

Distributed QUEST

Generated by Doxygen 1.6.1

Mon Apr 3 12:28:04 2017

Contents

| | | |
|----------|---|----------|
| 1 | Data Structure Index | 1 |
| 1.1 | Data Structures | 1 |
| 2 | File Index | 1 |
| 2.1 | File List | 1 |
| 3 | Data Structure Documentation | 2 |
| 3.1 | Complex Struct Reference | 2 |
| 3.1.1 | Detailed Description | 2 |
| 3.1.2 | Field Documentation | 2 |
| 3.2 | ComplexArray Struct Reference | 2 |
| 3.2.1 | Detailed Description | 3 |
| 3.2.2 | Field Documentation | 3 |
| 3.3 | MultiQubit Struct Reference | 3 |
| 3.3.1 | Detailed Description | 4 |
| 3.3.2 | Field Documentation | 4 |
| 3.4 | QUESTEnv Struct Reference | 5 |
| 3.4.1 | Detailed Description | 5 |
| 3.4.2 | Field Documentation | 5 |
| 4 | File Documentation | 6 |
| 4.1 | basicTemplate.c File Reference | 6 |
| 4.1.1 | Detailed Description | 7 |
| 4.1.2 | Define Documentation | 7 |
| 4.1.3 | Function Documentation | 7 |
| 4.1.4 | Variable Documentation | 11 |
| 4.2 | qubits.c File Reference | 11 |
| 4.2.1 | Detailed Description | 12 |
| 4.2.2 | Define Documentation | 12 |
| 4.2.3 | Function Documentation | 12 |
| 4.3 | qubits.h File Reference | 26 |
| 4.3.1 | Detailed Description | 27 |
| 4.3.2 | Function Documentation | 27 |
| 4.4 | qubits_env_local.c File Reference | 41 |
| 4.4.1 | Detailed Description | 41 |
| 4.4.2 | Function Documentation | 42 |

| | | |
|-------|---|----|
| 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 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

| | | |
|------------------------------|---|---|
| Complex | (Represents one complex number) | 2 |
| ComplexArray | (Represents an array of complex numbers grouped into an array of real components and an array of coressponding complex components) | 2 |
| MultiQubit | (Represents a system of qubits) | 3 |
| QUESTEnv | (Information about the environment the program is running in) | 5 |

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | | |
|--------------------------------------|---|----|
| basicTemplate.c | (Basic template for using the QUEST library) | 6 |
| qubits.c | (The core of the QUEST Library) | 11 |
| qubits.h | (Structs and specifications for functions that can be used from any environment (local, MPI)) | 26 |
| qubits_env_local.c | (An implementation of the API in qubits_env_wrapper.h for a local (non-MPI) environment) | 41 |
| qubits_env_mpi.c | (An implementation of the API in qubits_env_wrapper.h for an MPI environment) | 43 |
| qubits_env_wrapper.h | (Specifications for QUEST library functions whose implementation depends on environment (local, MPI)) | 50 |

3 Data Structure Documentation

3.1 Complex Struct Reference

Represents one complex number.

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

Data Fields

- double [real](#)
- double [imag](#)

3.1.1 Detailed Description

Represents one complex number.

Definition at line 15 of file qubits.h.

3.1.2 Field Documentation

3.1.2.1 double Complex::imag

Definition at line 18 of file qubits.h.

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

3.1.2.2 double Complex::real

Definition at line 17 of file qubits.h.

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

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

- [qubits.h](#)

3.2 ComplexArray Struct Reference

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

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

Data Fields

- double * [real](#)
- double * [imag](#)

3.2.1 Detailed Description

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

Definition at line 7 of file qubits.h.

3.2.2 Field Documentation

3.2.2.1 `double* ComplexArray::imag`

Definition at line 10 of file qubits.h.

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

3.2.2.2 `double* ComplexArray::real`

Definition at line 9 of file qubits.h.

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

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

- [qubits.h](#)

3.3 MultiQubit Struct Reference

Represents a system of qubits.

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

Data Fields

- [ComplexArray stateVec](#)
Probability 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 `rotateQubit()`.

3.3.2.2 `long long int MultiQubit::numAmps`

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

Definition at line 34 of file qubits.h.

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

3.3.2.3 `int MultiQubit::numChunks`

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

Definition at line 38 of file qubits.h.

Referenced by `calcTotalProbability()`, and `createMultiQubit()`.

3.3.2.4 `int MultiQubit::numQubits`

Number of qubits in the state.

Definition at line 31 of file qubits.h.

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

3.3.2.5 ComplexArray MultiQubit::pairStateVec

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

Definition at line 29 of file `qubits.h`.

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

3.3.2.6 ComplexArray MultiQubit::stateVec

Probablilty amplitudes for the multi qubit state.

Definition at line 27 of file `qubits.h`.

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

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

- [qubits.h](#)

3.4 QUESTEnv Struct Reference

Information about the environment the program is running in.

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

Data Fields

- `int rank`
- `int numRanks`

3.4.1 Detailed Description

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

Definition at line 44 of file `qubits.h`.

3.4.2 Field Documentation

3.4.2.1 int QUESTEnv::numRanks

Definition at line 47 of file `qubits.h`.

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

3.4.2.2 int QUESTEnv::rank

Definition at line 46 of file qubits.h.

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

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

- [qubits.h](#)

4 File Documentation

4.1 basicTemplate.c File Reference

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

`#include <stdlib.h>`

`#include <time.h>`

`#include <math.h>`

`#include <unistd.h>`

`#include <string.h>`

`#include <omp.h>`

`#include "QUEST/qubits.h"`

`#include "QUEST/qubits_env_wrapper.h"`

Defines

- `#define` [MaxAngles](#) 10
Max number of angles used to define qubit rotation.
- `#define` [maxNumQubits](#) 40
Max number of qubits in the system.
- `#define` [REPORT_STATE](#) 1
1: print end qubit state to file, 0: don't print

Functions

- `int` [main](#) (int nargs, char **varg)

Variables

- `const long double` [Pi](#) = 3.14159265358979323846264338327950288419716939937510

4.1.1 Detailed Description

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

Definition in file [basicTemplate.c](#).

4.1.2 Define Documentation

4.1.2.1 #define MaxAngles 10

Max number of angles used to define qubit rotation.

Definition at line 19 of file basicTemplate.c.

Referenced by main().

4.1.2.2 #define maxNumQubits 40

Max number of qubits in the system.

Definition at line 21 of file basicTemplate.c.

Referenced by main().

4.1.2.3 #define REPORT_STATE 1

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

Definition at line 23 of file basicTemplate.c.

Referenced by main().

4.1.3 Function Documentation

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

Definition at line 31 of file basicTemplate.c.

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

```

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

```

```

38     // These two lines will automatically set up the environment (multinode,
39     // openMP only etc)
40     QUESTEnv env;
41     initQUESTEnv(&env);
42
43     // model vars
44     int numQubits;
45     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) {
52             printf(" *** error: argument %d out of range (1 -- %d)\n"
, numQubits,maxNumQubits);
53             exit (EXIT_FAILURE);
54         }
55     } else {
56         printf(" *** error: too few arguments, number of qubits expected\
n");
57         exit (EXIT_FAILURE);
58     }
59
60     // Reporting
61     numAmps = 1L << numQubits;
62     if (env.rank==0){
63         printf("Demo of single qubit rotations.\n");
64         printf("Number of qubits is %d.\n", numQubits);
65         printf("Number of amps is %ld.\n", numAmps);
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>
75     initStateVec (&multiQubit);
76
77
78     //
79     // ===== ROTATIONS
80     //
81
82     // INITIALISE QUBIT ROTATION
83     // Edit these lines to change rotation angle
84     double angl,ang2,ang3;
85     Complex alpha, beta;
86
87     // define rotation angles
88     double angles[MaxAngles][3] = {
89         { 1.2320,  0.4230, -0.6523},
90         { 2.1213,  0.0000,  3.6520},
91         {-3.1213,  5.0230,  0.1230},
92         { 5.2341, -3.1001, -1.2340},
93         {-0.1234, -0.9876,  4.1234}
94     };
95     int numAngles=5,iAngle;
96
97     // rotate
98     angl = angles[0][0];
99     ang2 = angles[0][1];
100     ang3 = angles[0][2];
101

```

```

102     alpha.real = cos(ang1) * cos(ang2);
103     alpha.imag = cos(ang1) * sin(ang2);
104     beta.real  = sin(ang1) * cos(ang3);
105     beta.imag  = sin(ang1) * sin(ang3);
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++) {
112         // do rotation of each qubit
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
123     if (REPORT_STATE){
124         reportState(multiQubit);
125     }
126
127
128     //
129     // ===== perform a measurement
130     //
131     int measureQubit;
132     double stateProb,randProb;
133     /* // keep time */
134     /* wtime_start = system_timer (); */
135
136     // measure
137 /*
138     for (measureQubit=0; measureQubit<numQubits; measureQubit++) {
139         //for (measureQubit=0; measureQubit<l; measureQubit++) {
140             syncQUESTEnv(env);
141             wtime_start = system_timer ();
142             stateProb = findProbabilityOfZero (env.rank, numAmpsPerRank, numQ
ubits, measureQubit, stateVecReal,stateVecImag);
143             syncQUESTEnv(env);
144             wtime_stop = system_timer ();
145             if (env.rank==0) printf("    probability of 0 for qubit %d = %.14f
\n", measureQubit, stateProb);
146             if (env.rank==0) printf(" measurement qubit %d: elapsed time = %f
[s]\n", measureQubit, wtime_stop - wtime_start);
147         }
148 */
149     /* // keep time */
150     /* wtime_stop = system_timer (); */
151
152     /* // ----- timing report */
153     /* printf(" measurement: elapsed time = %f [s]\n", wtime_stop - wtime_sta
rt); */
154
155
156     //
157     // ===== two qubit phase gate
158     //
159     /* // keep time */
160     /* wtime_start = system_timer (); */
161 /*
162     // two qubit phase gate
163     if (numQubits >= 7) {

```

```

164         wtime_start = system_timer ();
165         controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 0, 2, stateVecReal, stateVecImag);
166         wtime_stop = system_timer ();
167         printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
168         wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 1, 3, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
169         wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 2, 4, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
170         wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 3, 5, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
171         wtime_start = system_timer (); controlPhaseGate (env.rank, numAmpsPerRank, numQubits, 4, 6, stateVecReal, stateVecImag); wtime_stop = system_timer (); printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start);
172     }
173 */
174     totalProbability = calcTotalProbability(multiQubit);
175     if (env.rank==0) printf("VERIFICATION: total probability=%.14f\n", totalProbability);
176 /*
177 if (env.rank==0){
178     printf("\n\nIn rank %d, the following is the final state after rotations.
179     \n\n",env.rank);
180     printf("codeOutput=[\n");
181     for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",stateVecReal[index],stateVecImag[index]);
182     printf("];\n\n");
183 }
184 syncQUESTEnv(env);
185 if (env.rank==1){
186     printf("\n\nIn rank %d, the following is the final state after rotations.
187     \n\n",env.rank);
188     printf("codeOutput=[\n");
189     for(index=0; index<=numAmpsPerRank-1; index++) printf("%.8f %.8f\n",stateVecReal[index],stateVecImag[index]);
190     printf("];\n\n");
191 }
192 syncQUESTEnv(env);
193 */
194     /* // keep time */
195     /* wtime_stop = system_timer (); */
196     /* // ---- timing report */
197     /* printf(" two qubit phase gate: elapsed time = %f [s]\n", wtime_stop - wtime_start); */
198
199
200     //
201     // ===== CLEANUP
202     //
203
204     // free memory
205
206     // REQUIRED ONCE PER MULTIQUBIT OBJECT
207     // When all operations on a set of qubits are completed, destroy the object
208     ct
209     destroyMultiQubit(multiQubit, env);

```

```

210
211         // ALWAYS REQUIRED ONCE AT END OF PROGRAM:
212         // These two lines will perform any necessary cleanup of the environment
(multinode,
213         // openMP only etc)
214         closeQUESTEnv(env);
215
216         return EXIT_SUCCESS;
217 }

```

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: $|000\dots 00\rangle$.
- void [rotateQubitLocal](#) ([MultiQubit](#) multiQubit, const int rotQubit, [Complex](#) alpha, [Complex](#) beta)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments.
- void [rotateQubitDistributed](#) ([MultiQubit](#) multiQubit, const int rotQubit, [Complex](#) rot1, [Complex](#) rot2, [ComplexArray](#) stateVecUp, [ComplexArray](#) stateVecLo, [ComplexArray](#) stateVecOut)

Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored seperately.
- double [findProbabilityOfZeroLocal](#) ([MultiQubit](#) multiQubit, const int measureQubit)

Measure the probability of a specified qubit being in the zero state.
- double [findProbabilityOfZeroDistributed](#) ([MultiQubit](#) multiQubit, const int measureQubit)

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

- int [extractBit](#) (const int locationOfBitFromRight, const long long int theEncodedNumber)
- void [controlPhaseGate](#) (const int numQubits, const int idQubit1, const int idQubit2, double *restrict stateVecReal, double *restrict stateVecImag)

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

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

4.2.1 Detailed Description

The core of the QUEST Library.

Definition in file [qubits.c](#).

4.2.2 Define Documentation

4.2.2.1 #define DEBUG 0

Definition at line 11 of file qubits.c.

Referenced by [calcTotalProbability\(\)](#), [initQUESTEnv\(\)](#), and [initStateVec\(\)](#).

4.2.3 Function Documentation

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

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

Definition at line 454 of file qubits.c.

References [extractBit\(\)](#).

```

456 {
457     long long int index;
458     long long int stateVecSize;
459     int bit1, bit2;
460
461     // ----- //
462     //           tests           //
```

```

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

```

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

Definition at line 16 of file qubits.c.

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

Referenced by main().

```

17 {
18     long long int numAmps = 1L << numQubits;
19     long long int numAmpsPerRank = numAmps/env.numRanks;
20
21     multiQubit->stateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->
stateVec.real));
22     multiQubit->stateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->
stateVec.imag));
23     if (env.numRanks>1){
24         multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(mu
ltiQubit->pairStateVec.real));
25         multiQubit->pairStateVec.imag = malloc(numAmpsPerRank * sizeof(mu
ltiQubit->pairStateVec.imag));
26     }
27
28     if ( (!(multiQubit->stateVec.real) || !(multiQubit->stateVec.imag))
&& numAmpsPerRank ) {
29         printf("Could not allocate memory!");
30         exit (EXIT_FAILURE);
31     }
32
33
34     if ( env.numRanks>1 && (!(multiQubit->pairStateVec.real) || !(multiQubit-

```

```

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

```

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

Definition at line 49 of file qubits.c.

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

Referenced by main().

```

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

```

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

Definition at line 433 of file qubits.c.

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

```

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

```

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

Definition at line 623 of file qubits.c.

References extractBit().


```

626 {
627     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);
635
636     stateVecSize = 1LL << numQubits;
637     double probOfFilter=0;
638
639 # ifdef _OPENMP
640 # pragma omp parallel for \
641     default (none) \
642     shared (stateVecSize, stateVecReal, stateVecImag) \
643     private (index, bit1, bit2, bit3) \
644     schedule (static) \
645     reduction (+:probOfFilter)
646 # endif
647     for (index=0; index<stateVecSize; index++) {
648         bit1 = extractBit (idQubit1, index);
649         bit2 = extractBit (idQubit2, index);
650         bit3 = extractBit (idQubit3, index);
651         if (!(bit1 && bit2 && bit3)) {
652             probOfFilter+= stateVecReal[index]*stateVecReal[index] +
stateVecImag[index]* stateVecImag [index];
653         }
654     }
655     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 \
660     default (none) \
661     shared (stateVecSize, stateVecReal, stateVecImag, myNorm) \
662     private (index, bit1, bit2, bit3) \
663     schedule (static)
664 # endif
665     for (index=0; index<stateVecSize; index++) {
666         bit1 = extractBit (idQubit1, index);
667         bit2 = extractBit (idQubit2, index);
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 }

```

4.2.3.6 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()`.

```

377 {
378     // ----- measured probability
379     double    totalProbability;                // probabil
ity (returned) value
380     // ----- temp variables
381     long long int thisTask;                    // task based a
pproach for expose loop with small granularity
382     long long int numTasks=multiQubit.numAmps;
383     // (good for shared memory parallelism)
384
385     // -----
386     //          tests
387     // -----
388     assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);
389
390     // -----
391     //          find probability
392     // -----
393
394     // initialise returned value
395     totalProbability = 0.0;
396
397     // initialise correction for kahan summation
398
399     //
400     // --- task-based shared-memory parallel implementation
401     //
402
403     double *stateVecReal = multiQubit.stateVec.real;
404     double *stateVecImag = multiQubit.stateVec.imag;
405
406 # ifdef _OPENMP
407 # pragma omp parallel for \
408     shared    (numTasks,stateVecReal,stateVecImag) \
409     private  (thisTask) \
410     schedule (static) \
411     reduction (+:totalProbability )
412 # endif
413     for (thisTask=0; thisTask<numTasks; thisTask++) {
414         // summation -- simple implementation
415         totalProbability += stateVecReal[thisTask]*stateVecReal[thisTask]
416
417                             + stateVecImag[thisTask]*stateVecImag[thisTask];
418
419         /*
420         // summation -- kahan correction
421         y = stateVecReal[thisTask]*stateVecReal[thisTask]
422         + stateVecImag[thisTask]*stateVecImag[thisTask] - c;
423         t = totalProbability + y;
424         c = (t - totalProbability) - y;

```

```

424             totalProbability = t;
425             */
426
427     }
428
429     return totalProbability;
430 }

```

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

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

Parameters:

- ← *multiQubit* object representing the set of qubits to be initialised
- ← *measureQubit* qubit to measure

Returns:

probability of qubit *measureQubit* being zero

Definition at line 289 of file qubits.c.

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

Referenced by findProbabilityOfZero().

```

291 {
292     // ----- sizes
293     long long int sizeBlock, // siz
e of blocks
294     sizeHalfBlock; // size of blocks ha
lved
295     // ----- indices
296     long long int thisBlock, // cur
rent block
297     index; // current inde
x for first half block
298     // ----- measured probability
299     double totalProbability; // 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
305     // ----- tests
306     // ----- tests
307     // ----- tests
308     assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);
309
310
311     // ----- dimensions
312     // ----- dimensions
313     // ----- dimensions
314     sizeHalfBlock = 1LL << (measureQubit); // number of
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
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         //
329         // --- task-based shared-memory parallel implementation
330         //
331
332         double *stateVecReal = multiQubit.stateVec.real;
333         double *stateVecImag = multiQubit.stateVec.imag;
334
335 # ifdef _OPENMP
336 # pragma omp parallel for \
337         shared      (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
338
339         private     (thisTask,thisBlock,index) \
340         schedule     (static) \
341         reduction    (+:totalProbability )
342 # endif
343         for (thisTask=0; thisTask<numTasks; thisTask++) {
344             thisBlock = thisTask / sizeHalfBlock;
345             index      = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
346
347             if (index<0){ printf("ABORTING as index=%Ld with thisBlock = %Ld
thisTask=%Ld \n", index,thisBlock,thisTask); exit(1);}
348
349             // summation -- simple implementation
350             totalProbability += stateVecReal[index]*stateVecReal[index]
+ stateVecImag[index]*stateVecImag[index];
351
352             /*
353             // summation -- kahan correction
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: $|000\dots00\rangle$.

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;
79     long long int index;
80
81     // dimension of the state vector
82     stateVecSize = multiQubit->numAmps;
83
84     if (DEBUG) printf("stateVecSize=%ld    now performing init with only one t
hread:\n",stateVecSize);
85
86     // Can't use multiQubit->stateVec as a private OMP var
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 \
93     default (none) \
94     shared (stateVecSize, stateVecReal, stateVecImag) \
95     private (index) \
96     schedule (static)
97 # endif
98     for (index=0; index<stateVecSize; index++) {
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
105         stateVecReal[0] = 1.0;
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
544     long long int thisBlock, // cur
rent block
545     index; // current inde
x for first half block
546     // ----- measured probability
547     double totalProbability, renorm; //
```

```

probability (returned) value
548     // ----- temp variables
549     long long int thisTask,numTasks;                                // tas
k based approach for expose loop with small granularity
550     // (good for shared memory parallelism)
551
552     // ----- tests
553     // -----
554     // -----
555     assert (measureQubit >= 0 && measureQubit < numQubits);
556
557
558     // ----- dimensions
559     // -----
560     // -----
561     sizeHalfBlock = 1LL << (measureQubit);                          // number of
state vector elements to sum,
562     // and then the number to skip
563     sizeBlock      = 2LL * sizeHalfBlock;                          // size of
blocks (pairs of measure and skip entries)
564
565     // ----- find probability
566     // -----
567     // -----
568     numTasks = 1LL << (numQubits-1);
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 \
578     shared      (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
579
580     private      (thisTask,thisBlock,index) \
581     schedule      (static) \
582     reduction (+:totalProbability)
583 # endif
584     for (thisTask=0; thisTask<numTasks; thisTask++) {
585         thisBlock = thisTask / sizeHalfBlock;
586         index      = thisBlock*sizeBlock + thisTask*sizeHalfBlock;
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 \
595     shared      (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
596
597     private      (thisTask,thisBlock,index) \
598     schedule      (static) \
599     reduction (+:totalProbability)
600 # endif
601     for (thisTask=0; thisTask<numTasks; thisTask++) {
602         thisBlock = thisTask / sizeHalfBlock;
603         index      = thisBlock*sizeBlock + thisTask*sizeHalfBlock;
604         stateVecReal[index]=stateVecReal[index]*renorm;
605         stateVecImag[index]=stateVecImag[index]*renorm;
606
607         stateVecReal[index+sizeHalfBlock]=0;
608         stateVecImag[index+sizeHalfBlock]=0;
609     }

```

```

609
610     //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.;
612     // for (index=0; index<2*numTasks; index++) {
613     //     checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] +
stateVecImag[index]*stateVecImag[index]);
614     // }
615     // if (checkTotal>0.00001){printf("Deviation of sum squared amps from un
ity is %.16f\n",checkTotal); exit(1);}
616
617     return totalProbability;
618 }

```

4.2.3.10 double probOffFilterOut111 (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;
690
691     // ----- //
692     //           tests                               //
693     // ----- //
694     assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
2 < numQubits);
695
696     stateVecSize = 1LL << numQubits;
697     double probOffFilter=0;
698
699 # ifdef _OPENMP
700 # pragma omp parallel for \
701     default (none) \
702     shared (stateVecSize, stateVecReal, stateVecImag) \
703     private (index, bit1, bit2, bit3) \
704     schedule (static) \
705     reduction (+:probOffFilter)
706 # endif
707     for (index=0; index<stateVecSize; index++) {
708         bit1 = extractBit (idQubit1, index);
709         bit2 = extractBit (idQubit2, index);
710         bit3 = extractBit (idQubit3, index);
711         if (!(bit1 && bit2 && bit3)) {
712             probOffFilter+= stateVecReal[index]*stateVecReal[index] +
stateVecImag[index]* stateVecImag [index];
713         }
714     }
715     return probOffFilter;
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);
506
507     stateVecSize = 1LL << numQubits;
508
509 # ifdef _OPENMP
510 # pragma omp parallel for \
511     default (none) \
512     shared (stateVecSize, stateVecReal, stateVecImag) \
513     private (index, bit1, bit2, bit3, bit4) \
514     schedule (static)
515 # endif
516     for (index=0; index<stateVecSize; index++) {
517         bit1 = extractBit (idQubit1, index);
518         bit2 = extractBit (idQubit2, index);
519         bit3 = extractBit (idQubit3, index);
520         bit4 = extractBit (idQubit4, index);
521         if (bit1 && bit2 && bit3 && bit4) {
522             stateVecReal [index] = - stateVecReal [index];
523             stateVecImag [index] = - stateVecImag [index];
524         }
525     }
526 }
```

4.2.3.12 void reportState (MultiQubit *multiQubit*)

Definition at line 58 of file qubits.c.

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

Referenced by `main()`.

```

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


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

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

Remarks:

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

Parameters:

- ↔ *multiQubit* object representing the set of qubits to be initialised
- ← *rotQubit* qubit to rotate
- ← *rot1* rotation angle
- ← *rot2* rotation angle
- ← *stateVecUp* probability amplitudes in upper half of a block
- ← *stateVecLo* probability amplitudes in lower half of a block
- *stateVecOut* array section to update (will correspond to either the lower or upper half of a block)

Definition at line 224 of file qubits.c.

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

Referenced by rotateQubit().

```

229 {
230     // ----- temp variables
231     double stateRealUp, stateRealLo, // storage
    for previous state values
232     stateImagUp, stateImagLo; // (used in updates)

233     // ----- temp variables
234     long long int thisTask; // task based a
    pproach for expose loop with small granularity
235     const long long int numTasks=multiQubit.numAmps;
236
237     // (good for shared memory parallelism)
238
239     // -----
240     // tests //
241     // ----- //
242     assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);
243
244     // -----
245     // rotate //
246     // ----- //
247
248     //
249     // --- task-based shared-memory parallel implementation
250     //
251     double rot1Real=rot1.real, rot1Imag=rot1.imag;
252     double rot2Real=rot2.real, rot2Imag=rot2.imag;
253     double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;
254     double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
255     double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.

```

```

    imag;
256 # pragma omp parallel \
257     default (none) \
258     shared (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)
264     for (thisTask=0; thisTask<numTasks; thisTask++) {
265         // store current state vector values in temp variables
266         stateRealUp = stateVecRealUp[thisTask];
267         stateImagUp = stateVecImagUp[thisTask];
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.
 $\alpha_{Re} = \cos(\text{angle1}) * \cos(\text{angle2})$

$\alpha_{Im} = \cos(\text{angle1}) * \sin(\text{angle2})$

$\beta_{Re} = \sin(\text{angle1}) * \cos(\text{angle3})$

$\beta_{Im} = \sin(\text{angle1}) * \sin(\text{angle3})$

Remarks:

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

Parameters:

- ↔ *multiQubit* object representing the set of qubits to be initialised
- ← *rotQubit* qubit to rotate
- ← *alpha* rotation angle
- ← *beta* rotation angle

Definition at line 126 of file qubits.c.

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

Referenced by rotateQubit().

```

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

```

```

183
184             thisBlock    = thisTask / sizeHalfBlock;
185             indexUp      = thisBlock*sizeBlock + thisTask%sizeHalfBloc
186     k;
187             indexLo      = indexUp + sizeHalfBlock;
188
189             // store current state vector values in temp variables
190             stateRealUp = stateVecReal[indexUp];
191             stateImagUp = stateVecImag[indexUp];
192
193             stateRealLo = stateVecReal[indexLo];
194             stateImagLo = stateVecImag[indexLo];
195
196             // state[indexUp] = alpha * state[indexUp] - conj(beta)
197     * state[indexLo]
198             stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag
199     *stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
200             stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag
201     *stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;
202
203             // state[indexLo] = beta * state[indexUp] + conj(alpha)
204     * state[indexLo]
205             stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*s
206     tateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
207             stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*s
208     tateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
209     } // end for loop
210 }
211 // 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 corresponding 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: $|000\dots00\rangle$.

- void [rotateQubitLocal](#) ([MultiQubit](#) multiQubit, const int rotQubit, [Complex](#) alpha, [Complex](#) beta)
Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments.
- void [rotateQubitDistributed](#) ([MultiQubit](#) multiQubit, const int rotQubit, [Complex](#) rot1, [Complex](#) rot2, [ComplexArray](#) stateVecUp, [ComplexArray](#) stateVecLo, [ComplexArray](#) stateVecOut)
Rotate a single qubit in the state vector of probability amplitudes, given the angle rotation arguments, and a subset of the state vector with upper and lower block values stored separately.
- 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 [filterOut11](#) (const int numQubits, const int idQubit1, const int idQubit2, const int idQubit3, double *restrict stateVecReal, double *restrict stateVecImag)
- double [probOfFilterOut11](#) (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;
460
461     // ----- //
462     //          tests                               //
463     // ----- //
464
465     assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
2 < numQubits);
466
467
468     // ----- //
469     //          initialise the state to |0000..0>      //
470     // ----- //
471
472     // dimension of the state vector
473     stateVecSize = 1LL << numQubits;
474
475 # ifdef _OPENMP
476 # pragma omp parallel for \
477     default (none) \
478     shared (stateVecSize, stateVecReal, stateVecImag) \
479     private (index, bit1, bit2) \
480     schedule (static)
481 # endif
482     for (index=0; index<stateVecSize; index++) {
483         bit1 = extractBit (idQubit1, index);
484         bit2 = extractBit (idQubit2, index);
485         if (bit1 && bit2) {
486             stateVecReal [index] = - stateVecReal [index];
487             stateVecImag [index] = - stateVecImag [index];
488         }
489     }
490 }

```

4.3.2.2 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 {
18     long long int numAmps = 1L << numQubits;
19     long long int numAmpsPerRank = numAmps/env.numRanks;
20
21     multiQubit->stateVec.real = malloc(numAmpsPerRank * sizeof(multiQubit->
stateVec.real));
22     multiQubit->stateVec.imag = malloc(numAmpsPerRank * sizeof(multiQubit->
stateVec.imag));
23     if (env.numRanks>1){
24         multiQubit->pairStateVec.real = malloc(numAmpsPerRank * sizeof(mu
ltiQubit->pairStateVec.real));
25         multiQubit->pairStateVec.imag = malloc(numAmpsPerRank * sizeof(mu
ltiQubit->pairStateVec.imag));
26     }
27

```

```

28         if ( !(multiQubit->stateVec.real) || !(multiQubit->stateVec.imag) )
29             && numAmpsPerRank ) {
30             printf("Could not allocate memory!");
31             exit (EXIT_FAILURE);
32         }
33
34         if ( env.numRanks>1 && !(multiQubit->pairStateVec.real) || !(multiQubit->
35 >pairStateVec.imag) )
36             && numAmpsPerRank ) {
37             printf("Could not allocate memory!");
38             exit (EXIT_FAILURE);
39         }
40         multiQubit->numQubits = numQubits;
41         multiQubit->numAmps = numAmpsPerRank;
42         multiQubit->chunkId = env.rank;
43         multiQubit->numChunks = env.numRanks;
44
45         initStateVec(multiQubit);
46         if (env.rank==0) printf("Number of amps per rank is %ld.\n", numAmpsPerRa
47 nk);
48     }

```

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

Definition at line 49 of file qubits.c.

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

Referenced by main().

```

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

```

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

Definition at line 433 of file qubits.c.

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

```

434 {
435     return (theEncodedNumber & ( 1LL << locationOfBitFromRight )) >> location
436     OfBitFromRight;
437 }

```

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 {
627     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);
635
636     stateVecSize = 1LL << numQubits;
637     double probOfFilter=0;
638
639 # ifdef _OPENMP
640 # pragma omp parallel for \
641     default (none) \
642     shared (stateVecSize, stateVecReal, stateVecImag) \
643     private (index, bit1, bit2, bit3) \
644     schedule (static) \
645     reduction (+:probOfFilter)
646 # endif
647     for (index=0; index<stateVecSize; index++) {
648         bit1 = extractBit (idQubit1, index);
649         bit2 = extractBit (idQubit2, index);
650         bit3 = extractBit (idQubit3, index);
651         if (!(bit1 && bit2 && bit3)) {
652             probOfFilter+= stateVecReal[index]*stateVecReal[index] +
stateVecImag[index]* stateVecImag [index];
653         }
654     }
655     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 \
660     default (none) \
661     shared (stateVecSize, stateVecReal, stateVecImag, myNorm) \
662     private (index, bit1, bit2, bit3) \
663     schedule (static)
664 # endif
665     for (index=0; index<stateVecSize; index++) {
666         bit1 = extractBit (idQubit1, index);
667         bit2 = extractBit (idQubit2, index);
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 ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```

377 {
378     // ----- measured probability
379     double    totalProbability;                                // probabil
ity (returned) value
380     // ----- temp variables
381     long long int thisTask;                                    // task based a
pproach for expose loop with small granularity
382     long long int numTasks=multiQubit.numAmps;
383     // (good for shared memory parallelism)
384
385     // -----
386     //          tests
387     // -----
388     assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);
389
390     // -----
391     //          find probability
392     // -----
393
394     // initialise returned value
395     totalProbability = 0.0;
396
397     // initialise correction for kahan summation
398
399     //
400     // --- task-based shared-memory parallel implementation
401     //
402
403     double *stateVecReal = multiQubit.stateVec.real;
404     double *stateVecImag = multiQubit.stateVec.imag;
405
406 # ifdef _OPENMP
407 # pragma omp parallel for \
408     shared    (numTasks,stateVecReal,stateVecImag) \
409     private   (thisTask) \
410     schedule  (static) \
411     reduction (+:totalProbability)
412 # endif
413     for (thisTask=0; thisTask<numTasks; thisTask++) {
414         // summation -- simple implementation
415         totalProbability += stateVecReal[thisTask]*stateVecReal[thisTask]
416
417                             + stateVecImag[thisTask]*stateVecImag[thisTask];
418
419         /*
420         // summation -- kahan correction

```

```

420         y = stateVecReal[thisTask]*stateVecReal[thisTask]
421         + stateVecImag[thisTask]*stateVecImag[thisTask] - c;
422         t = totalProbability + y;
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 ComplexArray::imag, MultiQubit::numAmps, MultiQubit::numQubits, ComplexArray::real, and MultiQubit::stateVec.

Referenced by findProbabilityOfZero().

```

291 {
292     // ----- sizes
293     long long int sizeBlock,
294     e of blocks
295     sizeHalfBlock;
296     lved
297     // ----- indices
298     long long int thisBlock,
299     rent block
300     index;
301     x for first half block
302     // ----- measured probability
303     double totalProbability;
304     ity (returned) value
305     // ----- temp variables
306     long long int thisTask;
307     pproach for expose loop with small granularity
308     long long int numTasks=multiQubit.numAmps>>1;
309     // (good for shared memory parallelism)
310
311     // ----- tests
312     assert (measureQubit >= 0 && measureQubit < multiQubit.numQubits);
313
314     // -----

```

```

312         //          dimensions                                //
313         // ----- //
314         sizeHalfBlock = 1LL << (measureQubit);                // number of
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
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         //
329         // --- task-based shared-memory parallel implementation
330         //
331
332         double *stateVecReal = multiQubit.stateVec.real;
333         double *stateVecImag = multiQubit.stateVec.imag;
334
335 # ifdef _OPENMP
336 # pragma omp parallel for \
337         shared      (numTasks, sizeBlock, sizeHalfBlock, stateVecReal, stateVecImag) \
338
339         private  (thisTask, thisBlock, index) \
340         schedule (static) \
341         reduction (+:totalProbability)
342 # endif
343         for (thisTask=0; thisTask<numTasks; thisTask++) {
344             thisBlock = thisTask / sizeHalfBlock;
345             index      = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
346
347             if (index<0){ printf("ABORTING as index=%Ld with thisBlock = %Ld
thisTask=%Ld \n", index, thisBlock, thisTask); exit(1); }
348
349             // summation -- simple implementation
350             totalProbability += stateVecReal[index]*stateVecReal[index]
+ stateVecImag[index]*stateVecImag[index];
351
352             /*
353             // summation -- kahan correction
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 }

```

4.3.2.8 void initStateVec (MultiQubit * multiQubit)

Initialise the state vector of probability amplitudes for a set of qubits to the zero state: $|000\dots00\rangle$.

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;
79     long long int index;
80
81     // dimension of the state vector
82     stateVecSize = multiQubit->numAmps;
83
84     if (DEBUG) printf("stateVecSize=%ld    now performing init with only one t
hread:\n",stateVecSize);
85
86     // Can't use multiQubit->stateVec as a private OMP var
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 \
93     default (none) \
94     shared (stateVecSize, stateVecReal, stateVecImag) \
95     private (index) \
96     schedule (static)
97 # endif
98     for (index=0; index<stateVecSize; index++) {
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
105         stateVecReal[0] = 1.0;
106         stateVecImag[0] = 0.0;
107     }
108
109     if (DEBUG) printf("COMPLETED INIT\n");
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
541     sizeBlock, // size of blocks
542     sizeHalfBlock; // size of blocks ha
lved
543     // ----- indices
544     long long int thisBlock, // cur
```

```

    rent block
545         index;                                // current inde
    x for first half block
546         // ----- measured probability
547         double    totalProbability, renorm;      //
    probability (returned) value
548         // ----- temp variables
549         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         // -----
555         assert (measureQubit >= 0 && measureQubit < numQubits);
556
557
558         // -----
559         //             dimensions
560         // -----
561         sizeHalfBlock = 1LL << (measureQubit);    // number of
    state vector elements to sum,
562         // and then the number to skip
563         sizeBlock      = 2LL * sizeHalfBlock;    // size of
    blocks (pairs of measure and skip entries)
564
565         // -----
566         //             find probability
567         // -----
568         numTasks = 1LL << (numQubits-1);
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 \
578         shared      (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
579
580         private      (thisTask,thisBlock,index) \
581         schedule      (static) \
582         reduction    (+:totalProbability)
582 # endif
583         for (thisTask=0; thisTask<numTasks; thisTask++) {
584             thisBlock = thisTask / sizeHalfBlock;
585             index      = thisBlock*sizeBlock + thisTask%sizeHalfBlock;
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 \
595         shared      (numTasks,sizeBlock,sizeHalfBlock, stateVecReal,stateVecImag) \
596
597         private      (thisTask,thisBlock,index) \
598         schedule      (static) \
599         reduction    (+:totalProbability)
599 # endif
600         for (thisTask=0; thisTask<numTasks; thisTask++) {
601             thisBlock = thisTask / sizeHalfBlock;
602             index      = 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
610         //SCB this is a debugging style check. It is probably useful to leave in,
        but it could be parallelised I guess
        // double checkTotal=1.;
611         // for (index=0; index<2*numTasks; index++) {
612         //     checkTotal=checkTotal-(stateVecReal[index]*stateVecReal[index] +
613         stateVecImag[index]*stateVecImag[index]);
614         // }
615         // if (checkTotal>0.00001){printf("Deviation of sum squared amps from un
        ity is %.16f\n",checkTotal); exit(1);}
616
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;
688     long long int stateVecSize;
689     int bit1, bit2, bit3;
690
691     // ----- //
692     //           tests                               //
693     // ----- //
694     assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
        2 < numQubits);
695
696     stateVecSize = 1LL << numQubits;
697     double probOfFilter=0;
698
699 # ifdef _OPENMP
700 # pragma omp parallel for \
701     default (none) \
702     shared (stateVecSize, stateVecReal, stateVecImag) \
703     private (index, bit1, bit2, bit3) \
704     schedule (static) \
705     reduction (+:probOfFilter)
706 # endif
707     for (index=0; index<stateVecSize; index++) {
708         bit1 = extractBit (idQubit1, index);
709         bit2 = extractBit (idQubit2, index);
710         bit3 = extractBit (idQubit3, index);
711         if (!(bit1 && bit2 && bit3)) {
712             probOfFilter+= stateVecReal[index]*stateVecReal[index] +
        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 {
498     long long int index;
499     long long int stateVecSize;
500     int bit1, bit2, bit3, bit4;
501
502     // ----- //
503     //          tests                      //
504     // ----- //
505     assert (idQubit1 >= 0 && idQubit2 >= 0 && idQubit1 < numQubits && idQubit
2 < numQubits);
506
507     stateVecSize = 1LL << numQubits;
508
509 # ifdef _OPENMP
510 # pragma omp parallel for \
511     default (none) \
512     shared (stateVecSize, stateVecReal, stateVecImag) \
513     private (index, bit1, bit2, bit3, bit4) \
514     schedule (static)
515 # endif
516     for (index=0; index<stateVecSize; index++) {
517         bit1 = extractBit (idQubit1, index);
518         bit2 = extractBit (idQubit2, index);
519         bit3 = extractBit (idQubit3, index);
520         bit4 = extractBit (idQubit4, index);
521         if (bit1 && bit2 && bit3 && bit4) {
522             stateVecReal [index] = - stateVecReal [index];
523             stateVecImag [index] = - stateVecImag [index];
524         }
525     }
526 }
```

4.3.2.12 void reportState (MultiQubit *multiQubit*)

Definition at line 58 of file qubits.c.

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

Referenced by `main()`.

```

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

```

        multiQubit.stateVec.imag[index]);
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
- ← *rot1* rotation angle
- ← *rot2* rotation angle
- ← *stateVecUp* probability amplitudes in upper half of a block
- ← *stateVecLo* probability amplitudes in lower half of a block
- *stateVecOut* array section to update (will correspond to either the lower or upper half of a block)

Definition at line 224 of file qubits.c.

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

Referenced by rotateQubit().

```

229 {
230     // ----- temp variables
231     double    stateRealUp, stateRealLo,                // storage
    for previous state values
232     stateImagUp, stateImagLo;                        // (used in updates)

233     // ----- temp variables
234     long long int thisTask;                            // task based a
    pproach for expose loop with small granularity
235     const long long int numTasks=multiQubit.numAmps;
236
237     // (good for shared memory parallelism)
238
239     // ----- tests
240     // ----- tests
241     // ----- tests
242     assert (rotQubit >= 0 && rotQubit < multiQubit.numQubits);
243
244     // ----- rotate
245     // ----- rotate
246     // ----- rotate
247
248     //
```



```

249         // --- task-based shared-memory parallel implementation
250         //
251         double rot1Real=rot1.real, rot1Imag=rot1.imag;
252         double rot2Real=rot2.real, rot2Imag=rot2.imag;
253         double *stateVecRealUp=stateVecUp.real, *stateVecImagUp=stateVecUp.imag;
254         double *stateVecRealLo=stateVecLo.real, *stateVecImagLo=stateVecLo.imag;
255         double *stateVecRealOut=stateVecOut.real, *stateVecImagOut=stateVecOut.
            imag;
256 # pragma omp parallel \
257     default (none) \
258     shared (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)
264     for (thisTask=0; thisTask<numTasks; thisTask++) {
265         // store current state vector values in temp variables
266         stateRealUp = stateVecRealUp[thisTask];
267         stateImagUp = stateVecImagUp[thisTask];
268
269         stateRealLo = stateVecRealLo[thisTask];
270         stateImagLo = stateVecImagLo[thisTask];
271
272         // state[indexUp] = alpha * state[indexUp] - conj(beta)
273         * state[indexLo]
274         stateVecRealOut[thisTask] = rot1Real*stateRealUp - rot1Im
            ag*stateImagUp + rot2Real*stateRealLo + rot2Imag*stateImagLo;
275         stateVecImagOut[thisTask] = rot1Real*stateImagUp + rot1Im
            ag*stateRealUp + rot2Real*stateImagLo - rot2Imag*stateRealLo;
276     } // end for loop
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.
 $\alpha_{Re} = \cos(\text{angle1}) * \cos(\text{angle2})$

$\alpha_{Im} = \cos(\text{angle1}) * \sin(\text{angle2})$

$\beta_{Re} = \sin(\text{angle1}) * \cos(\text{angle3})$

$\beta_{Im} = \sin(\text{angle1}) * \sin(\text{angle3})$

Remarks:

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

Parameters:

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

← *rotQubit* qubit to rotate

← *alpha* rotation angle

← *beta* rotation angle

Definition at line 126 of file qubits.c.

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

Referenced by rotateQubit().

```

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

```

```

178 # ifdef _OPENMP
179 # pragma omp for \
180           schedule (static)
181 # endif
182           for (thisTask=0; thisTask<numTasks; thisTask++) {
183
184               thisBlock  = thisTask / sizeHalfBlock;
185               indexUp    = thisBlock*sizeBlock + thisTask%sizeHalfBloc
186           k;
187               indexLo    = indexUp + sizeHalfBlock;
188
189               // store current state vector values in temp variables
190               stateRealUp = stateVecReal[indexUp];
191               stateImagUp = stateVecImag[indexUp];
192
193               stateRealLo = stateVecReal[indexLo];
194               stateImagLo = stateVecImag[indexLo];
195
196               // state[indexUp] = alpha * state[indexUp] - conj(beta)
197               * state[indexLo]
198               stateVecReal[indexUp] = alphaReal*stateRealUp - alphaImag
199               *stateImagUp - betaReal*stateRealLo - betaImag*stateImagLo;
200               stateVecImag[indexUp] = alphaReal*stateImagUp + alphaImag
201               *stateRealUp - betaReal*stateImagLo + betaImag*stateRealLo;
202               // state[indexLo] = beta * state[indexUp] + conj(alpha)
203               * state[indexLo]
204               stateVecReal[indexLo] = betaReal*stateRealUp - betaImag*s
205               tateImagUp + alphaReal*stateRealLo + alphaImag*stateImagLo;
206               stateVecImag[indexLo] = betaReal*stateImagUp + betaImag*s
207               tateRealUp + alphaReal*stateImagLo - alphaImag*stateRealLo;
208           } // end for loop
209       }
210   } // end of function definition

```

4.4 qubits_env_local.c File Reference

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

`#include "qubits.h"`

`#include "qubits_env_wrapper.h"`

Functions

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

4.4.1 Detailed Description

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

Definition in file [qubits_env_local.c](#).

4.4.2 Function Documentation

4.4.2.1 double calcTotalProbability (MultiQubit *multiQubit*)

Definition at line 43 of file qubits_env_local.c.

Referenced by main().

```

43                                     {
44         double pTotal=0;
45         long long int index;
46         long long int numAmpsPerRank = multiQubit.numAmps;
47         for (index=0; index<numAmpsPerRank; index++){
48             pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
49             index];
49             pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
50             index];
51         }
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
22         onsistency
22 }
```

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

Definition at line 94 of file qubits_env_local.c.

```

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

4.4.2.4 void initQUESTEnv (QUESTEnv * *env*)

Definition at line 9 of file qubits_env_local.c.

Referenced by main().

```

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

```

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

Definition at line 75 of file qubits_env_local.c.

Referenced by main().

```

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

```

4.4.2.6 void syncQUESTEnv (QUESTEnv *env*)

Definition at line 16 of file qubits_env_local.c.

```

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

```

4.5 qubits_env_mpi.c File Reference

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

`#include <stdlib.h>`

`#include <stdio.h>`

`#include "qubits.h"`

`#include "qubits_env_wrapper.h"`

Defines

- `#define` [DEBUG](#) 0

Functions

- void [initQUESTEnv](#) (QUESTEnv *env)
- void [syncQUESTEnv](#) (QUESTEnv env)
- void [closeQUESTEnv](#) (QUESTEnv env)
- int [isChunkToSkipInFindPZero](#) (int chunkId, int chunkSize, int measureQubit)

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

- double [calcTotalProbability](#) ([MultiQubit](#) multiQubit)
- int [chunkIsUpper](#) (int chunkId, int chunkSize, int rotQubit)
Returns whether a given chunk in position chunkId is in the upper or lower half of a block.
- void [getRotAngle](#) (int chunkIsUpper, [Complex](#) *rot1, [Complex](#) *rot2, [Complex](#) alpha, [Complex](#) beta)
Get rotation values for a given chunk.
- int [getChunkPairId](#) (int chunkIsUpper, int chunkId, int chunkSize, int rotQubit)
get position of corresponding chunk, holding values required to update values in my chunk (with chunkId) when rotating rotQubit.
- int [halfMatrixBlockFitsInChunk](#) (int chunkSize, int rotQubit)
return whether the current qubit rotation will use blocks that fit within a single chunk.
- void [rotateQubit](#) ([MultiQubit](#) multiQubit, const int rotQubit, [Complex](#) alpha, [Complex](#) beta)
- double [findProbabilityOfZero](#) ([MultiQubit](#) multiQubit, const int measureQubit)

4.5.1 Detailed Description

An implementation of the API in [qubits_env_wrapper.h](#) for an MPI environment.

Definition in file [qubits_env_mpi.c](#).

4.5.2 Define Documentation

4.5.2.1 #define DEBUG 0

Definition at line 10 of file [qubits_env_mpi.c](#).

4.5.3 Function Documentation

4.5.3.1 double calcTotalProbability ([MultiQubit](#) multiQubit)

Definition at line 81 of file [qubits_env_mpi.c](#).

References [DEBUG](#), [ComplexArray::imag](#), [MultiQubit::numAmps](#), [MultiQubit::numChunks](#), [ComplexArray::real](#), and [MultiQubit::stateVec](#).

```

81                                     {
82         double pTotal=0;
83         double allRankTotals=0;
84         long long int index;
85         long long int numAmpsPerRank = multiQubit.numAmps;
86         for (index=0; index<numAmpsPerRank; index++){
87             pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
88             index];
89             pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
89             index];

```

```

89         }
90         if (DEBUG) printf("before calc prob. %d\n", multiQubit.numChunks);
91         if (multiQubit.numChunks>1) MPI_Reduce(&pTotal, &allRankTotals, 1, MPI_DO
UBLE, MPI_SUM, 0, MPI_COMM_WORLD);
92         else allRankTotals=pTotal;
93
94         return allRankTotals;
95 }

```

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

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

Parameters:

- ← *chunkId* id of chunk in state vector
- ← *chunkSize* number of amps in chunk
- ← *rotQubit* qubit being rotated

Returns:

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

Definition at line 106 of file qubits_env_mpi.c.

Referenced by rotateQubit().

```

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

```

4.5.3.3 void closeQUESTEnv (QUESTEnv *env*)

Definition at line 36 of file qubits_env_mpi.c.

```

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

```

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

Definition at line 257 of file qubits_env_mpi.c.

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

```

259 {
260     double stateProb=0, totalStateProb=0;
261     int skipValuesWithinRank = halfMatrixBlockFitsInChunk (multiQubit.numAmps,
measureQubit);
262     if (skipValuesWithinRank) {
263         stateProb = findProbabilityOfZeroLocal (multiQubit, measureQubit);

264     } else {
265         if (!isChunkToSkipInFindPZero (multiQubit.chunkId, multiQubit.
numAmps, measureQubit)){
266             stateProb = findProbabilityOfZeroDistributed (multiQubit,
measureQubit);
267         } else stateProb = 0;
268     }
269     MPI_Reduce (&stateProb, &totalStateProb, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_CO
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().

```

149 {
150     long long int sizeHalfBlock = 1LL << (rotQubit);
151     int chunksPerHalfBlock = sizeHalfBlock/chunkSize;
152     if (chunkIsUpper){
153         return chunkId + chunksPerHalfBlock;
154     } else {
155         return chunkId - chunksPerHalfBlock;
156     }
157 }

```

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

Get rotation values for a given chunk.

Parameters:

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

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

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

or

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

← *alpha,beta* initial rotation values

Definition at line 127 of file qubits_env_mpi.c.

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

Referenced by rotateQubit().

```

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

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

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

Parameters:

← *chunkSize* number of amps in chunk

← *rotQubit* qubit being rotated

Returns:

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

Definition at line 167 of file qubits_env_mpi.c.

Referenced by findProbabilityOfZero(), and rotateQubit().

```

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

4.5.3.8 void initQUESTEnv (QUESTEnv * *env*)

Definition at line 12 of file qubits_env_mpi.c.

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

```

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

```

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

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

Parameters:

- ← *chunkId* id of chunk in state vector
- ← *chunkSize* number of amps in chunk
- ← *measureQubit* qubit being measured

Returns:

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

Definition at line 54 of file qubits_env_mpi.c.

Referenced by findProbabilityOfZero().

```

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

```

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

Definition at line 194 of file qubits_env_mpi.c.

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

```

196 {
197     // flag to require memory exchange. 1: an entire block fits on one rank,
198     // 0: at most half a block fits on one rank
199     int useLocalDataOnly = halfMatrixBlockFitsInChunk(multiQubit.numAmps, rot
200     Qubit);
201     Complex rot1, rot2;
202     // rank's chunk is in upper half of block
203     int rankIsUpper;
204     int pairRank; // rank of corresponding chunk
205     // MPI send/receive vars
206     int TAG=100;
207     MPI_Status status;
208
209     double *stateVecReal, stateVecImag, stateVecRealPair, stateVecImagPair;
210
211     if (useLocalDataOnly){
212         // all values required to update state vector lie in this rank
213         rotateQubitLocal(multiQubit, rotQubit, alpha, beta);
214     } else {
215         // need to get corresponding chunk of state vector from other ran
216         k
217         rankIsUpper = chunkIsUpper(multiQubit.chunkId, multiQubit.
218         numAmps, rotQubit);
219         getRotAngle(rankIsUpper, &rot1, &rot2, alpha, beta);
220         pairRank = getChunkPairId(rankIsUpper, multiQubit.chunkId, multiQ
221         ubit.numAmps, rotQubit);
222         //printf("%d rank has pair rank: %d\n", multiQubit.rank, pairRank
223         );
224         // get corresponding values from my pair
225         MPI_Sendrecv(multiQubit.stateVec.real, multiQubit.numAmps, MPI_DO
226         UBLE, pairRank, TAG,
227         multiQubit.pairStateVec.real, multiQubit.
228         numAmps, MPI_DOUBLE, pairRank, TAG,
229         MPI_COMM_WORLD, &status);
230         //printf("rank: %d err: %d\n", multiQubit.rank, err);
231         MPI_Sendrecv(multiQubit.stateVec.imag, multiQubit.numAmps, MPI_DO
232         UBLE, pairRank, TAG,
233         multiQubit.pairStateVec.imag, multiQubit.numAmps,
234         MPI_DOUBLE, pairRank, TAG,
235         MPI_COMM_WORLD, &status);
236         // this rank's values are either in the upper of lower half of th
237         e block. send values to rotateQubitDistributed
238         // in the correct order
239         if (rankIsUpper){
240             rotateQubitDistributed(multiQubit, rotQubit, rot1, rot2,
241             multiQubit.stateVec, //upper
242             multiQubit.pairStateVec, //lower
243             multiQubit.stateVec); //output
244         } else {
245             rotateQubitDistributed(multiQubit, rotQubit, rot1, rot2,
246             multiQubit.pairStateVec, //upper
247             multiQubit.stateVec, //lower
248             multiQubit.stateVec); //output
249         }
250     }
251 }
252 }
253 }
```

4.5.3.11 void syncQUESTEnv (QUESTEnv env)

Definition at line 32 of file qubits_env_mpi.c.

```

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

4.6 qubits_env_wrapper.h File Reference

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

Functions

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

4.6.1 Detailed Description

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

Definition in file [qubits_env_wrapper.h](#).

4.6.2 Function Documentation**4.6.2.1 double calcTotalProbability (MultiQubit multiQubit)**

Definition at line 43 of file qubits_env_local.c.

References [DEBUG](#), [ComplexArray::imag](#), [MultiQubit::numAmps](#), [MultiQubit::numChunks](#), [ComplexArray::real](#), and [MultiQubit::stateVec](#).

Referenced by [main\(\)](#).

```

43                                     {
44     double pTotal=0;
45     long long int index;
46     long long int numAmpsPerRank = multiQubit.numAmps;
47     for (index=0; index<numAmpsPerRank; index++){
48         pTotal+=multiQubit.stateVec.real[index]*multiQubit.stateVec.real[
49     index];
49         pTotal+=multiQubit.stateVec.imag[index]*multiQubit.stateVec.imag[
50     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
           consistency
22     }
```

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

Definition at line 94 of file qubits_env_local.c.

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

```

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

4.6.2.4 void initQUESTEnv (QUESTEnv * *env*)

Definition at line 9 of file qubits_env_local.c.

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

Referenced by main().

```

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

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

Definition at line 75 of file qubits_env_local.c.

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

Referenced by main().

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

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

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