Ket Quantum Programming

Language, Library, and Simulator

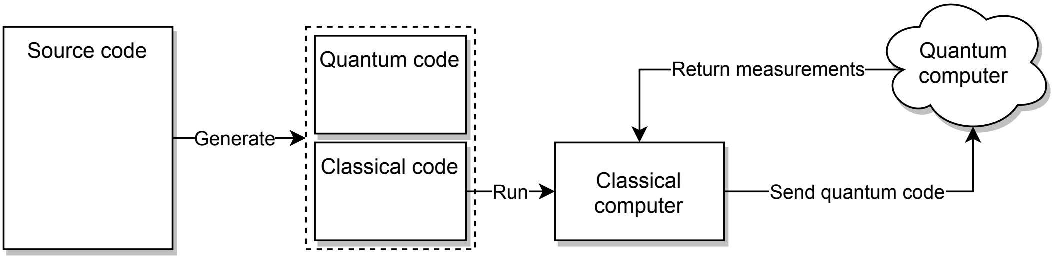
Evandro Chagas Ribeiro da Rosa

UNIVERSIDADE FEDERAL DE SANTA CATARINA

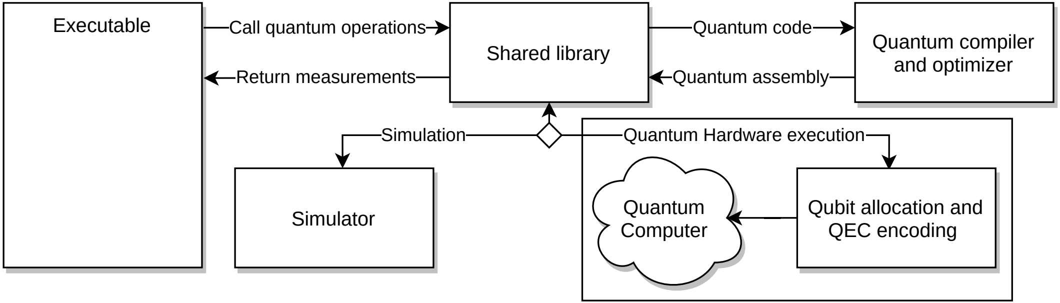
June 19, 2020

Table of Contents

Quantum Programming

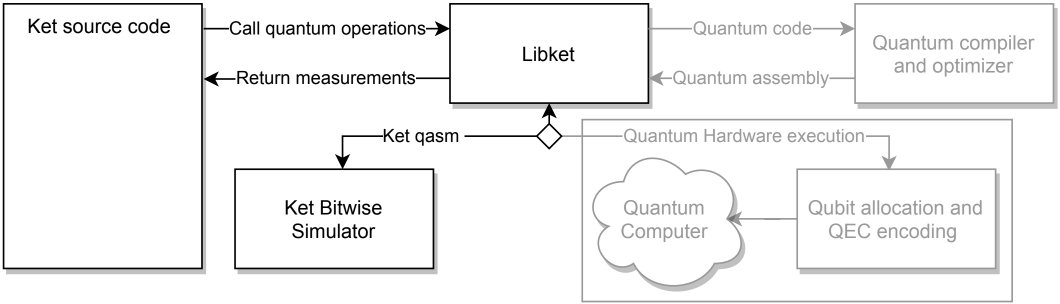


Quantum Programming



arXiv:2006.00131

Ket Quantum Programming



Ket Quantum Programming

- Ket programming language embedded in Python
  - Quantum gates
  - Controlled and Adjunto operations
  - quant and future
  - `https://gitlab.com/quantum-ket/ket`
- Library Ket for C++, C, and Python
  - `https://gitlab.com/quantum-ket/libket`
- Quantum simulator using the Bitwise representation
  - Optimizations
  - Plugins
  - `https://gitlab.com/quantum-ket/kbw`

Ket Programming Language

```
def bell(aux0, aux1):
    q = qalloc(2)
    if aux0 == 1:
        x(q(0))
    if aux1 == 1:
        x(q(1))
    h(q(0))
    ctrl(q(0), x, q(1))
    return q
```

```
def teleport(a):
    b = bell(0, 0)
    ctrl(a, x, b(0))
    h(a)
    m0 = measure(a)
    m1 = measure(b(0))
    if m1 == 1:
        x(b(1))
    if m0 == 1:
        z(b(1))
    return b(1)
```

```
a = qalloc(1)
h(a)
z(a)
y = teleport(a)
h(y)
print(measure(y).get())
```

```
LABEL @entry
  ALLOC   q1
  H       q1
  ALLOC   q2
  CTRL    q1  X  q2
  ALLOC   q0
  H       q0
  Z       q0
  CTRL    q0  X  q1
  H       q0
  MEASURE  q1  c1
  INT  i1  ZE  c1
  INT  i2  1
  INT  i3  i1  ==  i2
  BR   i3  @if.then0  @if.end1
```

```
LABEL @if.then0
  X      q2
  JUMP  @if.end1
LABEL @if.end1
  MEASURE  q0  c0
  INT  i0  ZE  c0
  INT  i4  1
  INT  i5  i0  ==  i4
  BR   i5  @if.then2  @if.end3
LABEL @if.then2
  Z  q2
  JUMP  @if.end3
LABEL @if.end3
  H  q2
  MEASURE  q2  c2
  INT  i6  ZE  c2
```

Adjunt Operation

```
def func(q):
    t(q)
    h(q)

q = qalloc(1)
func(q)
measure(q).get()
```

```
LABEL @entry
  ALLOC   q0
  T   q0
  H   q0
  MEASURE   q0   c0
  INT   i0   ZE   c0
```

```
def func(q):
    t(q)
    h(q)

q = qalloc(1)
adj(func, q)
measure(q).get()
```

```
LABEL @entry
  ALLOC   q0
  H    q0
  TD   q0
  MEASURE   q0   c0
  INT   i0   ZE   c0
```

Ket Runtime Library

- C++, easy to use, template and lambda
- C, for integration with other languages
- Python, a wrapper of the C++ code, used in the Ket language
- quant = array of qubits
- future = int on the QC (measurement result)
- Gates
    - x, y, z, h, s, sd, t, td, u1, u2, u3
    - ctrl
    - adj

Abstract Syntax Tree

```python
a = qalloc(2)
b = qalloc(2)
h(a|b)
mb = measure(b)
ma = measure(a)
c = qalloc(2)
if ma + mb > 2:
    x(c)
mc = measure(c)
print(ma.get())
print(mb.get())
print(mc.get())
```

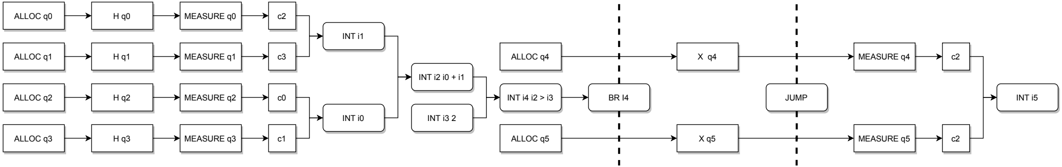
→

```
LABEL @entry
  ALLOC  q4
  ALLOC  q5
  ALLOC  q3
  H  q3
  ALLOC  q2
  H  q2
  ALLOC  q1
  H  q1
  ALLOC  q0
  H  q0
  MEASURE  q0  c2
  MEASURE  q1  c3
  INT  i1  ZE  c2 c3
  MEASURE  q2  c0
  MEASURE  q3  c1
  INT  i0  ZE  c0 c1
  INT  i2  i1  +  i0
  INT  i3  2
  INT  i4  i2  >  i3
  BR  i4  @if.then0  @if.end1
LABEL @if.then0
  X  q5
  X  q4
  JUMP  @if.end1
LABEL @if.end1
  MEASURE  q4  c4
  MEASURE  q5  c5
  INT  i5  ZE  c4 c5
```

Abstract Syntax Tree

- a = [q0, q1]
- b = [q2, q3]
- c = [q4, q5]

- ma = i1
- mb = i0
- mc = i5



Ket Bitwise Simulator

- Bitwise representation arXiv:2004.03560
  - Hashmap to store the quantum state
  - Noise-free simulation
- Keeps qubits separated when they are not entangled
- Bitwise plugins for complex operations

```python
from math import pi, gcd

def qft(q):
    lambd = lambda k : pi*k/2
    for i in range(len(q)):
        for j in range(i):
            ctrl(q(i), u1,
                lambd(i-j), q(j))
        h(q(i))

def period():
    reg1 = qalloc(4)
    h(reg1)
    reg2 = pown(7, reg1, 15)
    qft(reg1)
    return measure(reg1.invert()).get()
```

```python
r = period()
results = [r]
for _ in range(4):
    results.append(period())
    r = gcd(r, results[-1])

r = 2**4/r

print('measurements =', results)
print('r =', r)
p = gcd(int(7**(r/2))+1, 15)
q = gcd(int(7**(r/2))-1, 15)
print(15, '=', p , "x", q)
```

```
LABEL @entry
  ALLOC   q0
  H  q0
  ALLOC   q1
  H  q1
  ALLOC   q2
  H  q2
  ALLOC   q3
  H  q3
  ALLOC   q4
  ALLOC   q5
  ALLOC   q6
  ALLOC   q7
  PLUGIN  ket_pown  q0 q1 q2 q3
  ↪   q4 q5 q6 q7   "15 7"
```

```
H  q0
CTRL  q1  U1(1.57079632679)  q0
CTRL  q2  U1(3.14159265358)  q0
CTRL  q3  U1(4.71238898038)  q0
H  q1
CTRL  q2  U1(1.57079632679)  q1
CTRL  q3  U1(3.14159265358)  q1
H  q2
CTRL  q3  U1(1.57079632679)  q2
H  q3
MEASURE  q3  c0
MEASURE  q2  c1
MEASURE  q1  c2
MEASURE  q0  c3
INT  i0  ZE  c0 c1 c2 c3
```

Plugin

```
#include "ket_bitwise.hpp"
#include <sstream>

using namespace ket;

size_t pown(size_t a, size_t x, size_t n) {
    size_t result = 1 ;
    while (x--) { result *= a; result %= n; }
    return result;
}
```

```
class ket_pown : public bitwise_api {
public:
    void run(map &qbits, size_t size, std::string args) {
        std::stringstream ss{args}; size_t n, a; ss >> n >> a;
        map new_map;
        for (auto &i : qbits) {
            auto val = i.first[0] & ((1ul << size)-1);
            auto x = val >> (size/2); auto y = pown(a, x, n);
            val |= y;
            auto j = i.first; j[0] = val;
            new_map[j] = i.second;
        }
        qbits.swap(new_map);
    }
};
bitwise_plugin(ket_pown);
```

$\frac{1}{\sqrt{2}}\left(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}\right)$

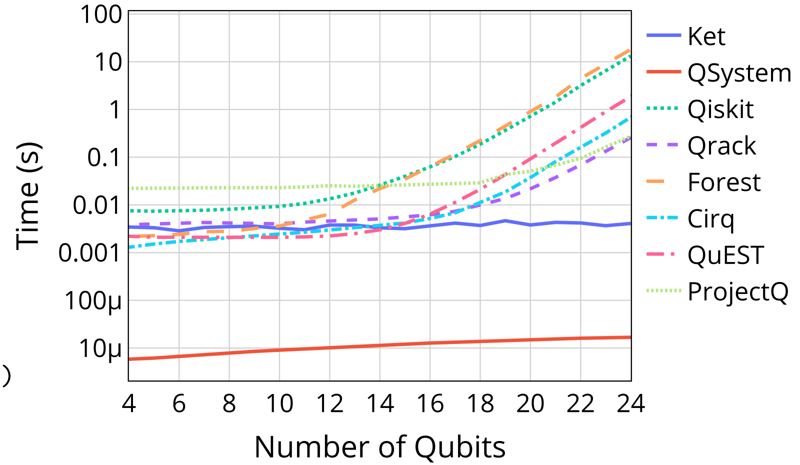
```
a = qalloc(1)

b = qalloc(n-1)

h(a)

ctrl(a, x, b)

measure(a|b).get()
```

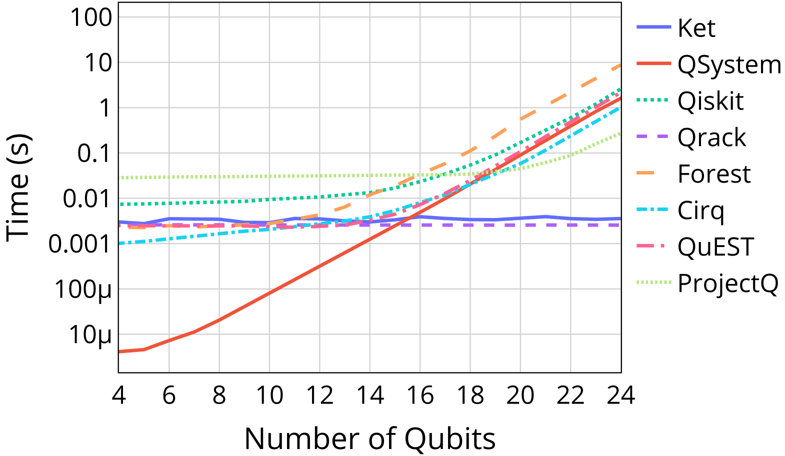


$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle$$

```
a = qalloc(n)

h(a)

measure(a).get()
```



$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \otimes |k\rangle$$

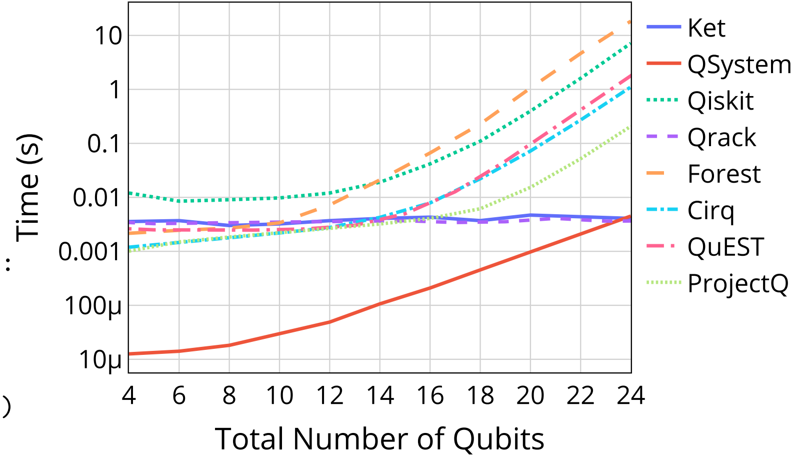
```
a = qalloc(n)

b = qalloc(n)

h(a)

for i in range(n):
    ctrl(a(i),
         x, b(i))

measure(a|b).get()
```



Conclusion

- Quantum programming
  - Dynamic interaction between classical and quantum data
  - Runtime quantum code generation
    - Generic quantum programming
    - Specific quantum code
  - Future
    - Classical statements on the quantum computer
    - Transparent integration with the classical statements
- Quantum circuit simulation
  - Bitwise representation with an adicional Optimization
  - Low cost controlled quantum gates
  - Bitwise plugin for complex quantum operations
- Libket, kqasm, bkw

Limitations and future implementations

- `future.get()`
- Integrate the future with more statements
- Include ket files
- More bitwise plugins
- Quantum debugging*

Installation

From source

Dependencies:

- Conan
- SWIG

```
git clone --recurse-submodules https://gitlab.com/quantum-ket/ket.git
mkdir ket/build
cd ket/build
cmake .. -DPYTHON_PACKAGES=\\
    /home/<user>/.local/lib/python3.<minor versions>/site-packages
sudo make install
```

Usage: `python -m keti <source>.ket`

Installation

Snap

```
sudo snap install ket --beta
```

Usage: `ket <source>.ket`

- Arch Linux
- CentOS
- Debian
- elementary OS
- Fedora
- KDE Neon
- Kubuntu
- Manjaro
- Linux Mint
- openSUSE
- Red Hat Enterprise Linux
- Ubuntu

Snap install instructions at `https://snapcraft.io/ket`

Thank You

Ket Quantum Programming

Language, Library, and Simulator

Evandro Chagas Ribeiro da Rosa

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

June 19, 2020