QDP: QCD Data Parallel Interface in C

This is a brief introduction to using QDP by giving a few simple examples. It will eventually become a simple primer on using the library. This document is available online at `http://www.physics.utah.edu/~osborn/scidac` and will be updated periodically. QDP and other SciDAC libraries are available at `http://www.lqcd.org` .

This is still beta software and should not be used for production code yet. Also there may still be some changes to the interface, but they should only be minor.

James C. Osborn

Feb. 18, 2003

simple example

```
#define QDP_Precision 'F'
#define QDP_Nc 3
#include <qdp.h>

int main(int argc, char *argv[]) {
  int lattice_size[4] = { 4,4,4,4 };
  QLA_Complex z;
  QDP_ColorMatrix *mat;
  QDP_ColorVector *v1, *v2;

  QDP_initialize(argc, argv);         /* start QDP */
  QDP_set_latsize(4, lattice_size);   /* set lattice size */
  QDP_create_layout();                /* create layout */

  mat = QDP_create_M();               /* create some fields */
  v1 = QDP_create_V();
  v2 = QDP_create_V();

  set_mat(mat); set_vec(v1);          /* set fields somehow */

  /* v2[i] = mat[i] * v1[i] for all sites i */
  QDP_V_eq_M_times_V(v2, mat, v1, QDP_all);

  /* z = sum v1[i] . v2[i] for even sites i */
  QDP_c_eq_V_dot_V(&z, v1, v2, QDP_even);

  if(QDP_this_node==0)                /* print result on node 0 */
    printf("result = %f%+fi\n", QLA_real(z), QLA_imag(z));

  QDP_destroy_M(mat);                 /* free fields */
  QDP_destroy_V(v1);
  QDP_destroy_V(v2);

  QDP_finalize();                     /* shutdown QDP */
  return 0;
}
```

plaquette

```
QLA_Real plaquette(QDP_ColorMatrix *link[])
{
  int mu, nu;
  QLA_Real plaq=0, tplaq;
  QDP_ColorMatrix *temp1, *temp2, *temp3, *temp4;

  temp1 = QDP_create_M(); temp2 = QDP_create_M();
  temp3 = QDP_create_M(); temp4 = QDP_create_M();

  for(mu=0; mu<3; ++mu) {
    for(nu=mu+1; nu<4; ++nu) {

      QDP_M_eq_sM(temp1, link[nu], QDP_neighbor[mu], QDP_forward, QDP_all);
      QDP_M_eq_sM(temp2, link[mu], QDP_neighbor[nu], QDP_forward, QDP_all);

      QDP_M_eq_Ma_times_M(temp3, link[nu], link[mu], QDP_all);

      QDP_M_eq_M_times_M(temp4, temp3, temp1, QDP_all);

      QDP_r_eq_re_M_dot_M(&tplaq, temp2, temp4, QDP_all);

      plaq += tplaq;
    }
  }

  QDP_destroy_M(temp1); QDP_destroy_M(temp2);
  QDP_destroy_M(temp3); QDP_destroy_M(temp4);

  return plaq;
}
```

plaquette (one global sum)

```
QLA_Real plaquette(QDP_ColorMatrix *link[])
{
  int mu, nu;
  QLA_Real plaq;
  QDP_ColorMatrix *temp1, *temp2, *temp3, *temp4;
  QDP_Real *treal1, *treal2;

  temp1 = QDP_create_M(); temp2 = QDP_create_M();
  temp3 = QDP_create_M(); temp4 = QDP_create_M();
  treal1 = QDP_create_R();
  treal2 = QDP_create_R();
  QDP_R_eq_zero(treal2, QDP_all);

  for(mu=0; mu<3; ++mu) {
    for(nu=mu+1; nu<4; ++nu) {
      QDP_M_eq_sM(temp1, link[nu], QDP_neighbor[mu], QDP_forward, QDP_all);
      QDP_M_eq_sM(temp2, link[mu], QDP_neighbor[nu], QDP_forward, QDP_all);

      QDP_M_eq_Ma_times_M(temp3, link[nu], link[mu], QDP_all);
      QDP_M_eq_M_times_M(temp4, temp3, temp1, QDP_all);

      QDP_R_eq_re_M_dot_M(treal1, temp2, temp4, QDP_all);
      QDP_R_peq_R(treal2, treal1, QDP_all);
    }
  }
  QDP_r_eq_sum_R(&plaq, treal2, QDP_all);

  QDP_destroy_R(treal1); QDP_destroy_R(treal2);
  QDP_destroy_M(temp1); QDP_destroy_M(temp2);
  QDP_destroy_M(temp3); QDP_destroy_M(temp4);

  return plaq;
}
```

subsets

```
int timeslice_func(int x[])
{
  return x[3];
}

void w_meson2(QDP_DiracPropagator *src1, QDP_DiracPropagator *src2)
{
  QLA_Complex *prop0;
  QDP_DiracPropagator *localmat;
  QDP_Subset *timeslices;

  prop0 = (QLA_Complex *) malloc( nt * sizeof(QLA_Complex) );

  localmat = QDP_create_P();

  timeslices = QDP_create_subset(timeslice_func, nt);

  /* Multiply by gamma_0 for gamma_5 - gamma_5*gamma_0 correlator */
  QDP_P_eq_P_times_gamma( localmat, src1, 0, QDP_all );

  /* trace over propagators */
  QDP_c_eq_P_dot_P_multi( prop0, localmat, src2, timeslices, nt );

  QDP_destroy_subset(timeslices);
  QDP_destroy_P(localmat);
  free(prop0);
}
```

optimizing shifts

consider:

```
QDP_r_eq_norm2_V(&x, src, QDP_all);
QDP_V_eq_sV(dest, src, QDP_neighbor[0], QDP_backward, QDP_all);
QDP_V_eq_zero(src, QDP_all);
QDP_r_eq_norm2_V(&y, dest, QDP_all);
```

x should equal y (within roundoff)

same but more efficient:

```
QDP_r_eq_norm2_V(&x, src, QDP_all);
QDP_V_eq_sV(dest, src, QDP_neighbor[0], QDP_backward, QDP_all);
QDP_r_eq_norm2_V(&y, dest, QDP_all);
QDP_discard_V(dest);
QDP_V_eq_zero(src, QDP_all);

...

QDP_r_eq_norm2_V(&z, dest, QDP_all);
error: attempt to use discarded data
```

Current Status of QDP/C

over 1200 math routines	done
custom subsets	done
custom shifts	done
lazy resolution of shifts	done
scalar communications	done
MPI communications	done
QMP communications	done (minor fixes)
number of dimensions != 4	not done yet
IO	in development
thoroughly tested	**NO**

- successfully passed several simple test codes
- replacement staggered inverter for MILC produces identical results
- ready for beta testing (hint, hint)

building

Currently the build process requires perl and GNU make.

Build Instructions:

```
1)  First build QLA and QMP (if used).


2)  Edit 'Makefile.inc' to use the appropriate libraries.


3)  Type 'make all' or 'make <pc>' where <pc> is one of
        f2, f3, fn, d2, d3, dn, df2, df3 or dfn.
```

Linking:

Depending on your choices in Makefile.inc you can end up with the following common libraries:

```
libqdp_common_milc.a
libqdp_common_mpi.a
libqdp_common_qmp.a
libqdp_common_vanilla.a
```

You should only link with one of these. Use either the mpi or qmp version to use MPI and QMP respectively. The vanilla version has no communication and should be used for single processor computing. The milc library can be used for linking with existing MILC code.

You will also need to link to (for example if using single precision 3 colors):

```
libqdp_int.a  libqdp_f.a  libqdp_f3.a
```

and the corresponding QLA libraries (for example):

```
libqla_random.a  libqla_int.a
libqla_f.a       libqla_f3.a
libqla_d.a       libqla_d3.a
libqla_df.a      libqla_df3.a
```

The higher precision libraries are needed for global sums.