Distributed QUEST

Generated by Doxygen 1.6.1

Mon Apr 3 12:28:04 2017

CONTENTS

Contents

1	Data	a Structure Index	1
	1.1	Data Structures	1
2	File	Index	1
	2.1	File List	1
3	Data	a Structure Documentation	2
	3.1	Complex Struct Reference	2
		3.1.1 Detailed Description	2
			2
	3.2	ComplexArray Struct Reference	2
		3.2.1 Detailed Description	3
			3
	3.3	MultiQubit Struct Reference	3
		3.3.1 Detailed Description	4
			4
	3.4	QUESTEnv Struct Reference	5
			5
			5
4	File	Documentation	6
	4.1		6
		•	7
			7
			7
			1
	4.2		1
			2
		•	2
			2
	4.3		6
	T. J		7
		F	.7
	4.4		.1
	7.7		.1
		•	.2
		7.7.2 1 unction Documentation	~

1 Data Structure Index 1

	4.5	qubits_env_mpi.c File Reference	43
		4.5.1 Detailed Description	44
		4.5.2 Define Documentation	44
		4.5.3 Function Documentation	44
	4.6	qubits_env_wrapper.h File Reference	50
		4.6.1 Detailed Description	50
		4.6.2 Function Documentation	50
1	D	ata Structure Index	
1.	1 E	Data Structures	
Н	ere are	e the data structures with brief descriptions:	
	Con	nplex (Represents one complex number)	2
		nplexArray (Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components)	2
	Mul	tiQubit (Represents a system of qubits)	3
	QUI	ESTEnv (Information about the environment the program is running in)	5
2	Fi	ile Index	
2.	1 F	File List	
Н	ere is	a list of all files with brief descriptions:	
	basi	cTemplate.c (Basic template for using the QUEST library)	6
	qubi	its.c (The core of the QUEST Library)	11
		its.h (Structs and specifications for functions that can be used from any environment (local, MPI) $)$	26
	-	its_env_local.c (An implementation of the API in qubits_env_wrapper.h for a local (non-MPI) environment)	41
	_	its_env_mpi.c (An implementation of the API in qubits_env_wrapper.h for an MPI environment)	43
	-	its_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 calc Total Probability(), create MultiQubit(), destroy MultiQubit(), find Probability Of Zero Distributed(), find Probability Of Zero Local(), in it State Vec(), report State(), rotate Qubit(), rotate Qubit Distributed(), and rotate Qubit Local().

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 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.

 $Referenced\ by\ createMultiQubit(),\ findProbabilityOfZero(),\ initStateVec(),\ reportState(),\ and\ rotate-Qubit().$

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().

4 File Documentation 6

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().

```
38
           // These two lines will automatically set up the environment (multinode,
39
           // openMP only etc)
40
           QUESTEnv env;
41
           initQUESTEnv(&env);
42
           // model vars
4.3
44
           int numQubits;
           long int index;
46
           long int numAmps;
47
48
           // get number of qubits from command line argument
49
           if (narg >= 2) {
50
                    numQubits = atoi(varg[1]);
51
                    if (numQubits < 1 || numQubits > maxNumQubits) {
                             printf(" *** error: argument %d out of range (1 -- %d) \n"
52
      , numQubits, maxNumQubits);
53
                             exit (EXIT_FAILURE);
54
55
           } else {
                    printf(" *** error: too few arguments, number of qubits expected\
56
57
                    exit (EXIT_FAILURE);
58
59
           // Reporting
60
61
           numAmps = 1L << numQubits;</pre>
62
           if (env.rank==0) {
6.3
                    printf("Demo of single qubit rotations.\n");
64
                    printf("Number of qubits is %d.\n", numQubits);
                    printf("Number of amps is %ld.\n", numAmps);
65
66
            }
67
68
           // CREATE QUBIT OBJECT: REQUIRED ONCE PER MULTIQUBIT OBJECT
69
           // Before doing any operations on a set of qubits, create the MultiQubit
      object that will be used to
70
           // represent the qubits
71
           MultiQubit multiQubit;
72
           createMultiQubit(&multiQubit, numQubits, env);
73
74
           // initialise the state to |0000..0>
           initStateVec (&multiQubit);
7.5
76
77
78
            // ===== ROTATIONS
79
80
           //
81
82
           // INITIALISE QUBIT ROTATION
           \ensuremath{//} Edit these lines to change rotation angle
8.3
84
           double ang1, ang2, ang3;
85
           Complex alpha, beta;
86
87
            // define rotation angles
88
           double angles[MaxAngles][3] = {
                    { 1.2320, 0.4230, -0.6523},
89
                    { 2.1213, 0.0000, 3.6520}, 
{-3.1213, 5.0230, 0.1230}, 
{ 5.2341, -3.1001, -1.2340},
90
91
92
                    \{-0.1234, -0.9876, 4.1234\}
93
94
            };
95
            int numAngles=5,iAngle;
96
97
           // rotate
           ang1 = angles[0][0];
ang2 = angles[0][1];
98
99
100
            ang3 = angles[0][2];
101
```

```
102
            alpha.real = cos(ang1) * cos(ang2);
103
            alpha.imag = cos(ang1) * sin(ang2);
            beta.real = sin(ang1) * cos(ang3);
beta.imag = sin(ang1) * sin(ang3);
104
105
106
107
            int rotQubit;
108
109
            // DO QUBIT ROTATION
110
            // Edit these lines to perform rotations as required
111
            for (rotQubit=0; rotQubit<numQubits; rotQubit++) {</pre>
                    // do rotation of each qubit
112
113
                     rotateQubit(multiQubit,rotQubit,alpha,beta);
114
115
            // END QUBIT ROTATION
116
117
            // Verification: check vector size is unchanged
118
            double totalProbability;
119
            totalProbability = calcTotalProbability(multiQubit);
120
            if (env.rank==0) printf("VERIFICATION: total probability=%.14f\n", totalP
     robability);
121
122
            // report state vector to file
            if (REPORT_STATE) {
123
124
                    reportState(multiQubit);
125
            }
126
127
128
129
            // ===== perform a measurement
130
            //
131
            int measureQubit;
132
           double stateProb, randProb;
133
            /\star // keep time \star/
134
            /* wtime_start = system_timer (); */
135
            // measure
136
137 /*
138
            for (measureQubit=0; measureQubit<numQubits; measureQubit++) {</pre>
139
            //for (measureQubit=0; measureQubit<1; measureQubit++) {</pre>
140
                     syncQUESTEnv(env);
                     wtime_start = system_timer ();
141
142
                     stateProb = findProbabilityOfZero (env.rank, numAmpsPerRank, numQ
     ubits, measureQubit, stateVecReal, stateVecImag);
143
                     syncQUESTEnv(env);
144
                     wtime_stop = system_timer ();
                    if (env.rank==0) printf(" probability of 0 for qubit %d = %.14f
145
      \n", measureQubit, stateProb);
146
                    if (env.rank==0) printf(" measurement qubit %d: elapsed time = %f
       [s]\n", measureQubit, wtime_stop - wtime_start);
147
           }
148 */
            /\star // keep time \star/
149
150
            /* wtime_stop = system_timer (); */
151
            /\star // ---- timing report \star/
152
            /* printf(" measurement: elapsed time = %f [s]\n", wtime_stop - wtime_sta
     rt); */
154
155
156
157
            // ===== two qubit phase gate
158
            //
            /\star // keep time \star/
159
            /* wtime_start = system_timer (); */
160
161 /*
162
            // two qubit phase gate
163
            if (numQubits >= 7) {
```

```
164
                    wtime_start = system_timer ();
165
                    controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 0, 2, stat
      eVecReal, stateVecImag);
166
                    wtime_stop = system_timer ();
                    printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_st
167
      op - wtime_start);
168
                    wtime_start = system_timer (); controlPhaseGate (env.rank, numAmp
      sPerRank, numQubits, 1, 3, stateVecReal, stateVecImag); wtime_stop = system_timer
      (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_
169
                    wtime_start = system_timer (); controlPhaseGate (env.rank, numAmp
      sPerRank, numQubits, 2, 4, stateVecReal, stateVecImag); wtime_stop = system_timer
      (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_
      start);
170
                    wtime_start = system_timer (); controlPhaseGate (env.rank, numAmp
      sPerRank, numQubits, 3, 5, stateVecReal, stateVecImag); wtime_stop = system_timer
       (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_
                    wtime_start = system_timer (); controlPhaseGate (env.rank, numAmp
171
      sPerRank, numQubits, 4, 6, stateVecReal, stateVecImag); wtime_stop = system_timer
      (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_
      start):
172
173 */
174
            totalProbability = calcTotalProbability(multiQubit);
175
            if (env.rank==0) printf("VERIFICATION: total probability=%.14f\n", totalP
     robability);
176 /*
177 if (env.rank==0){
           printf("\n\nIn rank %d, the following is the final state after rotations.
178
      \n\n",env.rank);
179
           printf("codeOutput=[\n");
            for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",state</pre>
180
      VecReal[index], stateVecImag[index]);
181
           printf("];\n\n");
182 }
183 syncQUESTEnv(env);
184
185 if (env.rank==1) {
           printf("\n\nIn rank %d, the following is the final state after rotations.
      \n'', env.rank);
187
           printf("codeOutput=[\n");
188
            for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",state</pre>
      VecReal[index], stateVecImag[index]);
189
           printf("];\n\n");
190 }
191 syncQUESTEnv(env);
192 */
            /* // keep time */
193
194
            /* wtime_stop = system_timer (); */
195
            /\star // ---- timing report \star/
196
            /* printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop -
197
     wtime_start); */
198
199
2.00
201
                    ==== CLEANUP
2.02
2.03
204
            // free memory
205
206
            // REQUIRED ONCE PER MULTIQUBIT OBJECT
207
            // When all operations on a set of qubits are completed, destroy the obje
208
            destroyMultiQubit (multiQubit, env);
209
```

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 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().

```
463
          // ----- //
464
465
         assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
    2 < numQubits);</pre>
466
467
         // -----//
468
             initialise the state to |0000..0>
470
471
472
         // dimension of the state vector
473
         stateVecSize = 1LL << numQubits;</pre>
474
475 # ifdef _OPENMP
476 # pragma omp parallel for \
       default (none)
shared (stateV
                 (stateVecSize, stateVecReal, stateVecImag ) \
478
        private (index,bit1,bit2)
479
         schedule (static)
480
481 # endif
for (index=0; index<stateVecSize; index++) {
                bit1 = extractBit (idQubit1, index);
483
                bit2 = extractBit (idQubit2, index);
484
                if (bit1 && bit2) {
486
                      stateVecReal [index] = - stateVecReal [index];
487
                      stateVecImag [index] = - stateVecImag [index];
488
                }
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 {
          long long int numAmps = 1L << numQubits;</pre>
18
19
          long long int numAmpsPerRank = numAmps/env.numRanks;
2.0
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){
                  multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(mu
24
      ltiQubit->pairStateVec.real));
                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;
          multiQubit->numAmps = numAmpsPerRank;
          multiQubit->chunkId = env.rank;
42
43
          multiQubit->numChunks = env.numRanks;
44
4.5
           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().

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().

```
626 {
627
          long long int index;
628
          long long int stateVecSize;
62.9
           int bit1, bit2, bit3;
          // ------//
631
          //
632
          // -----//
          assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
634
     2 < numQubits);</pre>
635
636
           stateVecSize = 1LL << numQubits;</pre>
          double probOfFilter=0;
638
639 # ifdef _OPENMP
640 \# pragma omp parallel for \setminus
641
        default (none)
642
          shared (stateVecSize, stateVecReal, stateVecImag) \
         private (index,bit1,bit2,bit3)
643
          schedule (static) \
644
          reduction ( +:probOfFilter )
645
646 # endif
           for (index=0; index<stateVecSize; index++) {</pre>
647
                  bit1 = extractBit (idQubit1, index);
649
                  bit2 = extractBit (idQubit2, index);
650
                  bit3 = extractBit (idQubit3, index);
                  if (!(bit1 && bit2 && bit3)) {
651
652
                         probOfFilter+= stateVecReal[index]*stateVecReal[index] +
     stateVecImag[index]* stateVecImag [index];
653
                 }
654
          if (probOfFilter<1e-16) { printf("Extremely small or negative profOfFilt
     er=%.8e; aborting! \n",probOfFilter); exit(1);}
656
          double myNorm=1/sqrt(probOfFilter);
657
658 # ifdef _OPENMP
659 # pragma omp parallel for \
          default (none) shared (stateV
660
                   (stateVecSize, stateVecReal, stateVecImag, myNorm ) \
661
          private (index,bit1,bit2,bit3)
663
          schedule (static)
664 # endif
for (index=0; index<stateVecSize; index++) {
                  bit1 = extractBit (idQubit1, index);
666
                  bit2 = extractBit (idQubit2, index);
667
668
                  bit3 = extractBit (idQubit3, index);
669
                  if ((bit1 && bit2 && bit3)) {
670
                         stateVecReal[index]=0;
                          stateVecImag [index]=0;
671
672
                  }else{
673
                          stateVecReal[index] *= myNorm;
674
                          stateVecImag[index] *= myNorm;
675
676
          }
          return probOfFilter;
677
678 }
```

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
378
379
          double totalProbability;
                                                                 // probabil
    ity (returned) value
          // ---- temp variables
380
381
          long long int thisTask;
                                                             // task based a
    pproach for expose loop with small granularity
382
          long long int numTasks=multiQubit.numAmps;
          // (good for shared memory parallelism)
383
384
385
          //
             tests
          // ------ //
387
388
          assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);</pre>
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
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) \
         schedule (static) \
410
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
                 // summation -- kahan correction
419
420
                 y = stateVecReal[thisTask]*stateVecReal[thisTask]
421
                 + stateVecImag[thisTask] * stateVecImag[thisTask] - c;
                 t = totalProbability + y;
422
423
                 c = (t - totalProbability) - y;
```

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.

 $References\ Complex Array::imag,\ MultiQubit::num Amps,\ MultiQubit::num Qubits,\ Complex Array::real,\ and\ MultiQubit::state Vec.$

Referenced by findProbabilityOfZero().

```
291 {
         // ---- sizes
2.92
293
         long long int sizeBlock,
                                                              // siz
    e of blocks
2.94
                                                   // size of blocks ha
         sizeHalfBlock;
        // ---- indices
long long int thisBlock,
295
296
                                                              // cur
    rent block
297
                                                       // current inde
            index;
    x for first half block
      // ---- measured probability
299
        double totalProbability;
                                                          // probabil
    ity (returned) value
        // ---- temp variables
300
301
         long long int thisTask;
                                                       // task based a
    pproach for expose loop with small granularity
        long long int numTasks=multiQubit.numAmps>>1;
302
303
         // (good for shared memory parallelism)
304
         // ----- //
305
306
        // tests
         // ----- //
307
         assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);</pre>
308
309
310
311
         // dimensions
312
                                                      //
         // ------//
313
                                                       // number of
314
         sizeHalfBlock = 1LL << (measureQubit);</pre>
     state vector elements to sum,
```

```
315
           // and then the number to skip
316
          sizeBlock = 2LL * sizeHalfBlock;
                                                                       // size of
      blocks (pairs of measure and skip entries)
317
           //
               find probability
319
           // ----- //
320
           // initialise returned value
322
323
           totalProbability = 0.0;
324
325
           // initialise correction for kahan summation
           printf("sizeHalfBlock=%Ld sizeBlock=%Ld numTasks=%Ld\n",sizeHalfBlock,siz
     eBlock, numTasks);
327
328
          // --- task-based shared-memory parallel implementation
329
330
331
332
           double *stateVecReal = multiQubit.stateVec.real;
           double *stateVecImag = multiQubit.stateVec.imag;
333
334
335 # ifdef _OPENMP
336 \# pragma omp parallel for \setminus
         shared (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
          private (thisTask,thisBlock,index) \
339
          schedule (static) \
340
           reduction ( +:totalProbability )
341 # endif
342
          for (thisTask=0; thisTask<numTasks; thisTask++) {</pre>
                  thisBlock = thisTask / sizeHalfBlock;
343
344
                          = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
                  index
345
346
                  if (index<0) { printf("ABORTING as index=%Ld with thisBlock = %Ld
      thisTask=%Ld \n", index,thisBlock,thisTask); exit(1);}
347
                   // summation -- simple implementation
348
                   totalProbability += stateVecReal[index]*stateVecReal[index]
349
                          + stateVecImag[index]*stateVecImag[index];
351
352
                  // summation -- kahan correction
353
354
                  y = stateVecReal[index]*stateVecReal[index]
355
                   + stateVecImag[index] * stateVecImag[index] - c;
356
                  t = totalProbability + y;
357
                  c = (t - totalProbability) - y;
358
                  totalProbability = t;
359
360
361
         }
362
363
         return totalProbability;
364 }
```

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().

```
77 {
78
           long long int stateVecSize;
           long long int index;
79
80
81
           // dimension of the state vector
           stateVecSize = multiQubit->numAmps;
82
83
84
           if (DEBUG) printf("stateVecSize=%Ld now performing init with only one t
      hread: \n", stateVecSize);
86
           // Can't use multiQubit->stateVec as a private OMP var
87
           double *stateVecReal = multiQubit->stateVec.real;
           double *stateVecImag = multiQubit->stateVec.imag;
88
89
90
           // initialise the state to |0000..0000\rangle
91 # ifdef _OPENMP
92 \# pragma omp parallel for \setminus
93
           default (none) \
94
                    (stateVecSize, stateVecReal, stateVecImag) \
           shared
95
           private (index) \
96
           schedule (static)
97 # endif
98
           for (index=0; index<stateVecSize; index++) {</pre>
99
                   stateVecReal[index] = 0.0;
100
                    stateVecImag[index] = 0.0;
101
            }
102
103
            if (multiQubit->chunkId==0) {
                    // zero state |0000..0000> has probability 1
104
                    stateVecReal[0] = 1.0;
105
106
                    stateVecImag[0] = 0.0;
107
            }
108
109
            if (DEBUG) printf("COMPLETED INIT\n");
110 }
```

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.

```
538 {
539
            // ---- sizes
540
            long long int numBlocks,
                                                                                 // num
     ber of blocks
541
           sizeBlock,
                                                                  // size of blocks
542
            sizeHalfBlock;
                                                                  // size of blocks ha
     lved
543
            // ---- indices
            long long int thisBlock,
544
                                                                                 // cur
     rent block
                                                                       // current inde
                index:
      x for first half block
546
            // ---- measured probability
            double totalProbability, renorm;
                                                                                    //
547
```

```
probability (returned) value
548
        // ---- temp variables
          long long int thisTask, numTasks;
549
                                                                     // tas
     k based approach for expose loop with small granularity
550
         // (good for shared memory parallelism)
551
552
          // tests
          // -----//
554
555
          assert (measureQubit >= 0 && measureQubit < numQubits);</pre>
556
557
558
          //
             dimensions
559
560
         // ----- //
state vector elements to sum,
          sizeHalfBlock = 1LL << (measureQubit);</pre>
                                                                // number of
      // and then the number to skip \dot{}
563
          sizeBlock = 2LL * sizeHalfBlock;
                                                                 // size of
     blocks (pairs of measure and skip entries)
564
565
          //
              find probability
566
         // -----
567
          numTasks = 1LL << (numQubits-1);</pre>
568
569
570
         // initialise returned value
571
         totalProbability = 0.0;
572
573
574
          // --- task-based shared-memory parallel implementation
575
          //
576 # ifdef _OPENMP
577 \# pragma omp parallel for \setminus
         shared (numTasks, sizeBlock, sizeHalfBlock, stateVecReal, stateVecImag) \
         private (thisTask,thisBlock,index) \
schedule (static) \
579
580
581
          reduction ( +:totalProbability )
for (thisTask=0; thisTask<numTasks; thisTask++) {
584
                 thisBlock = thisTask / sizeHalfBlock;
                         = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
585
586
587
                 totalProbability += stateVecReal[index]*stateVecReal[index]
588
                  + stateVecImag[index]*stateVecImag[index];
         }
589
590
          renorm=1/sqrt(totalProbability);
591
592
593 # ifdef _OPENMP
594 \# pragma omp parallel for \backslash
595 shared (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
596
         private (thisTask,thisBlock,index) \
         schedule (static) \
          reduction ( +:totalProbability )
598
599 # endif
for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
601
602
                 index = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
603
                 stateVecReal[index]=stateVecReal[index]*renorm;
604
                 stateVecImag[index]=stateVecImag[index]*renorm;
605
                 stateVecReal[index+sizeHalfBlock]=0;
606
607
                 stateVecImag[index+sizeHalfBlock]=0;
608
          }
```

```
609
610
           //{
m SCB} this is a debugging style check. It is probably useful to leave in,
      but it could be parallelised I guess
611
           // double checkTotal=1.;
           // for (index=0; index<2*numTasks; index++) {</pre>
612
           // checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] +
613
     stateVecImag[index]*stateVecImag[index]);
          // }
           // if (checkTotal>0.00001) {printf("Deviation of sum squared amps from un
615
     ity is %.16f\n",checkTotal); exit(1);}
616
617
           return totalProbability;
618 }
```

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().

```
686 {
687
           long long int index;
688
           long long int stateVecSize;
689
           int bit1, bit2, bit3;
           // ------//
691
692
693
           // -----
           assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
694
     2 < numQubits);</pre>
695
696
            stateVecSize = 1LL << numQubits;</pre>
           double probOfFilter=0;
698
699 # ifdef _OPENMP
700 # pragma omp parallel for \
701
          default (none)
           shared (stateVecSize, stateVecReal, stateVecImag ) \
private (index,bit1,bit2,bit3) \
702
703
704
           schedule (static) \
705
           reduction ( +:probOfFilter )
706 # endif
707
            for (index=0; index<stateVecSize; index++) {</pre>
                   bit1 = extractBit (idQubit1, index);
bit2 = extractBit (idQubit2, index);
708
709
                    bit3 = extractBit (idQubit3, index);
710
711
                   if (!(bit1 && bit2 && bit3)) {
712
                           probOfFilter+= stateVecReal[index]*stateVecReal[index] +
     stateVecImag[index]* stateVecImag [index];
713
                 }
714
715
           return probOfFilter;
716 }
```

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);</pre>
506
507
           stateVecSize = 1LL << numQubits;</pre>
508
509 # ifdef _OPENMP
510 # pragma omp parallel for \
511
         default (none)
          shared (stateVecSize, stateVecReal, stateVecImag ) \
private (index,bit1,bit2,bit3,bit4)
512
513
514
           schedule (static)
515 # endif
516
           for (index=0; index<stateVecSize; index++) {</pre>
                  bit1 = extractBit (idQubit1, index);
517
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().

```
FILE *state;
60
           char filename[100];
61
           long long int index;
62
           sprintf(filename, "state_rank_%d.csv", multiQubit.chunkId);
           state = fopen(filename, "w");
63
64
           if (multiQubit.chunkId==0) fprintf(state, "real, imag\n");
65
66
           for(index=0; index<multiQubit.numAmps; index++) {</pre>
                  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:

- \leftrightarrow *multiQubit* object representing the set of qubits to be initialised
- $\leftarrow rotQubit$ qubit to rotate
- \leftarrow *rot1* rotation angle
- ← rot2 rotation angle
- ← *stateVecUp* probability amplitudes in upper half of a block
- ← state VecLo 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().

```
229 {
230
        // ---- temp variables
231
         double stateRealUp, stateRealLo,
                                                            // storage
    for previous state values
2.32
        stateImagUp,stateImagLo;
                                                   // (used in updates)
233
         // ---- temp variables
        long long int thisTask;
                                                        // task based a
   pproach for expose loop with small granularity
2.35
         const long long int numTasks=multiQubit.numAmps;
236
         // (good for shared memory parallelism)
237
2.38
         // -----//
240
         // tests
         // ----- //
241
242
         assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);</pre>
243
2.44
245
         // ------//
246
247
248
249
         // --- task-based shared-memory parallel implementation
         //
250
251
         double rot1Real=rot1.real, rot1Imag=rot1.imag;
        double rot2Real=rot2.real, rot2Imag=rot2.imag;
253
        double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;
254
         double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
         double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.
```

```
imag;
256 \# pragma omp parallel \setminus
          default (none) \
shared (stateVe
257
258
                     (stateVecRealUp, stateVecImagUp, stateVecRealLo, stateVecImagLo, sta
     teVecRealOut, stateVecImagOut, \
259
                            rot1Real,rot1Imag, rot2Real,rot2Imag) \
260
            private (thisTask,stateRealUp,stateImagUp,stateRealLo,stateImagLo)
261
            {
262 # pragma omp for \
263
                    schedule (static)
                    for (thisTask=0; thisTask<numTasks; thisTask++) {</pre>
264
265
                             // store current state vector values in temp variables
266
                             stateRealUp = stateVecRealUp[thisTask];
                             stateImagUp = stateVecImagUp[thisTask];
2.67
268
269
                             stateRealLo = stateVecRealLo[thisTask];
270
                             stateImagLo = stateVecImagLo[thisTask];
271
272
                             // state[indexUp] = alpha * state[indexUp] - conj(beta)
      * state[indexLo]
273
                             stateVecRealOut[thisTask] = rot1Real*stateRealUp - rot1Im
      ag*stateImagUp + rot2Real*stateRealLo + rot2Imag*stateImagLo;
274
                            stateVecImagOut[thisTask] = rot1Real*stateImagUp + rot1Im
      ag*stateRealUp + rot2Real*stateImagLo - rot2Imag*stateRealLo;
275
                    } // end for loop
276
277 } // 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:

- \leftrightarrow *multiQubit* object representing the set of qubits to be initialised
- $\leftarrow rotQubit$ qubit to rotate
- \leftarrow *alpha* rotation angle
- \leftarrow *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().

```
127 {
        // ---- sizes
128
129
         long long int sizeBlock,
                                                                   // siz
    e of blocks
130
         sizeHalfBlock;
                                                       // size of blocks ha
    lved
          // ---- indices
131
         long long int thisBlock,
                                                                   // cur
    rent block
133
             indexUp, indexLo;
                                                           // current inde
     \boldsymbol{x} and corresponding index in lower half block
134
135
          // ---- temp variables
         double stateRealUp, stateRealLo,
136
                                                              // storage
    for previous state values
137
              stateImagUp,stateImagLo;
                                                               // (used in
     // ---- temp variables
138
139
         long long int thisTask;
                                                           // task based a
    pproach for expose loop with small granularity
140
        const long long int numTasks=multiQubit.numAmps>>1;
141
         // (good for shared memory parallelism)
142
         // ----- //
144
         // tests //
// -----//
145
146
          assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);</pre>
147
148
149
          // ------ //
150
          // dimensions
151
          // ------//
152
         sizeHalfBlock = 1LL << rotQubit;</pre>
153
                                                              // size of
    blocks halved
        sizeBlock = 2LL * sizeHalfBlock;
154
                                                               // size of
    blocks
155
156
          // rotate
158
          // ------ //
159
160
161
162
          // --- task-based shared-memory parallel implementation
163
          //
164
165
         // Can't use multiQubit.stateVec as a private OMP var
166
         double *stateVecReal = multiQubit.stateVec.real;
167
          double *stateVecImag = multiQubit.stateVec.imag;
168
          double alphaImag=alpha.imag, alphaReal=alpha.real;
          double betaImag=beta.imag, betaReal=beta.real;
169
170
171 # ifdef _OPENMP
172 # pragma omp parallel \
default (none) \
173 default (none) \
174 shared (sizeBlock, sizeHalfBlock, stateVecReal, stateVecImag, alphaReal, a
    lphaImag, betaReal,betaImag) \
   private (thisTask,thisBlock ,indexUp,indexLo, stateRealUp,stateImagUp,st
    ateRealLo, stateImagLo)
176 # endif
177
178 # ifdef _OPENMP
179 # pragma omp for \
180
               schedule (static)
181 # endif
                for (thisTask=0; thisTask<numTasks; thisTask++) {</pre>
182
```

```
183
184
                            thisBlock = thisTask / sizeHalfBlock;
185
                            indexUp
                                        = thisBlock*sizeBlock + thisTask%sizeHalfBloc
186
                                        = indexUp + sizeHalfBlock;
187
                            // store current state vector values in temp variables
188
189
                            stateRealUp = stateVecReal[indexUp];
190
                            stateImagUp = stateVecImag[indexUp];
191
192
                            stateRealLo = stateVecReal[indexLo];
193
                            stateImagLo = stateVecImag[indexLo];
194
195
                            // state[indexUp] = alpha * state[indexUp] - conj(beta)
      * state[indexLo]
196
                            stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag
      *stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
197
                            stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag
      *stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;
198
199
                            // state[indexLo] = beta * state[indexUp] + conj(alpha)
      * state[indexLol
                            stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*s
200
      tateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
                            stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*s
2.01
      tateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
202
                    } // end for loop
2.03
            }
204
205 } // 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 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().

```
456 {
457
         long long int index;
458
          long long int stateVecSize;
459
          int bit1, bit2;
461
          // ------//
462
463
464
465
          assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
    2 < numOubits);</pre>
466
467
468
          // -----
          // initialise the state to |0000..0>
469
470
471
472
          \ensuremath{//} dimension of the state vector
473
          stateVecSize = 1LL << numQubits;</pre>
474
475 # ifdef _OPENMP
476 # pragma omp parallel for \
     default (none)
    shared (stateVecSize, stateVecReal, stateVecImag) \
477
         private (index,bit1,bit2)
479
480
          schedule (static)
481 # endif
for (index=0; index<stateVecSize; index++) {
483
                 bit1 = extractBit (idQubit1, index);
                 bit2 = extractBit (idQubit2, index);
484
485
                 if (bit1 && bit2) {
486
                        stateVecReal [index] = - stateVecReal [index];
                        stateVecImag [index] = - stateVecImag [index];
487
488
                 }
489
         }
490 }
```

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().

```
17 {
          long long int numAmps = 1L << numQubits;</pre>
19
          long long int numAmpsPerRank = numAmps/env.numRanks;
2.0
          multiQubit->stateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->
     stateVec.real));
22
          multiQubit->stateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->
     stateVec.imag));
23
       if (env.numRanks>1){
                  multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(mu
24
     ltiQubit->pairStateVec.real));
25
              multiQubit->pairStateVec.imag = malloc(numAmpsPerRank * sizeof(mu
     ltiQubit->pairStateVec.imag));
26
      }
27
```

```
2.8
           if ( (!(multiQubit->stateVec.real) || !(multiQubit->stateVec.imag))
29
                    && numAmpsPerRank ) {
                   printf("Could not allocate memory!");
30
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;
4.3
           multiQubit->numChunks = env.numRanks;
44
45
           initStateVec(multiQubit);
46
           if (env.rank==0) printf("Number of amps per rank is %ld.\n", numAmpsPerRa
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().

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().

```
626 {
           long long int index;
628
           long long int stateVecSize;
629
           int bit1, bit2, bit3;
630
631
632
          // tests
           // ----- //
633
634
           assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
     2 < numQubits);</pre>
635
636
           stateVecSize = 1LL << numQubits;</pre>
637
           double probOfFilter=0;
638
639 # ifdef _OPENMP
640 \# pragma omp parallel for \backslash
641
         default (none)
          shared (stateVecSize, stateVecReal, stateVecImag ) \
private (index,bit1,bit2,bit3) \
643
644
          schedule (static) \
645
           reduction ( +:probOfFilter )
646 # endif
647
           for (index=0; index<stateVecSize; index++) {</pre>
                   bit1 = extractBit (idQubit1, index);
648
649
                   bit2 = extractBit (idQubit2, index);
                   bit3 = extractBit (idQubit3, index);
                   if (!(bit1 && bit2 && bit3)) {
651
652
                          probOfFilter+= stateVecReal[index]*stateVecReal[index] +
     stateVecImag[index] * stateVecImag [index];
653
                  }
654
           if (probOfFilter<1e-16) { printf("Extremely small or negative profOfFilt
655
     er=%.8e; aborting! \n",probOfFilter); exit(1);}
656
          double myNorm=1/sqrt(probOfFilter);
657
658 # ifdef _OPENMP
659 # pragma omp parallel for \
660
          default (none)
661
           shared (stateVecSize, stateVecReal, stateVecImag, myNorm ) \
          private (index,bit1,bit2,bit3)
662
           schedule (static)
663
664 # endif
665
           for (index=0; index<stateVecSize; index++) {</pre>
666
                   bit1 = extractBit (idQubit1, index);
                   bit2 = extractBit (idQubit2, index);
667
668
                   bit3 = extractBit (idQubit3, index);
669
                   if ((bit1 && bit2 && bit3)) {
670
                           stateVecReal[index]=0;
671
                           stateVecImag [index]=0;
672
                   }else{
673
                           stateVecReal[index] *= myNorm;
674
                           stateVecImag[index] *= myNorm;
675
                   }
676
           }
677
          return probOfFilter;
678 }
```

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.

 $References\ Complex Array::imag,\ MultiQubit::num Amps,\ MultiQubit::num Qubits,\ Complex Array::real,\ and\ MultiQubit::state Vec.$

Referenced by findProbabilityOfZero().

```
// ---- measured probability
379
         double totalProbability;
                                                                 // probabil
     ity (returned) value
     // ---- temp variables
         long long int thisTask;
                                                              // task based a
381
     pproach for expose loop with small granularity
         long long int numTasks=multiQubit.numAmps;
383
          // (good for shared memory parallelism)
384
385
386
         // tests
387
          // -----//
          assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);</pre>
388
389
          // find probability
391
          // ----- //
392
393
          // initialise returned value
394
395
          totalProbability = 0.0;
396
397
          // initialise correction for kahan summation
399
          // --- task-based shared-memory parallel implementation
400
401
402
403
          double *stateVecReal = multiQubit.stateVec.real;
          double *stateVecImag = multiQubit.stateVec.imag;
404
405
406 # ifdef _OPENMP
407 \# pragma omp parallel for \backslash
408
    shared (numTasks,stateVecReal,stateVecImag) \
         private
         private (thisTask) \
schedule (static) \
409
410
         reduction ( +:totalProbability )
411
412 # endif
for (thisTask=0; thisTask<numTasks; thisTask++) {
                 // summation -- simple implementation
415
                 totalProbability += stateVecReal[thisTask]*stateVecReal[thisTask]
416
                        + stateVecImag[thisTask] * stateVecImag[thisTask];
417
418
                 // summation -- kahan correction
419
```

```
420
                   y = stateVecReal[thisTask]*stateVecReal[thisTask]
421
                   + stateVecImag[thisTask] * stateVecImag[thisTask] - c;
                   t = totalProbability + y;
422
423
                   c = (t - totalProbability) - y;
424
                   totalProbability = t;
425
426
427
           }
428
429
          return totalProbability;
430 }
```

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.

 $References\ Complex Array::imag,\ MultiQubit::num Amps,\ MultiQubit::num Qubits,\ Complex Array::real,\ and\ MultiQubit::state Vec.$

Referenced by findProbabilityOfZero().

```
291 {
          // ---- sizes
          long long int sizeBlock,
                                                                         // siz
293
     e of blocks
                                                            // size of blocks ha
         sizeHalfBlock;
     lved
          // ---- indices
295
          long long int thisBlock,
                                                                        // cur
296
     rent block
297
             index;
                                                                // current inde
     x for first half block
          // ---- measured probability
double totalProbability;
298
299
                                                                    // probabil
     ity (returned) value
300
          // ---- temp variables
301
          long long int thisTask;
                                                                // task based a
     pproach for expose loop with small granularity
302
         long long int numTasks=multiQubit.numAmps>>1;
303
          // (good for shared memory parallelism)
304
              tests
          //
306
          // -----//
307
308
          assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);</pre>
309
310
311
```

```
312
                dimensions
          // ----- //
313
314
          sizeHalfBlock = 1LL << (measureQubit);
                                                                 // number of
     state vector elements to sum,
      ^{\prime\prime} and then the number to skip
315
          sizeBlock = 2LL * sizeHalfBlock;
                                                                   // size of
316
     blocks (pairs of measure and skip entries)
317
          // -----//
318
319
               find probability
          // -----
320
321
322
          // initialise returned value
323
          totalProbability = 0.0;
324
325
          // initialise correction for kahan summation
326
          printf("sizeHalfBlock=%Ld sizeBlock=%Ld numTasks=%Ld\n",sizeHalfBlock,siz
    eBlock, numTasks);
327
328
          // --- task-based shared-memory parallel implementation
329
330
331
          double *stateVecReal = multiQubit.stateVec.real;
333
          double *stateVecImag = multiQubit.stateVec.imag;
334
335 # ifdef _OPENMP
336 \# pragma omp parallel for \setminus
         shared (numTasks, sizeBlock, sizeHalfBlock, stateVecReal, stateVecImag) \
         private (thisTask,thisBlock,index) \
schedule (static) \
338
339
340
          reduction ( +:totalProbability )
341 # endif
for (thisTask=0; thisTask<numTasks; thisTask++) {
                 thisBlock = thisTask / sizeHalfBlock;
343
344
                          = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
345
                 if (index<0){ printf("ABORTING as index=%Ld with thisBlock = %Ld</pre>
346
     thisTask=%Ld \n", index,thisBlock,thisTask); exit(1);}
347
348
                  // summation -- simple implementation
349
                  totalProbability += stateVecReal[index]*stateVecReal[index]
350
                         + stateVecImag[index]*stateVecImag[index];
351
352
                  // summation -- kahan correction
353
354
                  y = stateVecReal[index]*stateVecReal[index]
355
                  + stateVecImag[index] * stateVecImag[index] - c;
356
                  t = totalProbability + y;
                  c = (t - totalProbability) - y;
357
358
                  totalProbability = t;
359
360
361
363
         return totalProbability;
364 }
```

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:

 \leftrightarrow *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().

```
77 {
78
           long long int stateVecSize;
79
           long long int index;
80
81
           // dimension of the state vector
82
           stateVecSize = multiQubit->numAmps;
83
           if (DEBUG) printf("stateVecSize=%Ld \, now performing init with only one t \,
84
     hread: \n", stateVecSize);
85
           // Can't use multiQubit->stateVec as a private OMP var
86
87
           double *stateVecReal = multiQubit->stateVec.real;
88
           double *stateVecImag = multiQubit->stateVec.imag;
89
90
           // initialise the state to |0000..0000>
91 # ifdef _OPENMP
92 # pragma omp parallel for \
           default (none) \
           shared (stateVecSize, stateVecReal, stateVecImag) \
private (index) \
94
95
96
           schedule (static)
97 # endif
98
           for (index=0; index<stateVecSize; index++) {</pre>
99
                   stateVecReal[index] = 0.0;
100
                    stateVecImag[index] = 0.0;
101
            }
102
103
            if (multiQubit->chunkId==0) {
104
                    // zero state |0000..0000> has probability 1
                    stateVecReal[0] = 1.0;
105
106
                    stateVecImag[0] = 0.0;
107
            }
108
           if (DEBUG) printf("COMPLETED INIT\n");
109
110 }
```

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.

```
538 {
539
            // ---- sizes
540
           long long int numBlocks,
                                                                                 // num
     ber of blocks
                                                                   // size of blocks
           sizeBlock,
542
            sizeHalfBlock;
                                                                   // size of blocks ha
     lved
543
            // ---- indices
                                                                                 // cur
544
            long long int thisBlock,
```

```
rent block
545
             index;
                                                             // current inde
    x for first half block
      // ---- measured probability
546
         double totalProbability, renorm;
   probability (returned) value
548
         // ---- temp variables
         long long int thisTask, numTasks;
                                                                     // tas
    k based approach for expose loop with small granularity
550
         // (good for shared memory parallelism)
551
          // ------ //
552
553
          // tests
          // -----
554
          assert (measureQubit >= 0 && measureQubit < numQubits);</pre>
555
556
557
558
559
          // dimensions
          // ------//
560
         sizeHalfBlock = 1LL << (measureQubit);</pre>
    state vector elements to sum,
562
     // and then the number to skip
         sizeBlock = 2LL * sizeHalfBlock;
563
                                                                // size of
    blocks (pairs of measure and skip entries)
564
        // find probability
566
          // -----
567
568
         numTasks = 1LL << (numOubits-1);</pre>
569
570
          // initialise returned value
571
         totalProbability = 0.0;
572
573
         // --- task-based shared-memory parallel implementation
574
575
          //
576 # ifdef _OPENMP
577 \# pragma omp parallel for \setminus
        shared (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
         private (thisTask,thisBlock,index) \
schedule (static) \
579
581
         reduction ( +:totalProbability )
582 # endif
for (thisTask=0; thisTask<numTasks; thisTask++) {
584
                 thisBlock = thisTask / sizeHalfBlock;
585
                 index = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
586
587
                 totalProbability += stateVecReal[index]*stateVecReal[index]
                  + stateVecImag[index]*stateVecImag[index];
589
590
         renorm=1/sqrt(totalProbability);
591
592
593 # ifdef _OPENMP
594 \# pragma omp parallel for \backslash
595
       shared (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
        private (thisTask,thisBlock,index) \
schedule (static) \
596
597
598
          reduction ( +:totalProbability )
599 # endif
for (thisTask=0; thisTask<numTasks; thisTask++) {
                thisBlock = thisTask / sizeHalfBlock;
601
                 index
602
                        = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
603
                 stateVecReal[index] = stateVecReal[index] * renorm;
```

```
604
                    stateVecImag[index]=stateVecImag[index] *renorm;
605
606
                    stateVecReal[index+sizeHalfBlock]=0;
607
                     stateVecImag[index+sizeHalfBlock]=0;
608
609
            //SCB this is a debugging style check. It is probably useful to leave in,
610
      but it could be parallelised I quess
           // double checkTotal=1.;
// for (index=0; index<2*numTasks; index++) {</pre>
611
612
            //
                checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] +
613
     stateVecImag[index]*stateVecImag[index]);
           // }
// if (checkTotal>0.00001){printf("Deviation of sum squared amps from un
614
615
     ity is %.16f\n", checkTotal); exit(1);}
617
            return totalProbability;
618 }
```

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().

```
686 {
687
           long long int index;
           long long int stateVecSize;
           int bit1, bit2, bit3;
689
690
691
           // tests
692
693
           // --
           assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
694
     2 < numOubits);
           stateVecSize = 1LL << numQubits;</pre>
696
697
           double probOfFilter=0;
698
699 # ifdef _OPENMP
700 # pragma omp parallel for \
         default (none)
shared (stateV
701
702
                    (stateVecSize, stateVecReal, stateVecImag ) \
          private (index,bit1,bit2,bit3)
703
704
          schedule (static) \
705
           reduction ( +:probOfFilter )
706 # endif
707
           for (index=0; index<stateVecSize; index++) {</pre>
708
                   bit1 = extractBit (idQubit1, index);
                   bit2 = extractBit (idQubit2, index);
709
710
                   bit3 = extractBit (idQubit3, index);
711
                   if (!(bit1 && bit2 && bit3)) {
                          probOfFilter+= stateVecReal[index]*stateVecReal[index] +
712
     stateVecImag[index]* stateVecImag [index];
713
      }
714
715
          return probOfFilter;
716 }
```

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 {
           long long int index;
498
499
          long long int stateVecSize;
500
           int bit1, bit2, bit3, bit4;
501
503
           //
                tests
           // -----//
504
           assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
     2 < numQubits);</pre>
506
507
           stateVecSize = 1LL << numQubits;</pre>
508
509 # ifdef _OPENMP
510 \# pragma omp parallel for \setminus
511
        default (none)
          shared (stateVecSize, stateVecReal, stateVecImag ) \
private (index,bit1,bit2,bit3,bit4)
513
           schedule (static)
514
515 # endif
516
           for (index=0; index<stateVecSize; index++) {</pre>
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];
                          stateVecImag [index] = - stateVecImag [index];
523
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.

```
file *state;
for char filename[100];
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename, "w");
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename, "w");
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename[100];
for char filename, "state_rank_%d.csv", multiQubit.chunkId);
for char filename, "state_rank_%d.csv", multiQubit.stateVec.real[index],
for char filename,
```

```
multiQubit.stateVec.imag[index]);
68      }
69      fclose(state);
70 }
```

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
- \leftarrow *rot1* rotation angle
- ← *rot2* rotation angle
- ← state VecUp probability amplitudes in upper half of a block
- ← *stateVecLo* probability amplitudes in lower half of a block
- \rightarrow *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.

 $References\ Complex Array::imag,\ Complex::imag,\ MultiQubit::num Amps,\ MultiQubit::num Qubits,\ Complex Array::real,\ and\ Complex::real.$

Referenced by rotateQubit().

```
229 {
         // ---- temp variables
double stateRealUp, stateRealLo,
2.30
231
                                                             // storage
    for previous state values
2.32
        stateImagUp,stateImagLo;
                                                     // (used in updates)
233
         // ---- temp variables
234
         long long int thisTask;
                                                         // task based a
    pproach for expose loop with small granularity
2.35
         const long long int numTasks=multiQubit.numAmps;
236
237
         // (good for shared memory parallelism)
2.38
          // -----//
239
240
         //
             tests
         // ------//
241
         assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);</pre>
2.4.3
244
245
2.46
247
248
          //
```

```
249
            // --- task-based shared-memory parallel implementation
250
251
            double rot1Real=rot1.real, rot1Imag=rot1.imag;
2.52
            double rot2Real=rot2.real, rot2Imag=rot2.imag;
            \verb|double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;|\\
            double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
2.54
255
            double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.
     imag;
256 \# pragma omp parallel \setminus
           default (none) \
shared (stateVecRealUp, stateVecImagUp, stateVecRealLo, stateVecImagLo, sta
257
258
     teVecRealOut, stateVecImagOut, \
259
                             rot1Real,rot1Imag, rot2Real,rot2Imag) \
2.60
            private (thisTask,stateRealUp,stateImagUp,stateRealLo,stateImagLo)
261
262 \# pragma omp for \
                    schedule (static)
2.63
264
                     for (thisTask=0; thisTask<numTasks; thisTask++) {</pre>
265
                             // store current state vector values in temp variables
266
                             stateRealUp = stateVecRealUp[thisTask];
                             stateImagUp = stateVecImagUp[thisTask];
267
2.68
269
                             stateRealLo = stateVecRealLo[thisTask];
                             stateImagLo = stateVecImagLo[thisTask];
270
2.71
272
                             // state[indexUp] = alpha * state[indexUp] - conj(beta)
      * state[indexLo]
2.73
                             stateVecRealOut[thisTask] = rot1Real*stateRealUp - rot1Im
      ag*stateImagUp + rot2Real*stateRealLo + rot2Imag*stateImagLo;
                            stateVecImagOut[thisTask] = rot1Real*stateImagUp + rot1Im
274
      ag*stateRealUp + rot2Real*stateImagLo - rot2Imag*stateRealLo;
275
                    } // end for loop
2.76
277 } // 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
- $\leftarrow rotQubit$ qubit to rotate
- \leftarrow *alpha* rotation angle
- \leftarrow *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().

```
127 {
         // ---- sizes
128
                                                                 // siz
129
         long long int sizeBlock,
    e of blocks
         sizeHalfBlock;
130
                                                     // size of blocks ha
         // ---- indices
131
         long long int thisBlock,
                                                                // cur
132
    rent block
      indexUp,indexLo;
133
                                                         // current inde
    x and corresponding index in lower half block
134
         // ---- temp variables
135
136
         double stateRealUp, stateRealLo,
                                                             // storage
    for previous state values
137
                                                            // (used in
                stateImagUp,stateImagLo;
         // ---- temp variables
138
                                                         // task based a
139
         long long int thisTask;
    pproach for expose loop with small granularity
140
         const long long int numTasks=multiQubit.numAmps>>1;
141
         // (good for shared memory parallelism)
142
143
144
         // -----//
145
         // tests
         // -----//
146
147
         assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);</pre>
148
149
         // -----//
150
         // dimensions
151
         // -----//
         sizeHalfBlock = 1LL << rotQubit;</pre>
                                                             // size of
153
    blocks halved
154
         sizeBlock
                    = 2LL * sizeHalfBlock;
                                                             // size of
     blocks
155
         // ----- //
157
158
             rotate
159
160
161
         // --- task-based shared-memory parallel implementation
162
163
164
         // Can't use multiQubit.stateVec as a private OMP var
165
         double *stateVecReal = multiQubit.stateVec.real;
         double *stateVecImag = multiQubit.stateVec.imag;
167
168
         double alphaImag=alpha.imag, alphaReal=alpha.real;
169
         double betaImag=beta.imag, betaReal=beta.real;
170
171 # ifdef _OPENMP
172 # pragma omp parallel \
         default (none) \
shared (sizeBlock, sizeHalfBlock, stateVecReal, stateVecImag, alphaReal, a
173
174
    lphaImag, betaReal,betaImag) \
175
        private (thisTask,thisBlock ,indexUp,indexLo, stateRealUp,stateImagUp,st
    ateRealLo, stateImagLo)
176 # endif
177
```

```
178 # ifdef _OPENMP
179 # pragma omp for \
180
                    schedule (static)
181 # endif
                    for (thisTask=0; thisTask<numTasks; thisTask++) {</pre>
182
183
                             thisBlock = thisTask / sizeHalfBlock;
184
                                          = thisBlock*sizeBlock + thisTask%sizeHalfBloc
185
                             indexUp
      k;
186
                             indexLo
                                          = indexUp + sizeHalfBlock;
187
188
                             // store current state vector values in temp variables
                             stateRealUp = stateVecReal[indexUp];
stateImagUp = stateVecImag[indexUp];
189
190
191
192
                             stateRealLo = stateVecReal[indexLo];
193
                             stateImagLo = stateVecImag[indexLo];
194
195
                             // state[indexUp] = alpha * state[indexUp] - conj(beta)
      * state[indexLo]
196
                             stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag
      *stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
197
                             stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag
      *stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;
198
199
                             // state[indexLo] = beta * state[indexUp] + conj(alpha)
      * state[indexLo]
2.00
                             stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*s
      tateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
                             stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*s
201
      tateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
202
                    } // end for loop
203
204
205 } // 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

Definition in file qubits_env_local.c.

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

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++) {</pre>
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.

4.4.2.4 void initQUESTEnv (QUESTEnv * env)

Definition at line 9 of file qubits_env_local.c.

```
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().

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.

 $References\ DEBUG,\ Complex Array::imag,\ MultiQubit::num Amps,\ MultiQubit::num Chunks,\ Complex Array::real,\ and\ MultiQubit::state Vec.$

```
81
                                                        {
           double pTotal=0;
82
83
           double allRankTotals=0;
           long long int index;
           long long int numAmpsPerRank = multiQubit.numAmps;
8.5
86
           for (index=0; index<numAmpsPerRank; index++) {</pre>
                   pTotal+=multiQubit.stateVec.real[index] *multiQubit.stateVec.real[
87
      index];
88
                    pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
      index1:
```

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 }</pre>
```

4.5.3.3 void closeQUESTEnv (QUESTEnv env)

Definition at line 36 of file qubits_env_mpi.c.

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(), half-MatrixBlockFitsInChunk(), isChunkToSkipInFindPZero(), and MultiQubit::numAmps.

```
259 {
260
           double stateProb=0, totalStateProb=0;
261
           int skipValuesWithinRank = halfMatrixBlockFitsInChunk(multiQubit.numAmps,
      measureQubit);
262
          if (skipValuesWithinRank) {
2.63
                   stateProb = findProbabilityOfZeroLocal(multiQubit, measureQubit);
            } else {
                    if (!isChunkToSkipInFindPZero(multiQubit.chunkId, multiQubit.
265
     numAmps, measureQubit)){
266
                            stateProb = findProbabilityOfZeroDistributed(multiQubit,
     measureQubit);
267
                    } else stateProb = 0;
2.68
           MPI_Reduce(&stateProb, &totalStateProb, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_CO
269
     MM_WORLD);
270
           return totalStateProb;
271 }
```

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().

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

Get rotation values for a given chunk.

Parameters:

```
\leftarrow chunkIsUpper 1: chunk is in upper half of block, 0: chunk is in lower half \rightarrow 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 {
                    *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;
1.5
           MPI_Initialized(&initialized);
16
           if (!initialized) {
17
                   MPI_Init(NULL, NULL);
                   MPI_Comm_size(MPI_COMM_WORLD, &numRanks);
18
19
                   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
2.0
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;
2.8
                   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 measureOubit)

Find chunks to skip when calculating probability of qubit being zero. When calculating probability of a bit q being zero, sum up $2^{\wedge}q$ values, then skip $2^{\wedge}q$ 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.

 $Referenced\ by\ find Probability Of Zero().$

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(), halfMatrixBlock-FitsInChunk(), ComplexArray::imag, MultiQubit::numAmps, MultiQubit::pairStateVec, ComplexArray::real, rotateQubitDistributed(), rotateQubitLocal(), and MultiQubit::stateVec.

```
196 {
197
            // 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, rot
     Qubit);
199
            Complex rot1, rot2;
200
            // rank's chunk is in upper half of block
201
202
            int rankIsUpper;
203
            int pairRank; // rank of corresponding chunk
204
205
            // MPI send/receive vars
            int TAG=100;
206
207
            MPI_Status status;
208
            double *stateVecReal, stateVecImag, stateVecRealPair, stateVecImagPair;
209
210
211
212
            if (useLocalDataOnly) {
213
                    // all values required to update state vector lie in this rank
214
                    rotateQubitLocal(multiQubit, rotQubit, alpha, beta);
215
            } else {
216
                    // need to get corresponding chunk of state vector from other ran
217
                    rankIsUpper = chunkIsUpper(multiQubit.chunkId, multiQubit.
      numAmps, rotOubit);
218
                    getRotAngle(rankIsUpper, &rot1, &rot2, alpha, beta);
219
                    pairRank = getChunkPairId(rankIsUpper, multiQubit.chunkId, multiQ
      ubit.numAmps, rotQubit);
220
                    //printf("%d rank has pair rank: %d\n", multiQubit.rank, pairRank
      );
221
                    // get corresponding values from my pair
                    MPI_Sendrecv(multiQubit.stateVec.real, multiQubit.numAmps, MPI_DO
222
      UBLE, pairRank, TAG,
223
                                     multiQubit.pairStateVec.real, multiQubit.
      numAmps, MPI_DOUBLE, pairRank, TAG,
224
                                     MPI_COMM_WORLD, &status);
225
                    //printf("rank: %d err: %d\n", multiQubit.rank, err);
                    MPI_Sendrecv(multiQubit.stateVec.imag, multiQubit.numAmps, MPI_DO
226
      UBLE, pairRank, TAG,
227
                                    multiQubit.pairStateVec.imag, multiQubit.numAmps,
      MPI_DOUBLE, pairRank, TAG,
228
                                    MPI_COMM_WORLD, &status);
229
                    // this rank's values are either in the upper of lower half of th
      e block. send values to rotateQubitDistributed
230
                    // in the correct order
231
                    if (rankIsUpper) {
232
                            rotateQubitDistributed(multiQubit,rotQubit,rot1,rot2,
233
                                     multiQubit.stateVec, //upper
234
                                     multiQubit.pairStateVec, //lower
235
                                     multiQubit.stateVec); //output
236
                    } else {
2.37
                            rotateQubitDistributed(multiQubit,rotQubit,rot1,rot2,
238
                                     multiQubit.pairStateVec, //upper
239
                                    multiQubit.stateVec, //lower
240
                                    multiQubit.stateVec); //output
241
                    }
242
            }
243 }
```

4.5.3.11 void syncQUESTEnv (QUESTEnv env)

Definition at line 32 of file qubits_env_mpi.c.

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.

```
43
                                                        {
44
           double pTotal=0;
45
           long long int index;
           long long int numAmpsPerRank = multiQubit.numAmps;
47
           for (index=0; index<numAmpsPerRank; index++) {</pre>
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.

 $References\ MultiQubit:: chunkId,\ findProbabilityOfZeroDistributed(),\ findProbabilityOfZeroLocal(),\ half-MatrixBlockFitsInChunk(),\ isChunkToSkipInFindPZero(),\ and\ MultiQubit:: numAmps.$

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().

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(), halfMatrixBlock-FitsInChunk(), ComplexArray::imag, MultiQubit::numAmps, MultiQubit::pairStateVec, ComplexArray::real, rotateQubitDistributed(), rotateQubitLocal(), and MultiQubit::stateVec.

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

basicTemplate.c, 6	qubits.c, 15
main, 7	qubits.h, 30
MaxAngles, 7	findProbabilityOfZeroLocal
maxNumQubits, 7	qubits.c, 16
_	-
Pi, 11	qubits.h, 32
REPORT_STATE, 7	getChunkPairId
calcTotalProbability	qubits_env_mpi.c, 46
	getRotAngle
qubits_env_local.c, 42	qubits_env_mpi.c, 46
qubits_env_mpi.c, 44	qubits_env_mpr.c, 46
qubits_env_wrapper.h, 50	halfMatrixBlockFitsInChunk
chunkId	qubits_env_mpi.c, 47
MultiQubit, 4	qubits_cnv_mpr.e, 47
chunkIsUpper	imag
qubits_env_mpi.c, 45	Complex, 2
closeQUESTEnv	Complex Array, 3
qubits_env_local.c, 42	initQUESTEnv
qubits_env_mpi.c, 45	qubits_env_local.c, 42
qubits_env_wrapper.h, 50	
Complex, 1	qubits_env_mpi.c, 47
imag, 2	qubits_env_wrapper.h, 51
real, 2	initStateVec
ComplexArray, 2	qubits.c, 18
imag, 3	qubits.h, 33
real, 3	isChunkToSkipInFindPZero
controlPhaseGate	qubits_env_mpi.c, 48
qubits.c, 12	main
qubits.h, 27	
createMultiQubit	basicTemplate.c, 7
qubits.c, 13	MaxAngles
qubits.h, 28	basicTemplate.c, 7
quotosii, 20	maxNumQubits
DEBUG	basicTemplate.c, 7
qubits.c, 12	measureInZero
qubits_env_mpi.c, 44	qubits.c, 19
destroyMultiQubit	qubits.h, 34
qubits.c, 14	MultiQubit, 3
qubits.h, 29	chunkId, 4
quotis.ii, 29	numAmps, 4
extractBit	numChunks, 4
qubits.c, 14	numQubits, 4
qubits.h, 29	pairStateVec, 4
quotis.ii, 29	stateVec, 5
filterOut111	
qubits.c, 14	numAmps
qubits.h, 29	MultiQubit, 4
findProbabilityOfZero	numChunks
qubits_env_local.c, 42	MultiQubit, 4
qubits_env_mpi.c, 45	numQubits
qubits_env_wrapper.h, 51	MultiQubit, 4
findProbabilityOfZeroDistributed	numRanks
marrouaumtyOrzeroDistributed	QUESTEnv, 5
	,

INDEX 54

pairStateVec	findProbabilityOfZero, 45
MultiQubit, 4	getChunkPairId, 46
Pi	getRotAngle, 46
basicTemplate.c, 11	halfMatrixBlockFitsInChunk, 47
probOfFilterOut111	initQUESTEnv, 47
qubits.c, 21	isChunkToSkipInFindPZero, 48
qubits.h, 36	rotateQubit, 48
	syncQUESTEnv, 49
quadCPhaseGate	qubits_env_wrapper.h, 50
qubits.c, 21	calcTotalProbability, 50
qubits.h, 36	closeQUESTEnv, 50
qubits.c, 11	findProbabilityOfZero, 51
controlPhaseGate, 12	initQUESTEnv, 51
createMultiQubit, 13	rotateQubit, 51
DEBUG, 12	syncQUESTEnv, 52
destroyMultiQubit, 14	QUESTEnv, 5
extractBit, 14	numRanks, 5
filterOut111, 14	rank, 5
findProbabilityOfZeroDistributed, 15	,
findProbabilityOfZeroLocal, 16	rank
initStateVec, 18	QUESTEnv, 5
measureInZero, 19	real
probOfFilterOut111, 21	Complex, 2
quadCPhaseGate, 21	Complex Array, 3
reportState, 22	REPORT_STATE
rotateQubitDistributed, 22	basicTemplate.c, 7
rotateQubitLocal, 24	reportState
qubits.h, 26	qubits.c, 22
controlPhaseGate, 27	qubits.h, 37
createMultiQubit, 28	rotateQubit
destroyMultiQubit, 29	qubits_env_local.c, 43
extractBit, 29	qubits_env_mpi.c, 48
filterOut111, 29	qubits_env_wrapper.h, 51
findProbabilityOfZeroDistributed, 30	rotateQubitDistributed
findProbabilityOfZeroLocal, 32	qubits.c, 22
initStateVec, 33	qubits.h, 37
measureInZero, 34	rotateQubitLocal
probOfFilterOut111, 36	qubits.c, 24
quadCPhaseGate, 36	qubits.h, 39
reportState, 37	quons.n, 39
rotateQubitDistributed, 37	stateVec
rotateQubitDocal, 39	MultiQubit, 5
qubits_env_local.c, 41	syncQUESTEnv
• ·	qubits_env_local.c, 43
calcTotalProbability, 42	qubits_env_mpi.c, 49
closeQUESTEnv, 42	qubits_env_wrapper.h, 52
findProbabilityOfZero, 42	quons_env_wrapper.n, 32
initQUESTEnv, 42	
rotateQubit, 43	
syncQUESTEnv, 43	
qubits_env_mpi.c, 43	
calcTotalProbability, 44	
chunkIsUpper, 45	
closeQUESTEnv, 45	
DEBUG, 44	