculate $S^{fr}$ using the RHMC algorithm [11] with optimised values for $n$ and the number of fractions. Each term of the action is split into one ultraviolet and two infrared parts,

$$S_{UV} = S_g \tag{4.15}$$

$$S_{IR-1} = S^l_{det} + S^s_{det} + S^l_{f1} \tag{4.16}$$

$$S_{IR-2} = S^l_{f2} + S^{fr}_s. \tag{4.17}$$

In the leap-frog integrator $S_{UV}$, $S_{IR-1}$ and $S_{IR-2}$ are put on three separate time scales,

$$V(\tau) = \left[ V_{IR-2} \left( \frac{\delta \tau}{2} \right)^2 A^{m_1} V_{IR-2} \left( \frac{\delta \tau}{2} \right)^n a \right. \tag{4.18}$$

$$A = V_{IR-1} \left( \frac{\delta \tau}{2m_1} \right)^2 B^{m_2} V_{IR-1} \left( \frac{\delta \tau}{2m_1} \right) \tag{4.19}$$

$$B = V_{UV} \left( \frac{\delta \tau}{2m_1 m_2} \right)^2 V_Q \left( \frac{\delta \tau}{m_1 m_2} \right) V_{UV} \left( \frac{\delta \tau}{2m_1 m_2} \right) \tag{4.20}$$

where $n = \tau/(\delta \tau)$ and the $V$s are evolution operators of the Hamiltonian.
4.2 Solvers

Besides the standard conjugate gradient (cg) solver BiCGstab and GMRES were implemented. Variants with mixed precision arithmetics are available for cg and BiCGstab. In order to reduce time spent in the solver chronological inversion [19] is employed and even-odd preconditioning as well as Schwarz preconditioning [20] are used.

5. Implementation details

The code is mostly written Fortran. The C preprocessor is used for preprocessing in general and the m4 macro processor for a few files. A simple mechanism is employed to automatically generate multi precision versions from the same source. BQCD is parallelised with MPI and OpenMP. The first version of the program was parallelised for a Cray T3E with the shmem library. shmem can still be used in the hopping matrix multiplication.

Random numbers are generated with ranlux [21, 22]. Binary data (SU(3) configurations) can either be stored in a native BQCD format or in the International Lattice DataGrid (ILDG) [23] format. The input parameter file and the log file are simple text files that have a keyword value(s) structure. Important parts of the program are instrumented for time profiling and performance measurements.

<table>
<thead>
<tr>
<th>#racks</th>
<th>#cores</th>
<th>hopping matrix multiplication (Fortran)</th>
<th>cg solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>per core</td>
<td>overall</td>
</tr>
<tr>
<td>1/2</td>
<td>2048</td>
<td>344</td>
<td>0.70</td>
</tr>
<tr>
<td>1</td>
<td>4096</td>
<td>429</td>
<td>1.76</td>
</tr>
<tr>
<td>2</td>
<td>8192</td>
<td>415</td>
<td>3.40</td>
</tr>
<tr>
<td>4</td>
<td>16384</td>
<td>407</td>
<td>6.67</td>
</tr>
</tbody>
</table>

**Table 1:** Performance figures for a $48^3 \times 96$ lattice obtained with the pure Fortran implementation on a Blue Gene/P.

<table>
<thead>
<tr>
<th>#racks</th>
<th>#cores</th>
<th>hopping matrix multiplication (assembler)</th>
<th>cg solver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>per core</td>
<td>overall</td>
</tr>
<tr>
<td>1/2</td>
<td>2048</td>
<td>1057</td>
<td>2.16</td>
</tr>
<tr>
<td>1</td>
<td>4096</td>
<td>1061</td>
<td>4.35</td>
</tr>
<tr>
<td>2</td>
<td>8192</td>
<td>1019</td>
<td>8.35</td>
</tr>
<tr>
<td>4</td>
<td>16384</td>
<td>923</td>
<td>15.11</td>
</tr>
</tbody>
</table>

**Table 2:** Performance figures for a $48^3 \times 96$ lattice obtained with an assembler implementation of the hopping matrix multiplication on a Blue Gene/P.
6. Performance

The program scales very well to large numbers of cores. The pure Fortran case even displays some super-linear speedup (see Figure 1). For Blue Gene and Itanium2 assembler implementations of the hopping matrix multiplication were provided by Th. Streuer. Performance figures for a $48^3 \times 96$ lattice obtained on a Blue Gene/P are given in Tables 1 and 2. The assembler implementations makes it possible to overlap communication with computation. This boosts the performance of the hopping multiplication of up to a factor of 3.1 and the whole conjugate gradient solver by a factor of 1.6 to 2.1 compared with the pure Fortran version. With this code it is possible to run simulations at a sustained overall speed of 11.2 Tflop/s.

7. Acknowledgements

We would like to thank Gerrit Schierholz, Roger Horsley, Dirk Pleiter, Paul Rakow and James Zanotti for support, stimulating discussions and bug reports, Thomas Streuer for providing assembler code and Andrea Nobile for discussions on Schwarz preconditioning. The computations were performed on the Blue Gene/P at Jülich Supercomputer Centre, Jülich, Germany.
References


