Distributed QUEST

Generated by Doxygen 1.6.1

Mon Apr 3 12:28:04 2017

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Complex (Represents one complex number) 2

ComplexArray (Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components) 2

MultiQubit (Represents a system of qubits) 3

QUESTEnv (Information about the environment the program is running in) 5

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

basicTemplate.c (Basic template for using the QUEST library) 6

qubits.c (The core of the QUEST Library) 11

qubits.h (Structs and specifications for functions that can be used from any environment (local, MPI)) 26

qubits_env_local.c (An implementation of the API in qubits_env_wrapper.h for a local (non-MPI) environment) 41

qubits_env_mpi.c (An implementation of the API in qubits_env_wrapper.h for an MPI environment) 43

qubits_env_wrapper.h (Specifications for QUEST library functions whose implementation depends on environment (local, MPI)) 50

3 Data Structure Documentation

3.1 Complex Struct Reference

Represents one complex number.

```
#include <qubits.h>
```

Data Fields

- double real
- double imag

3.1.1 Detailed Description

Represents one complex number.

Definition at line 15 of file qubits.h.

3.1.2 Field Documentation

3.1.2.1 double Complex::imag

Definition at line 18 of file qubits.h.

Referenced by getRotAngle(), main(), rotateQubitDistributed(), and rotateQubitLocal().

3.1.2.2 double Complex::real

Definition at line 17 of file qubits.h.

Referenced by getRotAngle(), main(), rotateQubitDistributed(), and rotateQubitLocal().

The documentation for this struct was generated from the following file:

- qubits.h

3.2 ComplexArray Struct Reference

Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components.

```
#include <qubits.h>
```

Data Fields

- double ∗ real
- double ∗ imag

3.2.1 Detailed Description

Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components.

Definition at line 7 of file qubits.h.

3.2.2 Field Documentation

3.2.2.1 double∗ ComplexArray::imag

Definition at line 10 of file qubits.h.

Referenced by calcTotalProbability(), createMultiQubit(), destroyMultiQubit(), findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), initStateVec(), reportState(), rotateQubit(), rotateQubitDistributed(), and rotateQubitLocal().

3.2.2.2 double∗ ComplexArray::real

Definition at line 9 of file qubits.h.

Referenced by calcTotalProbability(), createMultiQubit(), destroyMultiQubit(), findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), initStateVec(), reportState(), rotateQubit(), rotateQubitDistributed(), and rotateQubitLocal().

The documentation for this struct was generated from the following file:

- qubits.h

3.3 MultiQubit Struct Reference

Represents a system of qubits.

```
#include <qubits.h>
```

Data Fields

- ComplexArray stateVec

  Probablilty amplitudes for the multi qubit state.

- ComplexArray pairStateVec

  Temporary storage for a chunk of the state vector received from another process in the MPI version.

- int numQubits

  Number of qubits in the state.

- long long int numAmps

  Number of probability amplitudes held in stateVec by this process In the non-MPI version, this is the total number of amplitudes.

- int chunkId

  The position of the chunk of the state vector held by this process in the full state vector.

- int numChunks

  Number of chunks the state vector is broken up into -- the number of MPI processes used.

3.3.1 Detailed Description

Represents a system of qubits. Qubits are zero-based and the the first qubit is the rightmost

Definition at line 24 of file qubits.h.

3.3.2 Field Documentation

3.3.2.1 int MultiQubit::chunkId

The position of the chunk of the state vector held by this process in the full state vector.

Definition at line 36 of file qubits.h.

Referenced by createMultiQubit(), findProbabilityOfZero(), initStateVec(), reportState(), and rotateQubit().

3.3.2.2 long long int MultiQubit::numAmps

Number of probability amplitudes held in stateVec by this process In the non-MPI version, this is the total number of amplitudes.

Definition at line 34 of file qubits.h.

Referenced by calcTotalProbability(), createMultiQubit(), findProbabilityOfZero(), findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), initStateVec(), reportState(), rotateQubit(), rotateQubitDistributed(), and rotateQubitLocal().

3.3.2.3 int MultiQubit::numChunks

Number of chunks the state vector is broken up into -- the number of MPI processes used.

Definition at line 38 of file qubits.h.

Referenced by calcTotalProbability(), and createMultiQubit().

3.3.2.4 int MultiQubit::numQubits

Number of qubits in the state.

Definition at line 31 of file qubits.h.

Referenced by createMultiQubit(), findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), rotateQubitDistributed(), and rotateQubitLocal().

3.3.2.5 ComplexArray MultiQubit::pairStateVec

Temporary storage for a chunk of the state vector received from another process in the MPI version.

Definition at line 29 of file qubits.h.

Referenced by createMultiQubit(), destroyMultiQubit(), and rotateQubit().

3.3.2.6 ComplexArray MultiQubit::stateVec

Probablilty amplitudes for the multi qubit state.

Definition at line 27 of file qubits.h.

Referenced by calcTotalProbability(), createMultiQubit(), destroyMultiQubit(), findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), initStateVec(), reportState(), rotateQubit(), and rotateQubitLocal().

The documentation for this struct was generated from the following file:

- qubits.h

3.4 QUESTEnv Struct Reference

Information about the environment the program is running in.

```
#include <qubits.h>
```

Data Fields

- int rank
- int numRanks

3.4.1 Detailed Description

Information about the environment the program is running in. In practice, this holds info about MPI ranks and helps to hide MPI initialization code

Definition at line 44 of file qubits.h.

3.4.2 Field Documentation

3.4.2.1 int QUESTEnv::numRanks

Definition at line 47 of file qubits.h.

Referenced by createMultiQubit(), destroyMultiQubit(), and initQUESTEnv().

3.4.2.2 int QUESTEnv::rank

Definition at line 46 of file qubits.h.

Referenced by createMultiQubit(), initQUESTEnv(), and main().

The documentation for this struct was generated from the following file:

- qubits.h

4 File Documentation

4.1 basicTemplate.c File Reference

Basic template for using the QUEST library. `#include <stdio.h>`

```
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <unistd.h>
#include <string.h>
#include <omp.h>
#include "QUEST/qubits.h"
#include "QUEST/qubits_env_wrapper.h"
```

Defines

- #define MaxAngles 10

  Max number of angles used to define qubit rotation.

- #define maxNumQubits 40

  Max number of qubits in the system.

- #define REPORT_STATE 1

  1: print end qubit state to file, 0: don't print

Functions

- int main (int narg, char **varg)

Variables

- const long double Pi = 3.14159265358979323846264338327950288419716939937510

4.1.1 Detailed Description

Basic template for using the QUEST library. In general, leave the initialisation and cleanup sections as they are and edit the rotations, measurement and phase gate sections.

Definition in file basicTemplate.c.

4.1.2 Define Documentation

4.1.2.1 #define MaxAngles 10

Max number of angles used to define qubit rotation.

Definition at line 19 of file basicTemplate.c.

Referenced by main().

4.1.2.2 #define maxNumQubits 40

Max number of qubits in the system.

Definition at line 21 of file basicTemplate.c.

Referenced by main().

4.1.2.3 #define REPORT_STATE 1

1: print end qubit state to file, 0: don't print

Definition at line 23 of file basicTemplate.c.

Referenced by main().

4.1.3 Function Documentation

4.1.3.1 int main (int *narg*, char ∗∗ *varg*)

Definition at line 31 of file basicTemplate.c.

References calcTotalProbability(), closeQUESTEnv(), createMultiQubit(), destroyMultiQubit(), Complex::imag, initQUESTEnv(), initStateVec(), MaxAngles, maxNumQubits, QUESTEnv::rank, Complex::real, REPORT_STATE, reportState(), and rotateQubit().

```
31                                      {
32
33           //
34           // ===== INITIALISATION
35           //
36
37           // INIT ENVIRONMENT: ALWAYS REQUIRED ONCE AT BEGINNING OF PROGRAM
```

```
        // These two lines will automatically set up the environment (multinode,
        // openMP only etc)
        QUESTEnv env;
        initQUESTEnv(&env);

        // model vars
        int numQubits;
        long int index;
        long int numAmps;

        // get number of qubits from command line argument
        if (narg >= 2) {
                numQubits = atoi(varg[1]);
                if (numQubits < 1 || numQubits > maxNumQubits) {
                        printf(" *** error: argument %d out of range (1 -- %d)\n", numQubits,maxNumQubits);
                        exit (EXIT_FAILURE);
                }
        } else {
                printf(" *** error: too few arguments, number of qubits expected\n");
                exit (EXIT_FAILURE);
        }

        // Reporting
        numAmps = 1L << numQubits;
        if (env.rank==0){
                printf("Demo of single qubit rotations.\n");
                printf("Number of qubits is %d.\n", numQubits);
                printf("Number of amps is %ld.\n", numAmps);
        }

        // CREATE QUBIT OBJECT: REQUIRED ONCE PER MULTIQUBIT OBJECT
        // Before doing any operations on a set of qubits, create the MultiQubit object that will be used to
        // represent the qubits
        MultiQubit multiQubit;
        createMultiQubit(&multiQubit, numQubits, env);

        // initialise the state to |0000..0>
        initStateVec (&multiQubit);


        //
        // ===== ROTATIONS
        //

        // INITIALISE QUBIT ROTATION
        // Edit these lines to change rotation angle
        double ang1,ang2,ang3;
        Complex alpha, beta;

        // define rotation angles
        double angles[MaxAngles][3] = {
                { 1.2320,  0.4230, -0.6523},
                { 2.1213,  0.0000,  3.6520},
                {-3.1213,  5.0230,  0.1230},
                { 5.2341, -3.1001, -1.2340},
                {-0.1234, -0.9876,  4.1234}
        };
        int numAngles=5,iAngle;

        // rotate
        ang1 = angles[0][0];
        ang2 = angles[0][1];
        ang3 = angles[0][2];

        alpha.real = cos(ang1) * cos(ang2);
        alpha.imag = cos(ang1) * sin(ang2);
        beta.real  = sin(ang1) * cos(ang3);
        beta.imag  = sin(ang1) * sin(ang3);

        int rotQubit;

        // DO QUBIT ROTATION
        // Edit these lines to perform rotations as required
        for (rotQubit=0; rotQubit<numQubits; rotQubit++) {
                // do rotation of each qubit
                rotateQubit(multiQubit,rotQubit,alpha,beta);
        }
        // END QUBIT ROTATION

        // Verification: check vector size is unchanged
        double totalProbability;
        totalProbability = calcTotalProbability(multiQubit);
        if (env.rank==0) printf("VERIFICATION: total probability=%.14f\n", totalProbability);

        // report state vector to file
        if (REPORT_STATE){
                reportState(multiQubit);
        }


        //
        // ===== perform a measurement
        //
        int measureQubit;
        double stateProb,randProb;
        /* // keep time */
        /* wtime_start = system_timer (); */

        // measure
/*
        for (measureQubit=0; measureQubit<numQubits; measureQubit++) {
        //for (measureQubit=0; measureQubit<1; measureQubit++) {
                syncQUESTEnv(env);
                wtime_start = system_timer ();
                stateProb = findProbabilityOfZero (env.rank, numAmpsPerRank, numQubits, measureQubit, stateVecReal,stateVecImag);
                syncQUESTEnv(env);
                wtime_stop = system_timer ();
                if (env.rank==0) printf("   probability of 0 for qubit %d = %.14f\n", measureQubit, stateProb);
                if (env.rank==0) printf(" measurement qubit %d: elapsed time = %f [s]\n", measureQubit, wtime_stop - wtime_start);
        }
*/
        /* // keep time */
        /* wtime_stop = system_timer (); */

        /* // ----- timing report */
        /* printf(" measurement: elapsed time = %f [s]\n", wtime_stop - wtime_start); */


        //
        // ===== two qubit phase gate
        //
        /* // keep time */
        /* wtime_start = system_timer (); */
/*
        // two qubit phase gate
        if (numQubits >= 7) {
                wtime_start = system_timer ();
                controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 0, 2, stateVecReal, stateVecImag);
                wtime_stop = system_timer ();
                printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
                wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 1, 3, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
                wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 2, 4, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
                wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 3, 5, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
                wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 4, 6, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
        }
*/
        totalProbability = calcTotalProbability(multiQubit);
        if (env.rank==0) printf("VERIFICATION: total probability=%.14f\n", totalProbability);
/*
if (env.rank==0){
        printf("\n\nIn rank %d, the following is the final state after rotations.\n\n",env.rank);
        printf("codeOutput=[\n");
        for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",stateVecReal[index],stateVecImag[index]);
        printf("];\n\n");
}
syncQUESTEnv(env);

if (env.rank==1){
        printf("\n\nIn rank %d, the following is the final state after rotations.\n\n",env.rank);
        printf("codeOutput=[\n");
        for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",stateVecReal[index],stateVecImag[index]);
        printf("];\n\n");
}
syncQUESTEnv(env);
*/
        /* // keep time */
        /* wtime_stop = system_timer (); */

        /* // ----- timing report */
        /* printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start); */


        //
        // ======== CLEANUP
        //

        // free memory

        // REQUIRED ONCE PER MULTIQUBIT OBJECT
        // When all operations on a set of qubits are completed, destroy the object
        destroyMultiQubit(multiQubit, env);


        // ALWAYS REQUIRED ONCE AT END OF PROGRAM:
        // These two lines will perform any necessary cleanup of the environment (multinode,
        // openMP only etc)
        closeQUESTEnv(env);

        return EXIT_SUCCESS;
}
```

4.1.4 Variable Documentation

4.1.4.1 const long double Pi = 3.14159265358979323846264338327950288419716939937510

Definition at line 25 of file basicTemplate.c.

4.2 qubits.c File Reference

The core of the QUEST Library. `#include "math.h"`

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <assert.h>`

`#include "qubits.h"`

Defines

- #define DEBUG 0

Functions

- void createMultiQubit (MultiQubit ∗multiQubit, int numQubits, QUESTEnv env)
- void destroyMultiQubit (MultiQubit multiQubit, QUESTEnv env)
- void reportState (MultiQubit multiQubit)
- void initStateVec (MultiQubit ∗multiQubit)

  Initialise the state vector of probability amplitudes for a set of qubits to the zero state: |000...00>.

- void rotateQubitLocal (MultiQubit multiQubit, const int rotQubit, Complex alpha, Complex beta)

  Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments.

- void rotateQubitDistributed (MultiQubit multiQubit, const int rotQubit, Complex rot1, Complex rot2, ComplexArray stateVecUp, ComplexArray stateVecLo, ComplexArray stateVecOut)

  Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored seperately.

- double findProbabilityOfZeroLocal (MultiQubit multiQubit, const int measureQubit)

  Measure the probability of a specified qubit being in the zero state.

- double findProbabilityOfZeroDistributed (MultiQubit multiQubit, const int measureQubit)

Measure the probability of a specified qubit being in the zero state.

- int extractBit (const int locationOfBitFromRight, const long long int theEncodedNumber)
- void controlPhaseGate (const int numQubits, const int idQubit1, const int idQubit2, double ∗restrict stateVecReal, double ∗restrict stateVecImag)

  Implement the control phase (the two qubit phase gate).

- void quadCPhaseGate (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, const int idQubit4, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double measureInZero (const int numQubits, const int measureQubit, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double filterOut111 (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double probOfFilterOut111 (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, double ∗restrict stateVecReal, double ∗restrict stateVecImag)

4.2.1 Detailed Description

The core of the QUEST Library.

Definition in file qubits.c.

4.2.2 Define Documentation

4.2.2.1 #define DEBUG 0

Definition at line 11 of file qubits.c.

Referenced by calcTotalProbability(), initQUESTEnv(), and initStateVec().

4.2.3 Function Documentation

4.2.3.1 void controlPhaseGate (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Implement the control phase (the two qubit phase gate). REWRITE TO USE MULTIQUBIT input: // numQubits -- number of qubits // idQubit1, -- specified qubits // idQubit2 // stateVecReal, -- real/imag parts of // stateVecImag the state vector // // output: // stateVecReal, -- real/imag parts of // stateVecImag the state vector (overwritten) // //

Definition at line 454 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2;

        // ---------------------------------------------------------------- //
        //            tests                                                 //
463         // ---------------------------------------------------------------- //
464
465         assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
    2 < numQubits);
466
467
468         // ---------------------------------------------------------------- //
469         //           initialise the state to |0000..0>                    //
470         // ---------------------------------------------------------------- //
471
472         // dimension of the state vector
473         stateVecSize = 1LL << numQubits;
474
475 # ifdef _OPENMP
476 # pragma omp parallel for \
477         default  (none)                      \
478         shared   (stateVecSize, stateVecReal,stateVecImag ) \
479         private  (index,bit1,bit2)                     \
480         schedule (static)
481 # endif
482         for (index=0; index<stateVecSize; index++) {
483                 bit1 = extractBit (idQubit1, index);
484                 bit2 = extractBit (idQubit2, index);
485                 if (bit1 && bit2) {
486                         stateVecReal [index] = - stateVecReal [index];
487                         stateVecImag [index] = - stateVecImag [index];
488                 }
489         }
490 }
```

4.2.3.2 void createMultiQubit (MultiQubit ∗ *multiQubit*, int *numQubits*, QUESTEnv *env*)

Definition at line 16 of file qubits.c.

References MultiQubit::chunkId, ComplexArray::imag, initStateVec(), MultiQubit::numAmps, MultiQubit::numChunks, MultiQubit::numQubits, QUESTEnv::numRanks, MultiQubit::pairStateVec, QUESTEnv::rank, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
17 {
18         long long int numAmps = 1L << numQubits;
19         long long int numAmpsPerRank = numAmps/env.numRanks;
20
21         multiQubit->stateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->
    stateVec.real));
22         multiQubit->stateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->
    stateVec.imag));
23         if (env.numRanks>1){
24                 multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(mu
    ltiQubit->pairStateVec.real));
25                 multiQubit->pairStateVec.imag = malloc(numAmpsPerRank * sizeof(mu
    ltiQubit->pairStateVec.imag));
26         }
27
28         if ( (!(multiQubit->stateVec.real) || !(multiQubit->stateVec.imag))
29                  && numAmpsPerRank ) {
30                 printf("Could not allocate memory!");
31                 exit (EXIT_FAILURE);
32         }
33
34         if ( env.numRanks>1 && (!(multiQubit->pairStateVec.real) || !(multiQubit-
      >pairStateVec.imag))
35                 && numAmpsPerRank ) {
36                printf("Could not allocate memory!");
37                exit (EXIT_FAILURE);
38        }
39
40        multiQubit->numQubits = numQubits;
41        multiQubit->numAmps = numAmpsPerRank;
42        multiQubit->chunkId = env.rank;
43        multiQubit->numChunks = env.numRanks;
44
45        initStateVec(multiQubit);
46        if (env.rank==0) printf("Number of amps per rank is %ld.\n", numAmpsPerRa
      nk);
47 }
```

4.2.3.3 void destroyMultiQubit (MultiQubit *multiQubit*, QUESTEnv *env*)

Definition at line 49 of file qubits.c.

References ComplexArray::imag, QUESTEnv::numRanks, MultiQubit::pairStateVec, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
49                                                              {
50        free(multiQubit.stateVec.real);
51        free(multiQubit.stateVec.imag);
52        if (env.numRanks>1){
53                free(multiQubit.pairStateVec.real);
54                free(multiQubit.pairStateVec.imag);
55        }
56 }
```

4.2.3.4 int extractBit (const int *locationOfBitFromRight*, const long long int *theEncodedNumber*)

Definition at line 433 of file qubits.c.

Referenced by controlPhaseGate(), filterOut111(), probOfFilterOut111(), and quadCPhaseGate().

```
434 {
435         return (theEncodedNumber & ( 1LL << locationOfBitFromRight )) >> location
      OfBitFromRight;
436 }
```

4.2.3.5 double filterOut111 (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 623 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2, bit3;

        // ---------------------------------------------------------------- //
        //              tests                                               //
        // ---------------------------------------------------------------- //
        assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit2 < numQubits);

        stateVecSize = 1LL << numQubits;
        double probOfFilter=0;

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                  \
        shared   (stateVecSize, stateVecReal,stateVecImag ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)\
        reduction ( +:probOfFilter )
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if (!(bit1 && bit2 && bit3)) {
                        probOfFilter+= stateVecReal[index]*stateVecReal[index] + stateVecImag[index]* stateVecImag [index];
                }
        }
        if ( probOfFilter<1e-16 ){ printf("Extremely small or negative profOfFilter=%.8e; aborting! \n",probOfFilter); exit(1);}
        double myNorm=1/sqrt(probOfFilter);

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                  \
        shared   (stateVecSize, stateVecReal,stateVecImag, myNorm ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if ((bit1 && bit2 && bit3)) {
                        stateVecReal[index]=0;
                        stateVecImag [index]=0;
                }else{
                        stateVecReal[index] *= myNorm;
                        stateVecImag[index] *= myNorm;
                }
        }
        return probOfFilter;
}
```

4.2.3.6 double findProbabilityOfZeroDistributed (MultiQubit *multiQubit*, const int *measureQubit*)

Measure the probability of a specified qubit being in the zero state. Size of regions to skip is a multiple of chunkSize.

Parameters:

← ***multiQubit*** object representing the set of qubits to be initialised

← ***measureQubit*** qubit to measure

Returns:

probability of qubit measureQubit being zero

Definition at line 375 of file qubits.c.

References ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```
{
        // ----- measured probability
        double   totalProbability;                                  // probability (returned) value
        // ----- temp variables
        long long int thisTask;                                 // task based approach for expose loop with small granularity
        long long int numTasks=multiQubit.numAmps;
        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                  //
        // ---------------------------------------------------------------- //
        assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);

        // ---------------------------------------------------------------- //
        //            find probability                                       //
        // ---------------------------------------------------------------- //

        // initialise returned value
        totalProbability = 0.0;

        // initialise correction for kahan summation

        //
        // --- task-based shared-memory parallel implementation
        //

        double *stateVecReal = multiQubit.stateVec.real;
        double *stateVecImag = multiQubit.stateVec.imag;

# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,stateVecReal,stateVecImag) \
        private   (thisTask) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                // summation -- simple implementation
                totalProbability += stateVecReal[thisTask]*stateVecReal[thisTask]
                        + stateVecImag[thisTask]*stateVecImag[thisTask];

                /*
                // summation -- kahan correction
                y = stateVecReal[thisTask]*stateVecReal[thisTask]
                + stateVecImag[thisTask]*stateVecImag[thisTask] - c;
                t = totalProbability + y;
                c = (t - totalProbability) - y;
                totalProbability = t;
                */

        }

        return totalProbability;
}
```

4.2.3.7 double findProbabilityOfZeroLocal (MultiQubit *multiQubit*, const int *measureQubit*)

Measure the probability of a specified qubit being in the zero state. Size of regions to skip is less than the size of one chunk.

Parameters:

← ***multiQubit*** object representing the set of qubits to be initialised

← ***measureQubit*** qubit to measure

Returns:

probability of qubit measureQubit being zero

Definition at line 289 of file qubits.c.

References ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```
{
        // ----- sizes
        long long int sizeBlock,                                           // siz
  e of blocks
        sizeHalfBlock;                                       // size of blocks ha
  lved
        // ----- indices
        long long int thisBlock,                                           // cur
  rent block
             index;                                             // current inde
  x for first half block
        // ----- measured probability
        double   totalProbability;                                     // probabil
  ity (returned) value
        // ----- temp variables
        long long int thisTask;                                 // task based a
  pproach for expose loop with small granularity
        long long int numTasks=multiQubit.numAmps>>1;
        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //           tests                                                  //
        // ---------------------------------------------------------------- //
        assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);


        // ---------------------------------------------------------------- //
        //           dimensions                                             //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << (measureQubit);                       // number of
   state vector elements to sum,
        // and then the number to skip
        sizeBlock     = 2LL * sizeHalfBlock;                           // size of blocks (pairs of measure and skip entries)

        // ---------------------------------------------------------------- //
        //            find probability                                      //
        // ---------------------------------------------------------------- //

        // initialise returned value
        totalProbability = 0.0;

        // initialise correction for kahan summation
        printf("sizeHalfBlock=%Ld sizeBlock=%Ld numTasks=%Ld\n",sizeHalfBlock,sizeBlock,numTasks);

        //
        // --- task-based shared-memory parallel implementation
        //

        double *stateVecReal = multiQubit.stateVec.real;
        double *stateVecImag = multiQubit.stateVec.imag;

# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;

                if (index<0){ printf("ABORTING as index=%Ld with thisBlock = %Ld thisTask=%Ld \n", index,thisBlock,thisTask); exit(1);}

                // summation -- simple implementation
                totalProbability += stateVecReal[index]*stateVecReal[index]
                        + stateVecImag[index]*stateVecImag[index];

                /*
                // summation -- kahan correction
                y = stateVecReal[index]*stateVecReal[index]
                + stateVecImag[index]*stateVecImag[index] - c;
                t = totalProbability + y;
                c = (t - totalProbability) - y;
                totalProbability = t;
                */

        }

        return totalProbability;
}
```

4.2.3.8 void initStateVec (MultiQubit * *multiQubit*)

Initialise the state vector of probability amplitudes for a set of qubits to the zero state: |000...00>.

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

Definition at line 76 of file qubits.c.

References MultiQubit::chunkId, DEBUG, ComplexArray::imag, MultiQubit::numAmps, ComplexArray::real, and MultiQubit::stateVec.

Referenced by createMultiQubit(), and main().

```
{
        long long int stateVecSize;
        long long int index;

        // dimension of the state vector
        stateVecSize = multiQubit->numAmps;

        if (DEBUG) printf("stateVecSize=%Ld   now performing init with only one t
   hread:\n",stateVecSize);

        // Can't use multiQubit->stateVec as a private OMP var
        double *stateVecReal = multiQubit->stateVec.real;
        double *stateVecImag = multiQubit->stateVec.imag;

        // initialise the state to |0000..0000>
# ifdef _OPENMP
# pragma omp parallel for \
        default  (none) \
        shared   (stateVecSize, stateVecReal, stateVecImag) \
        private  (index) \
        schedule (static)
# endif
        for (index=0; index<stateVecSize; index++) {
                stateVecReal[index] = 0.0;
                stateVecImag[index] = 0.0;
        }

        if (multiQubit->chunkId==0){
                // zero state |0000..0000> has probability 1
                stateVecReal[0] = 1.0;
                stateVecImag[0] = 0.0;
        }

        if (DEBUG) printf("COMPLETED INIT\n");
}
```

4.2.3.9 double measureInZero (const int *numQubits*, const int *measureQubit*, double *restrict *stateVecReal*, double *restrict *stateVecImag*)

Definition at line 534 of file qubits.c.

```
{
        // ----- sizes
        long long int numBlocks,                                           // num
   ber of blocks
        sizeBlock,                                            // size of blocks
        sizeHalfBlock;                                        // size of blocks ha
   lved
        // ----- indices
        long long int thisBlock,                                           // cur
   rent block
             index;                                                // current inde
   x for first half block
        // ----- measured probability
        double   totalProbability, renorm;                                    //
```

```
  probability (returned) value
        // ----- temp variables
        long long int thisTask,numTasks;                                   // task based approach for expose loop with small granularity
        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (measureQubit >= 0 && measureQubit < numQubits);


        // ---------------------------------------------------------------- //
        //            dimensions                                            //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << (measureQubit);                       // number of state vector elements to sum,
        // and then the number to skip
        sizeBlock     = 2LL * sizeHalfBlock;                           // size of blocks (pairs of measure and skip entries)

        // ---------------------------------------------------------------- //
        //            find probability                                      //
        // ---------------------------------------------------------------- //
        numTasks = 1LL << (numQubits-1);

        // initialise returned value
        totalProbability = 0.0;

        //
        // --- task-based shared-memory parallel implementation
        //
# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;

                totalProbability += stateVecReal[index]*stateVecReal[index]
                        + stateVecImag[index]*stateVecImag[index];
        }
        renorm=1/sqrt(totalProbability);


# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
                stateVecReal[index]=stateVecReal[index]*renorm;
                stateVecImag[index]=stateVecImag[index]*renorm;

                stateVecReal[index+sizeHalfBlock]=0;
                stateVecImag[index+sizeHalfBlock]=0;
        }

        //SCB this is a debugging style check. It is probably useful to leave in, but it could be parallelised I guess
        //  double checkTotal=1.;
        //  for (index=0; index<2*numTasks; index++) {
        //      checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] + stateVecImag[index]*stateVecImag[index]);
        //  }
        //  if (checkTotal>0.00001){printf("Deviation of sum squared amps from unity is %.16f\n",checkTotal); exit(1);}

        return totalProbability;
}
```

4.2.3.10 double probOfFilterOut111 (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 683 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2, bit3;

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit2 < numQubits);

        stateVecSize = 1LL << numQubits;
        double probOfFilter=0;

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                      \
        shared   (stateVecSize, stateVecReal,stateVecImag ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)\
        reduction ( +:probOfFilter )
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if (!(bit1 && bit2 && bit3)) {
                        probOfFilter+= stateVecReal[index]*stateVecReal[index] + stateVecImag[index]* stateVecImag [index];
                }
        }
        return probOfFilter;
}
```

4.2.3.11 void quadCPhaseGate (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, const int *idQubit4*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 496 of file qubits.c.

References extractBit().

```
497 {
498         long long int index;
499         long long int stateVecSize;
500         int bit1, bit2, bit3, bit4;
501 
502         // ---------------------------------------------------------------- //
503         //            tests                                                 //
504         // ---------------------------------------------------------------- //
505         assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
      2 < numQubits);
506 
507         stateVecSize = 1LL << numQubits;
508 
509 # ifdef _OPENMP
510 # pragma omp parallel for \
511         default  (none)                      \
512         shared   (stateVecSize, stateVecReal,stateVecImag ) \
513         private  (index,bit1,bit2,bit3,bit4)                  \
514         schedule (static)
515 # endif
516         for (index=0; index<stateVecSize; index++) {
517                 bit1 = extractBit (idQubit1, index);
518                 bit2 = extractBit (idQubit2, index);
519                 bit3 = extractBit (idQubit3, index);
520                 bit4 = extractBit (idQubit4, index);
521                 if (bit1 && bit2 && bit3 && bit4) {
522                         stateVecReal [index] = - stateVecReal [index];
523                         stateVecImag [index] = - stateVecImag [index];
524                 }
525         }
526 }
```

4.2.3.12 void reportState (MultiQubit *multiQubit*)

Definition at line 58 of file qubits.c.

References MultiQubit::chunkId, ComplexArray::imag, MultiQubit::numAmps, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
58                                       {
59          FILE *state;
60          char filename[100];
61          long long int index;
62          sprintf(filename, "state_rank_%d.csv", multiQubit.chunkId);
63          state = fopen(filename, "w");
64          if (multiQubit.chunkId==0) fprintf(state, "real, imag\n");
65 
66          for(index=0; index<multiQubit.numAmps; index++){
67                  fprintf(state, "%.12f, %.12f\n", multiQubit.stateVec.real[index],
       multiQubit.stateVec.imag[index]);
68          }
69          fclose(state);
70 }
```

4.2.3.13 void rotateQubitDistributed (MultiQubit *multiQubit*, const int *rotQubit*, Complex *rot1*, Complex *rot2*, ComplexArray *stateVecUp*, ComplexArray *stateVecLo*, ComplexArray *stateVecOut*)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored seperately.

Remarks:

Qubits are zero-based and the the first qubit is the rightmost

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

← ***rotQubit*** qubit to rotate

← ***rot1*** rotation angle

← ***rot2*** rotation angle

← ***stateVecUp*** probability amplitudes in upper half of a block

← ***stateVecLo*** probability amplitudes in lower half of a block

→ ***stateVecOut*** array section to update (will correspond to either the lower or upper half of a block)

Definition at line 224 of file qubits.c.

References ComplexArray::imag, Complex::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and Complex::real.

Referenced by rotateQubit().

```
{
        // ----- temp variables
        double   stateRealUp,stateRealLo,                             // storage for previous state values
        stateImagUp,stateImagLo;                                  // (used in updates)

        // ----- temp variables
        long long int thisTask;                                      // task based approach for expose loop with small granularity
        const long long int numTasks=multiQubit.numAmps;

        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);

        // ---------------------------------------------------------------- //
        //            rotate                                                //
        // ---------------------------------------------------------------- //

        //
        // --- task-based shared-memory parallel implementation
        //
        double rot1Real=rot1.real, rot1Imag=rot1.imag;
        double rot2Real=rot2.real, rot2Imag=rot2.imag;
        double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;
        double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
        double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.
```

```
  imag;
# pragma omp parallel \
        default  (none) \
        shared   (stateVecRealUp,stateVecImagUp,stateVecRealLo,stateVecImagLo,stateVecRealOut,stateVecImagOut, \
                        rot1Real,rot1Imag, rot2Real,rot2Imag) \
        private  (thisTask,stateRealUp,stateImagUp,stateRealLo,stateImagLo)
        {
# pragma omp for \
                schedule (static)
                for (thisTask=0; thisTask<numTasks; thisTask++) {
                        // store current state vector values in temp variables
                        stateRealUp = stateVecRealUp[thisTask];
                        stateImagUp = stateVecImagUp[thisTask];

                        stateRealLo = stateVecRealLo[thisTask];
                        stateImagLo = stateVecImagLo[thisTask];

                        // state[indexUp] = alpha * state[indexUp] - conj(beta) * state[indexLo]
                        stateVecRealOut[thisTask] = rot1Real*stateRealUp - rot1Imag*stateImagUp + rot2Real*stateRealLo + rot2Imag*stateImagLo;
                        stateVecImagOut[thisTask] = rot1Real*stateImagUp + rot1Imag*stateRealUp + rot2Real*stateImagLo - rot2Imag*stateRealLo;
                } // end for loop
        }
} // end of function definition
```

4.2.3.14 void rotateQubitLocal (MultiQubit *multiQubit*, const int *rotQubit*, Complex *alpha*, Complex *beta*)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments. alphaRe = cos(angle1) ∗ cos(angle2)

alphaIm = cos(angle1) ∗ sin(angle2)

betaRe = sin(angle1) ∗ cos(angle3)

betaIm = sin(angle1) ∗ sin(angle3)

Remarks:

Qubits are zero-based and the the first qubit is the rightmost

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

← ***rotQubit*** qubit to rotate

← ***alpha*** rotation angle

← ***beta*** rotation angle

Definition at line 126 of file qubits.c.

References Complex::imag, ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, Complex::real, ComplexArray::real, and MultiQubit::stateVec.

Referenced by rotateQubit().

```
{
        // ----- sizes
        long long int sizeBlock,                                           // size of blocks
        sizeHalfBlock;                                             // size of blocks halved
        // ----- indices
        long long int thisBlock,                                           // current block
             indexUp,indexLo;                                           // current index and corresponding index in lower half block

        // ----- temp variables
        double   stateRealUp,stateRealLo,                              // storage for previous state values
                 stateImagUp,stateImagLo;                              // (used in updates)
        // ----- temp variables
        long long int thisTask;                                    // task based approach for expose loop with small granularity
        const long long int numTasks=multiQubit.numAmps>>1;
        // (good for shared memory parallelism)


        // ---------------------------------------------------------------- //
        //            tests                                                  //
        // ---------------------------------------------------------------- //
        assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);


        // ---------------------------------------------------------------- //
        //            dimensions                                             //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << rotQubit;                              // size of blocks halved
        sizeBlock     = 2LL * sizeHalfBlock;                          // size of blocks


        // ---------------------------------------------------------------- //
        //            rotate                                                 //
        // ---------------------------------------------------------------- //

        //
        // --- task-based shared-memory parallel implementation
        //

        // Can't use multiQubit.stateVec as a private OMP var
        double *stateVecReal = multiQubit.stateVec.real;
        double *stateVecImag = multiQubit.stateVec.imag;
        double alphaImag=alpha.imag, alphaReal=alpha.real;
        double betaImag=beta.imag, betaReal=beta.real;

# ifdef _OPENMP
# pragma omp parallel \
        default  (none) \
        shared   (sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag, alphaReal,alphaImag, betaReal,betaImag) \
        private  (thisTask,thisBlock ,indexUp,indexLo, stateRealUp,stateImagUp,stateRealLo,stateImagLo)
# endif
        {
# ifdef _OPENMP
# pragma omp for \
                schedule (static)
# endif
                for (thisTask=0; thisTask<numTasks; thisTask++) {

                        thisBlock   = thisTask / sizeHalfBlock;
                        indexUp     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
                        indexLo     = indexUp + sizeHalfBlock;

                        // store current state vector values in temp variables
                        stateRealUp = stateVecReal[indexUp];
                        stateImagUp = stateVecImag[indexUp];

                        stateRealLo = stateVecReal[indexLo];
                        stateImagLo = stateVecImag[indexLo];

                        // state[indexUp] = alpha * state[indexUp] - conj(beta) * state[indexLo]
                        stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag*stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
                        stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag*stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;

                        // state[indexLo] = beta  * state[indexUp] + conj(alpha) * state[indexLo]
                        stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*stateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
                        stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*stateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
                } // end for loop
        }

} // end of function definition
```

4.3 qubits.h File Reference

Structs and specifications for functions that can be used from any environment (local, MPI).

Data Structures

- struct ComplexArray

    Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components.

- struct Complex

    Represents one complex number.

- struct MultiQubit

    Represents a system of qubits.

- struct QUESTEnv

    Information about the environment the program is running in.

Functions

- void createMultiQubit (MultiQubit *multiQubit, int numQubits, QUESTEnv env)
- void destroyMultiQubit (MultiQubit multiQubit, QUESTEnv env)
- void reportState (MultiQubit multiQubit)
- void initStateVec (MultiQubit *multiQubit)

Initialise the state vector of probability amplitudes for a set of qubits to the zero state: |000...00>.

- void rotateQubitLocal (MultiQubit multiQubit, const int rotQubit, Complex alpha, Complex beta)

  Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments.

- void rotateQubitDistributed (MultiQubit multiQubit, const int rotQubit, Complex rot1, Complex rot2, ComplexArray stateVecUp, ComplexArray stateVecLo, ComplexArray stateVecOut)

  Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored seperately.

- double findProbabilityOfZeroLocal (MultiQubit multiQubit, const int measureQubit)

  Measure the probability of a specified qubit being in the zero state.

- double findProbabilityOfZeroDistributed (MultiQubit multiQubit, const int measureQubit)

  Measure the probability of a specified qubit being in the zero state.

- int extractBit (const int locationOfBitFromRight, const long long int theEncodedNumber)
- void controlPhaseGate (const int numQubits, const int idQubit1, const int idQubit2, double ∗restrict stateVecReal, double ∗restrict stateVecImag)

  Implement the control phase (the two qubit phase gate).

- void quadCPhaseGate (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, const int idQubit4, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double measureInZero (const int numQubits, const int measureQubit, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double filterOut111 (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, double ∗restrict stateVecReal, double ∗restrict stateVecImag)
- double probOfFilterOut111 (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, double ∗restrict stateVecReal, double ∗restrict stateVecImag)

4.3.1 Detailed Description

Structs and specifications for functions that can be used from any environment (local, MPI).

Definition in file qubits.h.

4.3.2 Function Documentation

4.3.2.1 void controlPhaseGate (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Implement the control phase (the two qubit phase gate). REWRITE TO USE MULTIQUBIT input: // numQubits -- number of qubits // idQubit1, -- specified qubits // idQubit2 // stateVecReal, -- real/imag parts of // stateVecImag the state vector // // output: // stateVecReal, -- real/imag parts of // stateVecImag the state vector (overwritten) // //

Definition at line 454 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2;

        // ----------------------------------------------------------------- //
        //            tests                                                  //
        // ----------------------------------------------------------------- //

        assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit2 < numQubits);


        // ----------------------------------------------------------------- //
        //            initialise the state to |0000..0>                      //
        // ----------------------------------------------------------------- //

        // dimension of the state vector
        stateVecSize = 1LL << numQubits;

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                       \
        shared   (stateVecSize, stateVecReal,stateVecImag ) \
        private  (index,bit1,bit2)                      \
        schedule (static)
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                if (bit1 && bit2) {
                        stateVecReal [index] = - stateVecReal [index];
                        stateVecImag [index] = - stateVecImag [index];
                }
        }
}
```

4.3.2.2 void createMultiQubit (MultiQubit ∗ *multiQubit*, int *numQubits*, QUESTEnv *env*)

Definition at line 16 of file qubits.c.

References MultiQubit::chunkId, ComplexArray::imag, initStateVec(), MultiQubit::numAmps, MultiQubit::numChunks, MultiQubit::numQubits, QUESTEnv::numRanks, MultiQubit::pairStateVec, QUESTEnv::rank, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
{
        long long int numAmps = 1L << numQubits;
        long long int numAmpsPerRank = numAmps/env.numRanks;

        multiQubit->stateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->stateVec.real));
        multiQubit->stateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->stateVec.imag));
        if (env.numRanks>1){
                multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->pairStateVec.real));
                multiQubit->pairStateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->pairStateVec.imag));
        }

28          if ( (!(multiQubit->stateVec.real) || !(multiQubit->stateVec.imag))
29                   && numAmpsPerRank ) {
30                  printf("Could not allocate memory!");
31                  exit (EXIT_FAILURE);
32          }
33
34          if ( env.numRanks>1 && (!(multiQubit->pairStateVec.real) || !(multiQubit-
      >pairStateVec.imag))
35                   && numAmpsPerRank ) {
36                  printf("Could not allocate memory!");
37                  exit (EXIT_FAILURE);
38          }
39
40          multiQubit->numQubits = numQubits;
41          multiQubit->numAmps = numAmpsPerRank;
42          multiQubit->chunkId = env.rank;
43          multiQubit->numChunks = env.numRanks;
44
45          initStateVec(multiQubit);
46          if (env.rank==0) printf("Number of amps per rank is %ld.\n", numAmpsPerRa
      nk);
47 }
```

4.3.2.3 void destroyMultiQubit (MultiQubit *multiQubit*, QUESTEnv *env*)

Definition at line 49 of file qubits.c.

References ComplexArray::imag, QUESTEnv::numRanks, MultiQubit::pairStateVec, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
49                                                          {
50          free(multiQubit.stateVec.real);
51          free(multiQubit.stateVec.imag);
52          if (env.numRanks>1){
53                  free(multiQubit.pairStateVec.real);
54                  free(multiQubit.pairStateVec.imag);
55          }
56 }
```

4.3.2.4 int extractBit (const int *locationOfBitFromRight*, const long long int *theEncodedNumber*)

Definition at line 433 of file qubits.c.

Referenced by controlPhaseGate(), filterOut111(), probOfFilterOut111(), and quadCPhaseGate().

```
434 {
435          return (theEncodedNumber & ( 1LL << locationOfBitFromRight )) >> location
      OfBitFromRight;
436 }
```

4.3.2.5 double filterOut111 (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, double *restrict *stateVecReal*, double *restrict *stateVecImag*)

Definition at line 623 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2, bit3;

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit2 < numQubits);

        stateVecSize = 1LL << numQubits;
        double probOfFilter=0;

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                      \
        shared   (stateVecSize, stateVecReal,stateVecImag ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)\
        reduction ( +:probOfFilter )
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if (!(bit1 && bit2 && bit3)) {
                        probOfFilter+= stateVecReal[index]*stateVecReal[index] + stateVecImag[index]* stateVecImag [index];
                }
        }
        if ( probOfFilter<1e-16 ){ printf("Extremely small or negative profOfFilter=%.8e; aborting! \n",probOfFilter); exit(1);}
        double myNorm=1/sqrt(probOfFilter);

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                      \
        shared   (stateVecSize, stateVecReal,stateVecImag, myNorm ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if ((bit1 && bit2 && bit3)) {
                        stateVecReal[index]=0;
                        stateVecImag [index]=0;
                }else{
                        stateVecReal[index] *= myNorm;
                        stateVecImag[index] *= myNorm;
                }
        }
        return probOfFilter;
}
```

4.3.2.6 double findProbabilityOfZeroDistributed (MultiQubit *multiQubit*, const int *measureQubit*)

Measure the probability of a specified qubit being in the zero state. Size of regions to skip is a multiple of chunkSize.

Parameters:

← ***multiQubit*** object representing the set of qubits to be initialised

← ***measureQubit*** qubit to measure

Returns:

probability of qubit measureQubit being zero

Definition at line 375 of file qubits.c.

References ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```
377 {
378         // ----- measured probability
379         double   totalProbability;                                  // probabil
      ity (returned) value
380         // ----- temp variables
381         long long int thisTask;                                 // task based a
      pproach for expose loop with small granularity
382         long long int numTasks=multiQubit.numAmps;
383         // (good for shared memory parallelism)
384 
385         // ---------------------------------------------------------------- //
386         //           tests                                                  //
387         // ---------------------------------------------------------------- //
388         assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);
389 
390         // ---------------------------------------------------------------- //
391         //           find probability                                       //
392         // ---------------------------------------------------------------- //
393 
394         // initialise returned value
395         totalProbability = 0.0;
396 
397         // initialise correction for kahan summation
398 
399         //
400         // --- task-based shared-memory parallel implementation
401         //
402 
403         double *stateVecReal = multiQubit.stateVec.real;
404         double *stateVecImag = multiQubit.stateVec.imag;
405 
406 # ifdef _OPENMP
407 # pragma omp parallel for \
408         shared    (numTasks,stateVecReal,stateVecImag) \
409         private   (thisTask) \
410         schedule  (static) \
411         reduction ( +:totalProbability )
412 # endif
413         for (thisTask=0; thisTask<numTasks; thisTask++) {
414                 // summation -- simple implementation
415                 totalProbability += stateVecReal[thisTask]*stateVecReal[thisTask]

416                         + stateVecImag[thisTask]*stateVecImag[thisTask];
417 
418                 /*
419                 // summation -- kahan correction
```

```
                y = stateVecReal[thisTask]*stateVecReal[thisTask]
                + stateVecImag[thisTask]*stateVecImag[thisTask] - c;
                t = totalProbability + y;
                c = (t - totalProbability) - y;
                totalProbability = t;
                */

        }

        return totalProbability;
}
```

4.3.2.7 double findProbabilityOfZeroLocal (MultiQubit *multiQubit*, const int *measureQubit*)

Measure the probability of a specified qubit being in the zero state. Size of regions to skip is less than the size of one chunk.

Parameters:

← ***multiQubit*** object representing the set of qubits to be initialised

← ***measureQubit*** qubit to measure

Returns:

probability of qubit measureQubit being zero

Definition at line 289 of file qubits.c.

References ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```
{
        // ----- sizes
        long long int sizeBlock,                                           // size of blocks
        sizeHalfBlock;                                       // size of blocks halved
        // ----- indices
        long long int thisBlock,                                           // current block
             index;                                                // current index for first half block
        // ----- measured probability
        double   totalProbability;                                     // probability (returned) value
        // ----- temp variables
        long long int thisTask;                                    // task based approach for expose loop with small granularity
        long long int numTasks=multiQubit.numAmps>>1;
        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                  //
        // ---------------------------------------------------------------- //
        assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);


        // ---------------------------------------------------------------- //
        //            dimensions                                          //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << (measureQubit);                       // number of state vector elements to sum,
        // and then the number to skip
        sizeBlock     = 2LL * sizeHalfBlock;                           // size of blocks (pairs of measure and skip entries)

        // ---------------------------------------------------------------- //
        //            find probability                                    //
        // ---------------------------------------------------------------- //

        // initialise returned value
        totalProbability = 0.0;

        // initialise correction for kahan summation
        printf("sizeHalfBlock=%Ld sizeBlock=%Ld numTasks=%Ld\n",sizeHalfBlock,sizeBlock,numTasks);

        //
        // --- task-based shared-memory parallel implementation
        //

        double *stateVecReal = multiQubit.stateVec.real;
        double *stateVecImag = multiQubit.stateVec.imag;

# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \

        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;

                if (index<0){ printf("ABORTING as index=%Ld with thisBlock = %Ld thisTask=%Ld \n", index,thisBlock,thisTask); exit(1);}

                // summation -- simple implementation
                totalProbability += stateVecReal[index]*stateVecReal[index]
                        + stateVecImag[index]*stateVecImag[index];

                /*
                // summation -- kahan correction
                y = stateVecReal[index]*stateVecReal[index]
                + stateVecImag[index]*stateVecImag[index] - c;
                t = totalProbability + y;
                c = (t - totalProbability) - y;
                totalProbability = t;
                */

        }

        return totalProbability;
}
```

4.3.2.8 void initStateVec (MultiQubit * *multiQubit*)

Initialise the state vector of probability amplitudes for a set of qubits to the zero state: |000...00>.

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

Definition at line 76 of file qubits.c.

References MultiQubit::chunkId, DEBUG, ComplexArray::imag, MultiQubit::numAmps, ComplexArray::real, and MultiQubit::stateVec.

Referenced by createMultiQubit(), and main().

```
{
        long long int stateVecSize;
        long long int index;

        // dimension of the state vector
        stateVecSize = multiQubit->numAmps;

        if (DEBUG) printf("stateVecSize=%Ld   now performing init with only one thread:\n",stateVecSize);

        // Can't use multiQubit->stateVec as a private OMP var
        double *stateVecReal = multiQubit->stateVec.real;
        double *stateVecImag = multiQubit->stateVec.imag;

        // initialise the state to |0000..0000>
# ifdef _OPENMP
# pragma omp parallel for \
        default  (none) \
        shared   (stateVecSize, stateVecReal, stateVecImag) \
        private  (index) \
        schedule (static)
# endif
        for (index=0; index<stateVecSize; index++) {
                stateVecReal[index] = 0.0;
                stateVecImag[index] = 0.0;
        }

        if (multiQubit->chunkId==0){
                // zero state |0000..0000> has probability 1
                stateVecReal[0] = 1.0;
                stateVecImag[0] = 0.0;
        }

        if (DEBUG) printf("COMPLETED INIT\n");
}
```

4.3.2.9 double measureInZero (const int *numQubits*, const int *measureQubit*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 534 of file qubits.c.

```
{
        // ----- sizes
        long long int numBlocks,                                           // number of blocks
        sizeBlock,                                           // size of blocks
        sizeHalfBlock;                                       // size of blocks halved
        // ----- indices
        long long int thisBlock,                                           // cur
```

```
  rent block
             index;                                                   // current inde
  x for first half block
        // ----- measured probability
        double   totalProbability, renorm;                                      //
  probability (returned) value
        // ----- temp variables
        long long int thisTask,numTasks;                                   // tas
  k based approach for expose loop with small granularity
        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (measureQubit >= 0 && measureQubit < numQubits);


        // ---------------------------------------------------------------- //
        //            dimensions                                            //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << (measureQubit);                      // number of
   state vector elements to sum,
        // and then the number to skip
        sizeBlock     = 2LL * sizeHalfBlock;                          // size of
   blocks (pairs of measure and skip entries)

        // ---------------------------------------------------------------- //
        //            find probability                                      //
        // ---------------------------------------------------------------- //
        numTasks = 1LL << (numQubits-1);

        // initialise returned value
        totalProbability = 0.0;

        //
        // --- task-based shared-memory parallel implementation
        //
# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \

        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;

                totalProbability += stateVecReal[index]*stateVecReal[index]
                        + stateVecImag[index]*stateVecImag[index];
        }
        renorm=1/sqrt(totalProbability);


# ifdef _OPENMP
# pragma omp parallel for \
        shared    (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \

        private   (thisTask,thisBlock,index) \
        schedule  (static) \
        reduction ( +:totalProbability )
# endif
        for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
                index     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
                stateVecReal[index]=stateVecReal[index]*renorm;
                stateVecImag[index]=stateVecImag[index]*renorm;

                stateVecReal[index+sizeHalfBlock]=0;
                stateVecImag[index+sizeHalfBlock]=0;
        }

        //SCB this is a debugging style check. It is probably useful to leave in, but it could be parallelised I guess
        //  double checkTotal=1.;
        //  for (index=0; index<2*numTasks; index++) {
        //      checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] + stateVecImag[index]*stateVecImag[index]);
        //  }
        //  if (checkTotal>0.00001){printf("Deviation of sum squared amps from unity is %.16f\n",checkTotal); exit(1);}

        return totalProbability;
}
```

4.3.2.10 double probOfFilterOut111 (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 683 of file qubits.c.

References extractBit().

```
{
        long long int index;
        long long int stateVecSize;
        int bit1, bit2, bit3;

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit2 < numQubits);

        stateVecSize = 1LL << numQubits;
        double probOfFilter=0;

# ifdef _OPENMP
# pragma omp parallel for \
        default  (none)                       \
        shared   (stateVecSize, stateVecReal,stateVecImag ) \
        private  (index,bit1,bit2,bit3)                \
        schedule (static)\
        reduction ( +:probOfFilter )
# endif
        for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
                bit2 = extractBit (idQubit2, index);
                bit3 = extractBit (idQubit3, index);
                if (!(bit1 && bit2 && bit3)) {
                        probOfFilter+= stateVecReal[index]*stateVecReal[index] + stateVecImag[index]* stateVecImag [index];
                }
        }
        return probOfFilter;
}
```

4.3.2.11 void quadCPhaseGate (const int *numQubits*, const int *idQubit1*, const int *idQubit2*, const int *idQubit3*, const int *idQubit4*, double ∗restrict *stateVecReal*, double ∗restrict *stateVecImag*)

Definition at line 496 of file qubits.c.

References extractBit().

```
497 {
498         long long int index;
499         long long int stateVecSize;
500         int bit1, bit2, bit3, bit4;
501 
502         // ---------------------------------------------------------------- //
503         //            tests                                                 //
504         // ---------------------------------------------------------------- //
505         assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
      2 < numQubits);
506 
507         stateVecSize = 1LL << numQubits;
508 
509 # ifdef _OPENMP
510 # pragma omp parallel for \
511         default  (none)                      \
512         shared   (stateVecSize, stateVecReal,stateVecImag ) \
513         private  (index,bit1,bit2,bit3,bit4)                   \
514         schedule (static)
515 # endif
516         for (index=0; index<stateVecSize; index++) {
517                 bit1 = extractBit (idQubit1, index);
518                 bit2 = extractBit (idQubit2, index);
519                 bit3 = extractBit (idQubit3, index);
520                 bit4 = extractBit (idQubit4, index);
521                 if (bit1 && bit2 && bit3 && bit4) {
522                         stateVecReal [index] = - stateVecReal [index];
523                         stateVecImag [index] = - stateVecImag [index];
524                 }
525         }
526 }
```

4.3.2.12 void reportState (MultiQubit *multiQubit*)

Definition at line 58 of file qubits.c.

References MultiQubit::chunkId, ComplexArray::imag, MultiQubit::numAmps, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
58                                      {
59         FILE *state;
60         char filename[100];
61         long long int index;
62         sprintf(filename, "state_rank_%d.csv", multiQubit.chunkId);
63         state = fopen(filename, "w");
64         if (multiQubit.chunkId==0) fprintf(state, "real, imag\n");
65 
66         for(index=0; index<multiQubit.numAmps; index++){
67                 fprintf(state, "%.12f, %.12f\n", multiQubit.stateVec.real[index],
    multiQubit.stateVec.imag[index]);
        }
        fclose(state);
}
```

4.3.2.13 void rotateQubitDistributed (MultiQubit *multiQubit*, const int *rotQubit*, Complex *rot1*, Complex *rot2*, ComplexArray *stateVecUp*, ComplexArray *stateVecLo*, ComplexArray *stateVecOut*)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored seperately.

Remarks:

Qubits are zero-based and the the first qubit is the rightmost

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

← ***rotQubit*** qubit to rotate

← ***rot1*** rotation angle

← ***rot2*** rotation angle

← ***stateVecUp*** probability amplitudes in upper half of a block

← ***stateVecLo*** probability amplitudes in lower half of a block

→ ***stateVecOut*** array section to update (will correspond to either the lower or upper half of a block)

Definition at line 224 of file qubits.c.

References ComplexArray::imag, Complex::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and Complex::real.

Referenced by rotateQubit().

```
{
        // ----- temp variables
        double   stateRealUp,stateRealLo,                             // storage
  for previous state values
        stateImagUp,stateImagLo;                             // (used in updates)

        // ----- temp variables
        long long int thisTask;                                   // task based a
  pproach for expose loop with small granularity
        const long long int numTasks=multiQubit.numAmps;

        // (good for shared memory parallelism)

        // ---------------------------------------------------------------- //
        //            tests                                                 //
        // ---------------------------------------------------------------- //
        assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);

        // ---------------------------------------------------------------- //
        //            rotate                                                //
        // ---------------------------------------------------------------- //

        //
        // --- task-based shared-memory parallel implementation
        //
        double rot1Real=rot1.real, rot1Imag=rot1.imag;
        double rot2Real=rot2.real, rot2Imag=rot2.imag;
        double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;
        double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
        double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.imag;
# pragma omp parallel \
        default  (none) \
        shared   (stateVecRealUp,stateVecImagUp,stateVecRealLo,stateVecImagLo,stateVecRealOut,stateVecImagOut, \
                        rot1Real,rot1Imag, rot2Real,rot2Imag) \
        private  (thisTask,stateRealUp,stateImagUp,stateRealLo,stateImagLo)
        {
# pragma omp for \
                schedule (static)
                for (thisTask=0; thisTask<numTasks; thisTask++) {
                        // store current state vector values in temp variables
                        stateRealUp = stateVecRealUp[thisTask];
                        stateImagUp = stateVecImagUp[thisTask];

                        stateRealLo = stateVecRealLo[thisTask];
                        stateImagLo = stateVecImagLo[thisTask];

                        // state[indexUp] = alpha * state[indexUp] - conj(beta) * state[indexLo]
                        stateVecRealOut[thisTask] = rot1Real*stateRealUp - rot1Imag*stateImagUp + rot2Real*stateRealLo + rot2Imag*stateImagLo;
                        stateVecImagOut[thisTask] = rot1Real*stateImagUp + rot1Imag*stateRealUp + rot2Real*stateImagLo - rot2Imag*stateRealLo;
                } // end for loop
        }
} // end of function definition
```

4.3.2.14 void rotateQubitLocal (MultiQubit *multiQubit*, const int *rotQubit*, Complex *alpha*, Complex *beta*)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments. alphaRe = cos(angle1) ∗ cos(angle2)

alphaIm = cos(angle1) ∗ sin(angle2)

betaRe = sin(angle1) ∗ cos(angle3)

betaIm = sin(angle1) ∗ sin(angle3)

Remarks:

Qubits are zero-based and the the first qubit is the rightmost

Parameters:

↔ ***multiQubit*** object representing the set of qubits to be initialised

← ***rotQubit*** qubit to rotate

← ***alpha*** rotation angle

← ***beta*** rotation angle

Definition at line 126 of file qubits.c.

References Complex::imag, ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, Complex::real, ComplexArray::real, and MultiQubit::stateVec.

Referenced by rotateQubit().

```
{
        // ----- sizes
        long long int sizeBlock,                                           // size of blocks
        sizeHalfBlock;                                       // size of blocks halved
        // ----- indices
        long long int thisBlock,                                           // current block
             indexUp,indexLo;                                    // current index and corresponding index in lower half block

        // ----- temp variables
        double   stateRealUp,stateRealLo,                             // storage for previous state values
                 stateImagUp,stateImagLo;                             // (used in updates)
        // ----- temp variables
        long long int thisTask;                                   // task based approach for expose loop with small granularity
        const long long int numTasks=multiQubit.numAmps>>1;
        // (good for shared memory parallelism)


        // ---------------------------------------------------------------- //
        //           tests                                                  //
        // ---------------------------------------------------------------- //
        assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);


        // ---------------------------------------------------------------- //
        //           dimensions                                             //
        // ---------------------------------------------------------------- //
        sizeHalfBlock = 1LL << rotQubit;                               // size of blocks halved
        sizeBlock     = 2LL * sizeHalfBlock;                           // size of blocks


        // ---------------------------------------------------------------- //
        //           rotate                                                 //
        // ---------------------------------------------------------------- //

        //
        // --- task-based shared-memory parallel implementation
        //

        // Can’t use multiQubit.stateVec as a private OMP var
        double *stateVecReal = multiQubit.stateVec.real;
        double *stateVecImag = multiQubit.stateVec.imag;
        double alphaImag=alpha.imag, alphaReal=alpha.real;
        double betaImag=beta.imag, betaReal=beta.real;

# ifdef _OPENMP
# pragma omp parallel \
        default  (none) \
        shared   (sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag, alphaReal,alphaImag, betaReal,betaImag) \
        private  (thisTask,thisBlock ,indexUp,indexLo, stateRealUp,stateImagUp,stateRealLo,stateImagLo)
# endif
        {
```

```
# ifdef _OPENMP
# pragma omp for \
                schedule (static)
# endif
                for (thisTask=0; thisTask<numTasks; thisTask++) {

                        thisBlock   = thisTask / sizeHalfBlock;
                        indexUp     = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
                        indexLo     = indexUp + sizeHalfBlock;

                        // store current state vector values in temp variables
                        stateRealUp = stateVecReal[indexUp];
                        stateImagUp = stateVecImag[indexUp];

                        stateRealLo = stateVecReal[indexLo];
                        stateImagLo = stateVecImag[indexLo];

                        // state[indexUp] = alpha * state[indexUp] - conj(beta) * state[indexLo]
                        stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag*stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
                        stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag*stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;

                        // state[indexLo] = beta  * state[indexUp] + conj(alpha) * state[indexLo]
                        stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*stateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
                        stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*stateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
                } // end for loop
        }

} // end of function definition
```

4.4 qubits_env_local.c File Reference

An implementation of the API in qubits_env_wrapper.h for a local (non-MPI) environment. `#include <stdlib.h>`

`#include "qubits.h"`

`#include "qubits_env_wrapper.h"`

Functions

- void initQUESTEnv (QUESTEnv ∗env)
- void syncQUESTEnv (QUESTEnv env)
- void closeQUESTEnv (QUESTEnv env)
- double calcTotalProbability (MultiQubit multiQubit)
- void rotateQubit (MultiQubit multiQubit, const int rotQubit, Complex alpha, Complex beta)
- double findProbabilityOfZero (MultiQubit multiQubit, const int measureQubit)

4.4.1 Detailed Description

An implementation of the API in qubits_env_wrapper.h for a local (non-MPI) environment.

Definition in file qubits_env_local.c.

4.4.2 Function Documentation

4.4.2.1 double calcTotalProbability (MultiQubit *multiQubit*)

Definition at line 43 of file qubits_env_local.c.

Referenced by main().

```
43                                                     {
44         double pTotal=0;
45         long long int index;
46         long long int numAmpsPerRank = multiQubit.numAmps;
47         for (index=0; index<numAmpsPerRank; index++){
48                 pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
     index];
49                 pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
     index];
50         }
51         return pTotal;
52 }
```

4.4.2.2 void closeQUESTEnv (QUESTEnv *env*)

Definition at line 20 of file qubits_env_local.c.

Referenced by main().

```
20                                    {
21         // MPI finalize goes here in MPI version. Call this function anyway for c
     onsistency
22 }
```

4.4.2.3 double findProbabilityOfZero (MultiQubit *multiQubit*, const int *measureQubit*)

Definition at line 94 of file qubits_env_local.c.

```
96 {
97         double stateProb=0;
98         stateProb = findProbabilityOfZeroLocal(multiQubit, measureQubit);
99         return stateProb;
100 }
```

4.4.2.4 void initQUESTEnv (QUESTEnv * *env*)

Definition at line 9 of file qubits_env_local.c.

Referenced by main().

```
9                              {
10          // init MPI environment
11          int rank, numRanks;
12          env->rank=0;
13          env->numRanks=1;
14 }
```

4.4.2.5 void rotateQubit (MultiQubit *multiQubit*, const int *rotQubit*, Complex *alpha*, Complex *beta*)

Definition at line 75 of file qubits_env_local.c.

Referenced by main().

```
77 {
78          // all values required to update state vector lie in this rank
79          rotateQubitLocal(multiQubit, rotQubit, alpha, beta);
80 }
```

4.4.2.6 void syncQUESTEnv (QUESTEnv *env*)

Definition at line 16 of file qubits_env_local.c.

```
16                             {
17          // MPI Barrier goes here in MPI version.
18 }
```

4.5 qubits_env_mpi.c File Reference

An implementation of the API in qubits_env_wrapper.h for an MPI environment. `#include <mpi.h>`

`#include <stdlib.h>`

`#include <stdio.h>`

`#include "qubits.h"`

`#include "qubits_env_wrapper.h"`

Defines

- #define DEBUG 0

Functions

- void initQUESTEnv (QUESTEnv ∗env)
- void syncQUESTEnv (QUESTEnv env)
- void closeQUESTEnv (QUESTEnv env)
- int isChunkToSkipInFindPZero (int chunkId, int chunkSize, int measureQubit)

    Find chunks to skip when calculating probability of qubit being zero.

- double calcTotalProbability (MultiQubit multiQubit)
- int chunkIsUpper (int chunkId, int chunkSize, int rotQubit)

  Returns whether a given chunk in position chunkId is in the upper or lower half of a block.

- void getRotAngle (int chunkIsUpper, Complex ∗rot1, Complex ∗rot2, Complex alpha, Complex beta)

  Get rotation values for a given chunk.

- int getChunkPairId (int chunkIsUpper, int chunkId, int chunkSize, int rotQubit)

  get position of corresponding chunk, holding values required to update values in my chunk (with chunkId) when rotating rotQubit.

- int halfMatrixBlockFitsInChunk (int chunkSize, int rotQubit)

  return whether the current qubit rotation will use blocks that fit within a single chunk.

- void rotateQubit (MultiQubit multiQubit, const int rotQubit, Complex alpha, Complex beta)
- double findProbabilityOfZero (MultiQubit multiQubit, const int measureQubit)

4.5.1 Detailed Description

An implementation of the API in qubits_env_wrapper.h for an MPI environment.

Definition in file qubits_env_mpi.c.

4.5.2 Define Documentation

4.5.2.1 #define DEBUG 0

Definition at line 10 of file qubits_env_mpi.c.

4.5.3 Function Documentation

4.5.3.1 double calcTotalProbability (MultiQubit *multiQubit*)

Definition at line 81 of file qubits_env_mpi.c.

References DEBUG, ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numChunks, ComplexArray::real, and MultiQubit::stateVec.

```
81                                                   {
82          double pTotal=0;
83          double allRankTotals=0;
84          long long int index;
85          long long int numAmpsPerRank = multiQubit.numAmps;
86          for (index=0; index<numAmpsPerRank; index++){
87                  pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
      index];
88                  pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
      index];
89          }
90          if (DEBUG) printf("before calc prob. %d\n", multiQubit.numChunks);
91          if (multiQubit.numChunks>1) MPI_Reduce(&pTotal, &allRankTotals, 1, MPI_DO
      UBLE, MPI_SUM, 0, MPI_COMM_WORLD);
92          else allRankTotals=pTotal;
93
94          return allRankTotals;
95 }
```

4.5.3.2 int chunkIsUpper (int *chunkId*, int *chunkSize*, int *rotQubit*)

Returns whether a given chunk in position chunkId is in the upper or lower half of a block.

Parameters:

← ***chunkId*** id of chunk in state vector

← ***chunkSize*** number of amps in chunk

← ***rotQubit*** qubit being rotated

Returns:

1: chunk is in upper half of block, 0: chunk is in lower half of block

Definition at line 106 of file qubits_env_mpi.c.

Referenced by rotateQubit().

```
107 {
108          long long int sizeHalfBlock = 1LL << (rotQubit);
109          long long int sizeBlock = sizeHalfBlock*2;
110          long long int posInBlock = (chunkId*chunkSize) % sizeBlock;
111          return posInBlock<sizeHalfBlock;
112 }
```

4.5.3.3 void closeQUESTEnv (QUESTEnv *env*)

Definition at line 36 of file qubits_env_mpi.c.

```
36                                {
37          int finalized;
38          MPI_Finalized(&finalized);
39          if (!finalized) MPI_Finalize();
40          else printf("ERROR: Trying to close QUESTEnv multiple times. Ignoring\n")
      ;
41 }
```

4.5.3.4 double findProbabilityOfZero (MultiQubit *multiQubit*, const int *measureQubit*)

Definition at line 257 of file qubits_env_mpi.c.

References MultiQubit::chunkId, findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), halfMatrixBlockFitsInChunk(), isChunkToSkipInFindPZero(), and MultiQubit::numAmps.

```
{
        double stateProb=0, totalStateProb=0;
        int skipValuesWithinRank = halfMatrixBlockFitsInChunk(multiQubit.numAmps, measureQubit);
        if (skipValuesWithinRank) {
                stateProb = findProbabilityOfZeroLocal(multiQubit, measureQubit);
        } else {
                if (!isChunkToSkipInFindPZero(multiQubit.chunkId, multiQubit.numAmps, measureQubit)){
                        stateProb = findProbabilityOfZeroDistributed(multiQubit, measureQubit);
                } else stateProb = 0;
        }
        MPI_Reduce(&stateProb, &totalStateProb, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
        return totalStateProb;
}
```

4.5.3.5 int getChunkPairId (int *chunkIsUpper*, int *chunkId*, int *chunkSize*, int *rotQubit*)

get position of corresponding chunk, holding values required to update values in my chunk (with chunkId) when rotating rotQubit.

Parameters:

← ***chunkIsUpper*** 1: chunk is in upper half of block, 0: chunk is in lower half

← ***chunkId*** id of chunk in state vector

← ***chunkSize*** number of amps in chunk

← ***rotQubit*** qubit being rotated

Returns:

chunkId of chunk required to rotate rotQubit

Definition at line 148 of file qubits_env_mpi.c.

Referenced by rotateQubit().

```
{
        long long int sizeHalfBlock = 1LL << (rotQubit);
        int chunksPerHalfBlock = sizeHalfBlock/chunkSize;
        if (chunkIsUpper){
                return chunkId + chunksPerHalfBlock;
        } else {
                return chunkId - chunksPerHalfBlock;
        }
}
```

4.5.3.6 void getRotAngle (int *chunkIsUpper*, Complex ∗ *rot1*, Complex ∗ *rot2*, Complex *alpha*, Complex *beta*)

Get rotation values for a given chunk.

Parameters:

← ***chunkIsUpper*** 1: chunk is in upper half of block, 0: chunk is in lower half

→ ***rot1,rot2*** rotation values to use, allocated for upper/lower such that

```
stateUpper = rot1 * stateUpper + conj(rot2)  * stateLower
```

or

```
stateLower = rot1 * stateUpper + conj(rot2)  * stateLower
```

← ***alpha,beta*** initial rotation values

Definition at line 127 of file qubits_env_mpi.c.

References Complex::imag, and Complex::real.

Referenced by rotateQubit().

```
128 {
129         if (chunkIsUpper){
130                 *rot1=alpha;
131                 rot2->real=-beta.real;
132                 rot2->imag=-beta.imag;
133         } else {
134                 *rot1=beta;
135                 *rot2=alpha;
136         }
137 }
```

4.5.3.7 int halfMatrixBlockFitsInChunk (int *chunkSize*, int *rotQubit*)

return whether the current qubit rotation will use blocks that fit within a single chunk.

Parameters:

← ***chunkSize*** number of amps in chunk

← ***rotQubit*** qubit being rotated

Returns:

1: one chunk fits in one block 0: chunk is larger than block

Definition at line 167 of file qubits_env_mpi.c.

Referenced by findProbabilityOfZero(), and rotateQubit().

```
168 {
169         long long int sizeHalfBlock = 1LL << (rotQubit);
170         if (chunkSize > sizeHalfBlock) return 1;
171         else return 0;
172 }
```

4.5.3.8 void initQUESTEnv (QUESTEnv * *env*)

Definition at line 12 of file qubits_env_mpi.c.

References DEBUG, QUESTEnv::numRanks, and QUESTEnv::rank.

```
12                                {
13          // init MPI environment
14          int rank, numRanks, initialized;
15          MPI_Initialized(&initialized);
16          if (!initialized){
17                  MPI_Init(NULL, NULL);
18                  MPI_Comm_size(MPI_COMM_WORLD, &numRanks);
19                  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
20
21                  if (DEBUG) {
22                          char hostName[256];
23                          int hostNameLen;
24                          MPI_Get_processor_name(hostName, &hostNameLen);
25                          printf("rank %d on host %s\n", rank, hostName);
26                  }
27                  env->rank=rank;
28                  env->numRanks=numRanks;
29          } else printf("ERROR: Trying to initialize QUESTEnv multiple times. Ignor
      ing\n");
30 }
```

4.5.3.9 int isChunkToSkipInFindPZero (int *chunkId*, int *chunkSize*, int *measureQubit*)

Find chunks to skip when calculating probability of qubit being zero. When calculating probability of a bit q being zero, sum up 2^q values, then skip 2^q values, etc. This function finds if an entire chunk is in the range of values to be skipped

Parameters:

← ***chunkId*** id of chunk in state vector

← ***chunkSize*** number of amps in chunk

← ***measureQubi*** qubit being measured

Returns:

int -- 1: skip, 0: don't skip

Definition at line 54 of file qubits_env_mpi.c.

Referenced by findProbabilityOfZero().

```
54                                                                          {
55          long long int sizeHalfBlock = 1LL << (measureQubit);
56          int numChunksToSkip = sizeHalfBlock/chunkSize;
57          // calculate probability by summing over numChunksToSkip, then skipping n
      umChunksToSkip, etc
58          int bitToCheck = chunkId & numChunksToSkip;
59          return bitToCheck;
60 }
```

4.5.3.10 void rotateQubit (MultiQubit *multiQubit*, const int *rotQubit*, Complex *alpha*, Complex *beta*)

Definition at line 194 of file qubits_env_mpi.c.

References MultiQubit::chunkId, chunkIsUpper(), getChunkPairId(), getRotAngle(), halfMatrixBlockFitsInChunk(), ComplexArray::imag, MultiQubit::numAmps, MultiQubit::pairStateVec, ComplexArray::real, rotateQubitDistributed(), rotateQubitLocal(), and MultiQubit::stateVec.

```
{
        // flag to require memory exchange. 1: an entire block fits on one rank, 0: at most half a block fits on one rank
        int useLocalDataOnly = halfMatrixBlockFitsInChunk(multiQubit.numAmps, rotQubit);
        Complex rot1, rot2;

        // rank's chunk is in upper half of block
        int rankIsUpper;
        int pairRank; // rank of corresponding chunk

        // MPI send/receive vars
        int TAG=100;
        MPI_Status status;

        double *stateVecReal, stateVecImag, stateVecRealPair, stateVecImagPair;


        if (useLocalDataOnly){
                // all values required to update state vector lie in this rank
                rotateQubitLocal(multiQubit, rotQubit, alpha, beta);
        } else {
                // need to get corresponding chunk of state vector from other rank
                rankIsUpper = chunkIsUpper(multiQubit.chunkId, multiQubit.numAmps, rotQubit);
                getRotAngle(rankIsUpper, &rot1, &rot2, alpha, beta);
                pairRank = getChunkPairId(rankIsUpper, multiQubit.chunkId, multiQubit.numAmps, rotQubit);
                //printf("%d rank has pair rank: %d\n", multiQubit.rank, pairRank);
                // get corresponding values from my pair
                MPI_Sendrecv(multiQubit.stateVec.real, multiQubit.numAmps, MPI_DOUBLE, pairRank, TAG,
                                 multiQubit.pairStateVec.real, multiQubit.numAmps, MPI_DOUBLE, pairRank, TAG,
                                 MPI_COMM_WORLD, &status);
                //printf("rank: %d err: %d\n", multiQubit.rank, err);
                MPI_Sendrecv(multiQubit.stateVec.imag, multiQubit.numAmps, MPI_DOUBLE, pairRank, TAG,
                                multiQubit.pairStateVec.imag, multiQubit.numAmps, MPI_DOUBLE, pairRank, TAG,
                                MPI_COMM_WORLD, &status);
                // this rank's values are either in the upper of lower half of the block. send values to rotateQubitDistributed
                // in the correct order
                if (rankIsUpper){
                        rotateQubitDistributed(multiQubit,rotQubit,rot1,rot2,
                                multiQubit.stateVec, //upper
                                multiQubit.pairStateVec, //lower
                                multiQubit.stateVec); //output
                } else {
                        rotateQubitDistributed(multiQubit,rotQubit,rot1,rot2,
                                multiQubit.pairStateVec, //upper
                                multiQubit.stateVec, //lower
                                multiQubit.stateVec); //output
                }
        }
}
```

4.5.3.11 void syncQUESTEnv (QUESTEnv *env*)

Definition at line 32 of file qubits_env_mpi.c.

```
32                              {
33         MPI_Barrier(MPI_COMM_WORLD);
34 }
```

4.6 qubits_env_wrapper.h File Reference

Specifications for QUEST library functions whose implementation depends on environment (local, MPI).

Functions

- void initQUESTEnv (QUESTEnv ∗env)
- void closeQUESTEnv (QUESTEnv env)
- void syncQUESTEnv (QUESTEnv env)
- double calcTotalProbability (MultiQubit multiQubit)
- void rotateQubit (MultiQubit multiQubit, const int rotQubit, Complex alpha, Complex beta)
- double findProbabilityOfZero (MultiQubit multiQubit, const int measureQubit)

4.6.1 Detailed Description

Specifications for QUEST library functions whose implementation depends on environment (local, MPI).

Definition in file qubits_env_wrapper.h.

4.6.2 Function Documentation

4.6.2.1 double calcTotalProbability (MultiQubit *multiQubit*)

Definition at line 43 of file qubits_env_local.c.

References DEBUG, ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numChunks, ComplexArray::real, and MultiQubit::stateVec.

Referenced by main().

```
43                                                 {
44         double pTotal=0;
45         long long int index;
46         long long int numAmpsPerRank = multiQubit.numAmps;
47         for (index=0; index<numAmpsPerRank; index++){
48                 pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
       index];
49                 pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
       index];
50         }
51         return pTotal;
52 }
```

4.6.2.2 void closeQUESTEnv (QUESTEnv *env*)

Definition at line 20 of file qubits_env_local.c.

Referenced by main().

```
20                              {
21          // MPI finalize goes here in MPI version. Call this function anyway for c
     onsistency
22 }
```

4.6.2.3 double findProbabilityOfZero (MultiQubit *multiQubit*, const int *measureQubit*)

Definition at line 94 of file qubits_env_local.c.

References MultiQubit::chunkId, findProbabilityOfZeroDistributed(), findProbabilityOfZeroLocal(), halfMatrixBlockFitsInChunk(), isChunkToSkipInFindPZero(), and MultiQubit::numAmps.

```
96 {
97          double stateProb=0;
98          stateProb = findProbabilityOfZeroLocal(multiQubit, measureQubit);
99          return stateProb;
100 }
```

4.6.2.4 void initQUESTEnv (QUESTEnv ∗ *env*)

Definition at line 9 of file qubits_env_local.c.

References DEBUG, QUESTEnv::numRanks, and QUESTEnv::rank.

Referenced by main().

```
9                              {
10          // init MPI environment
11          int rank, numRanks;
12          env->rank=0;
13          env->numRanks=1;
14 }
```

4.6.2.5 void rotateQubit (MultiQubit *multiQubit*, const int *rotQubit*, Complex *alpha*, Complex *beta*)

Definition at line 75 of file qubits_env_local.c.

References MultiQubit::chunkId, chunkIsUpper(), getChunkPairId(), getRotAngle(), halfMatrixBlockFitsInChunk(), ComplexArray::imag, MultiQubit::numAmps, MultiQubit::pairStateVec, ComplexArray::real, rotateQubitDistributed(), rotateQubitLocal(), and MultiQubit::stateVec.

Referenced by main().

```
{
        // all values required to update state vector lie in this rank
        rotateQubitLocal(multiQubit, rotQubit, alpha, beta);
}
```

4.6.2.6 void syncQUESTEnv (QUESTEnv *env*)

Definition at line 16 of file qubits_env_local.c.

```
16                            {
17        // MPI Barrier goes here in MPI version.
18 }
```

Index