Quantum Programming Languages and Design

Ed Kuijpers[1]

HBO-ICT Technical Computing

April 25, 2024

[1] e.a.kuijpers@hva.nl

Table of contents

Language and library choice

- Language choice: PYPL PopularitY of Programming Language, TIOBE-index
- Choices: compiled vs interpreted, object oriented, support, open source or commercial, maintenance
- Many software library choices
- IDE's (commercial or open sources)
- Combination of languages (Jupyter notebook)

Top 5 2024

Top 5 prediction for 2024

- Python
- Qiskit (IBM)
- Ocean (Programming suite D-Wave)
- Q#(Microsoft)
- cirq (Google)

Alternatives

- programming the full stack
- Twist Developed at MIT focusing on entanglement
- Quantum programming studio
- Quantum composer from IBM
- ACM Digital Library 532,722 Results for: All: quantum programming language

Programming Analog computers

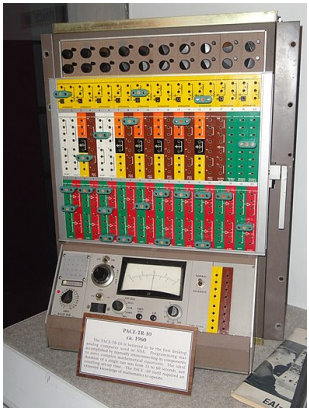


Figure: Example manual hardware programming

Visual programming in instrumentation

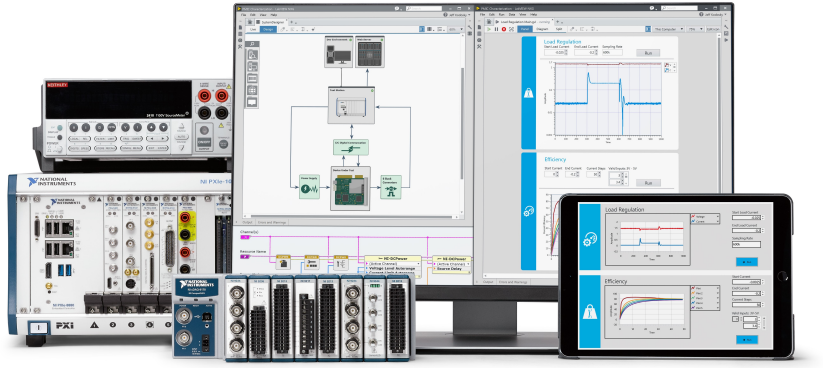


Figure: LabView

Visual programming for Quantum Computing

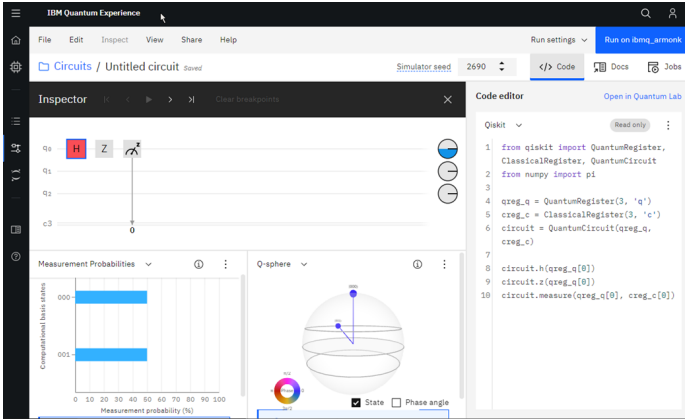


Figure: Quantum composer from IBM

Quantum software development kits, part I

- Qiskit developed by IBM
- Tensorflow quantum: Hybrid quantum-classical machine learning developed by Google
- Quantum Development Kit(QDK) developed by Microsoft using language Q#
- Rigetti: PyQuil, Quile, QVM
- YAO based on the Julia language Luo et al. 2020, open source started with cooperation between China and Canada

Quantum software development kits, part II

- Strawberry Fields: An open-source Python library developed by Xanadu(Canada)
- PennyLane: An open-source Python library developed by Xanadu Quantum Software
- ProjectQ: compilation framework capable of targeting various types of hardware at ETH, Swiss
- TKET,t$|ket\,|\rangle$: quantum programming environment and optimizing compiler developed by Cambridge Quantum

Pennylane demos and stack

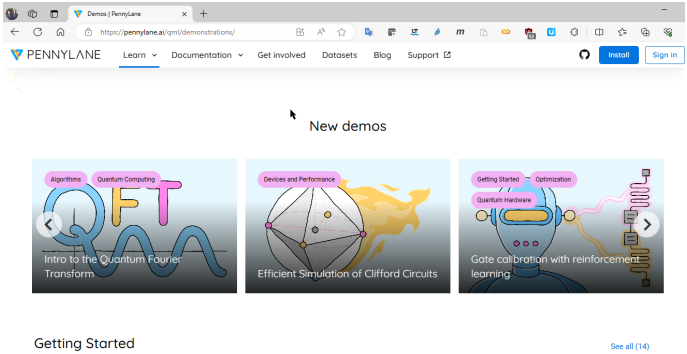
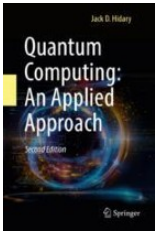


Figure: Pennylane QML

Quantum software programming overviews

- Quantum programming Wikipedia
- Quantiki.org overview simulators
- Book Hidary 2019 updated in second edition Hidary 2021, Github repository
- overview Abhijith et al. 2020



Summary of Quantum Programming Languages I

Year	Language	Reference(s)	Semantics	Host Language	Paradigm
1996	Quantum Lambda Calculi	[188]	Denotational	lambda Calculus	Functional
1998	QCL	[214–217]		C	Imperative
2000	qGCL	[252, 329–331]	Operational	Pascal	Imperative
2003	λ_q	[295, 296]	Operational	Lambda Calculus	Functional
2003	Q language	[31, 32]		C++	Imperative
2004	QFC (QPL)	[256–258]	Denotational	Flowchart syntax (Textual syntax)	Functional
2005	QPAlg	[143, 164]		Process calculus	Other
2005	QML	[10, 11, 115]	Denotational	Syntax similar to Haskell	Functional
2004	CQP	[103–105]	Operational	Process calculus	Other
2005	cQPL	[187]	Denotational		Functional
2006	LanQ	[196–199]	Operational	C	Imperative
2008	NDQJava	[312]		Java	Imperative
2009	Cove	[237]		C#	Imperative
2011	QuECT	[47]		Java	Circuit
2012	Scaffold	[1, 140]		C (C++)	Imperative
2013	QuaFL	[166]		Haskell	Functional
2013	Quipper	[116, 117]	Operational	Haskell	Functional
2013	Chisel-Q	[182]		Scala	Imperative, functional
2014	LIQUi|⟩	[306]	Denotational	F#	Functional
2015	Proto-Quipper	[244, 247]		Haskell	Functional
2016	QASM	[220]		Assembly language	Imperative
2016	FJQuantum	[81]		Feather-weight Java	Imperative
2016	ProjectQ	[124, 279, 285]		Python	Imperative, functional
2016	pyQuil (Quil)	[272]		Python	Imperative

Summary of Quantum Programming Languages II

2017	Forest	[61, 272]		Python	Declarative
2017	OpenQASM	[66]		Assembly language	Imperative
2017	qPCF	[222, 224]		Lambda calculus	Functional
2017	QWIRE	[225]	Denotational	Coq proof assistant	Circuit
2017	cQASM	[149]		Assembly language	Imperative
2017	Qiskit	[4, 242]		Python	Imperative, functional
2018	IQu	[223]		Idealized Algol	Imperative
2018	Strawberry Fields	[150, 151]		Python	Imperative, functional
2018	Blackbird	[150, 151]		Python	Imperative, functional
2018	QuantumOptics.jl	[161]		Julia	Imperative
2018	Cirq	[284]		Python	Imperative, functional
2018	Q#	[282]		C#	Imperative
2018	$Q\|SI\rangle$	[181]		.Net language	Imperative
2020	Silq	[34]	Operational		Imperative, functional
2020	Quingo	[286]		Python	Imperative

* **Year**: The invented year of the language.
* **Language**: The name of the language.
* **Reference(s)**: The main reference paper(s) of the language.
* **Semantics**: The type(s) of semantics for the language the authors described in the reference(s).
* **Host language**: The classical language on (or to) which the language is based (or extended).
* **Paradigm**: We consider each language to belong to three types of paradigms: *imperative language*, *functional language*, and *circuit design language*.

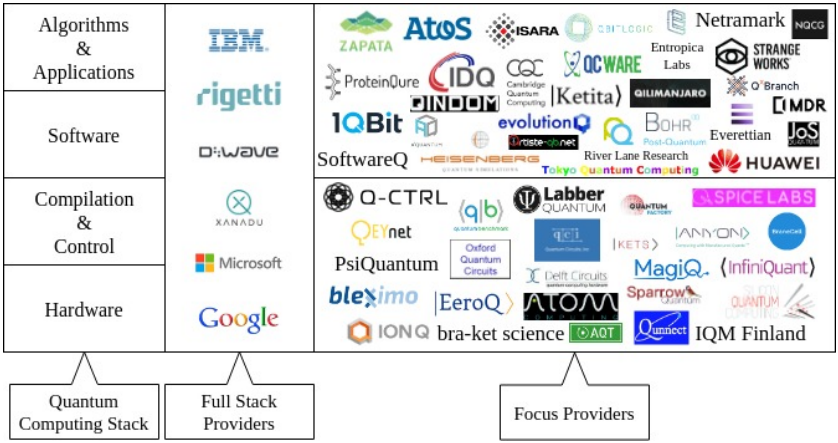
From Historical Summary of Quantum Programming Languages in Zhao 2020

Applications

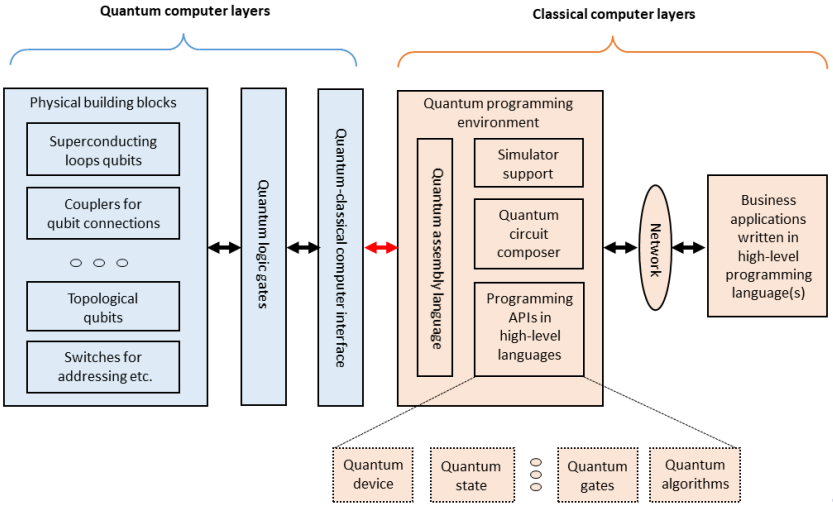
- Quantum Protocol Zoo (many security algorithms)
- Quantum Algorithm Zoo
- Grand Unification of Quantum Algorithms?

Quantum software stacks

Quantum Computing Report , Quantum Computing: Top Players 2021

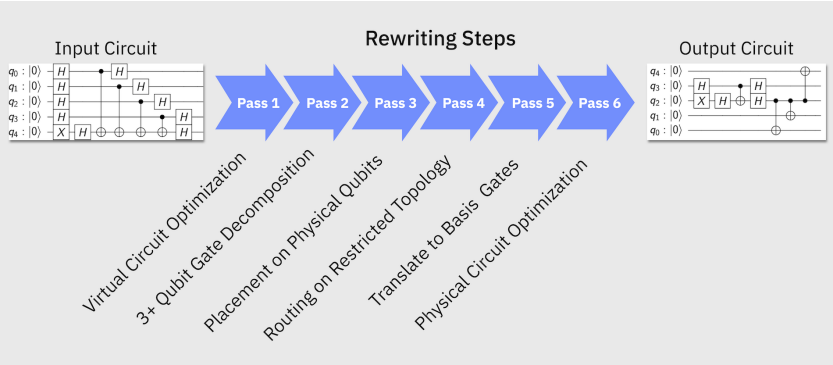


Architecture



Transpiling

Transforming quantum computer programs to executable code

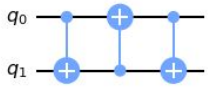


Transpiling SWAP-gate

See: Transpiler in Qiskit

. A SWAP gate is not a native gate on the IBM Q devices, and must be decomposed into three CNOT gates:

```
swap_circ = QuantumCircuit(2)
swap_circ.swap(0, 1)
swap_circ.decompose().draw(output='mpl')
```

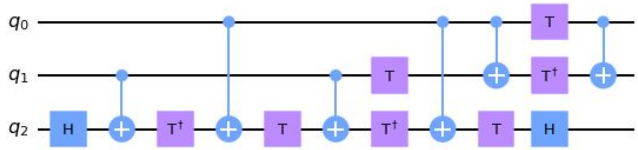


Transpiling Toffoli-gate

See: Transpiler in Qiskit

A Toffoli, or controlled-controlled-not gate (*ccx*), is a three-qubit gate. Given that our basis gate set includes only single- and two-qubit gates, it is obvious that this gate must be decomposed. This decomposition is quite costly:

```
ccx_circ = QuantumCircuit(3)
ccx_circ.ccx(0, 1, 2)
ccx_circ.decompose().draw(output='mpl')
```



Controlling transpiler output

- Transpiler API
- Qiskit tutorial transpiling and passmanager

QuTech Quantum computer visit

Control hardware example at QuTech

Cyrogene cooling example at QuTech

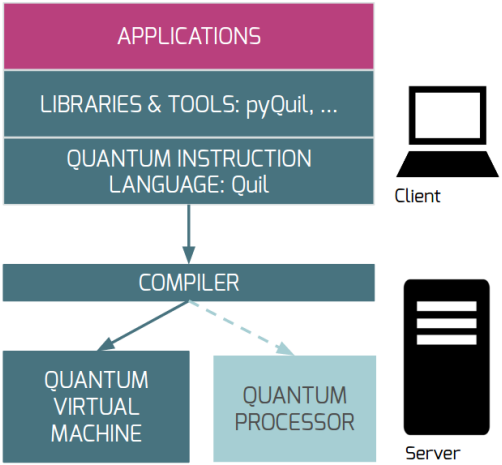
Quantum Computing Instruction Sets

- Quil: Quil (Smith, Curtis, and Zeng 2017)
- cQASM: cQASM(Khammassi et al. 2018)
- OpenQASM: OpenQASM(Cross et al. 2017, McKay et al. 2018)
- Blackbird: Blackbird (Killoran et al. 2019,Bromley et al. 2020)
- QMASM: QMASM(Pakin 2016)

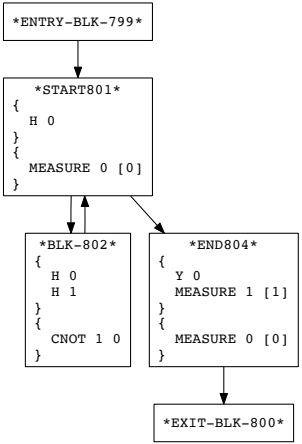
Quil: A Portable Quantum Instruction Language

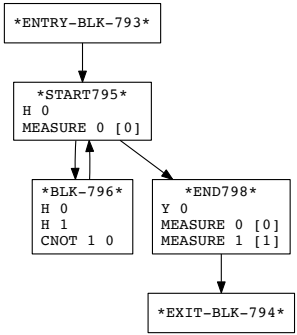
- PyQuil: A library for easily generating Quil programs to be executed using the Rigetti Forest SDK
- Compilers: quilc: The Rigetti optimizing Quil compiler
- Simulators: qvm: The Rigetti high-performance quantum virtual machine
- Use in Rigetti Computing through the Forest quantum programming API.

Outline



Examples Quil

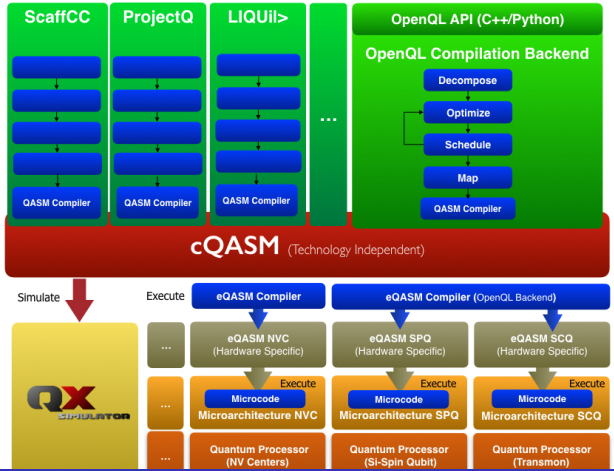




cQASM

Quantum Inspire uses cQASM 1.0. cQASM is used to describe relatively simple circuits for the current generation of quantum computers and is work in progress for more future developments. A cQASM reference describes the language. Quantum Inspire uses cQASM 1.0

cQASM compilation stack



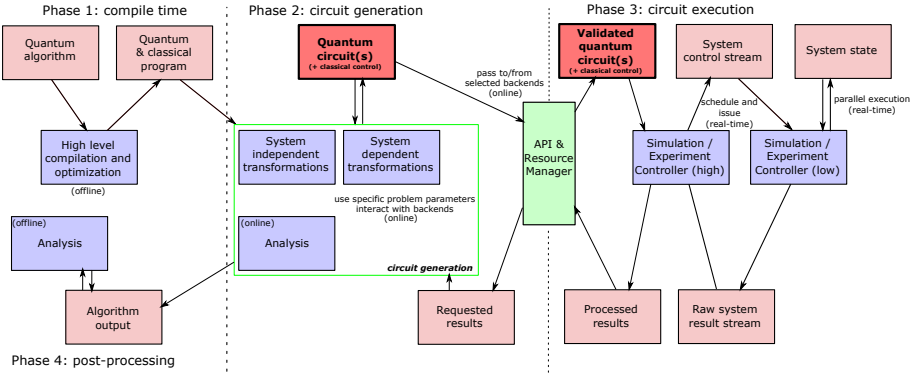
cQASM properties

- developed in Delft
- Programming language
- Qubit gate operations
- Single Gate Multiple-Qubits
- Qubit initialization and measurement
- Display commands

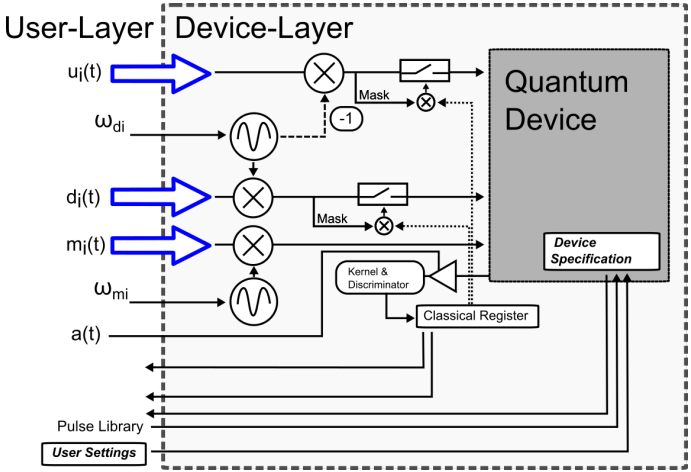
OpenQASM

OpenQASM is an imperative programming language designed for near-term quantum computing algorithms and applications. Its main goal is to serve as an intermediate representation for higher-level compilers to communicate with quantum hardware. Current version is draft 3.0 of the OpenQASM specification.

OpenQASM



OpenQASM


User-Layer
Device-Layer
$u_i(t)$
ω_{di}
$d_i(t)$
$m_i(t)$
ω_{mi}
a(t)
Mask
-1
Mask
Quantum Device
Device Specification
Kernel & Discriminator
Classical Register
Pulse Library
User Settings

OpenQASM properties

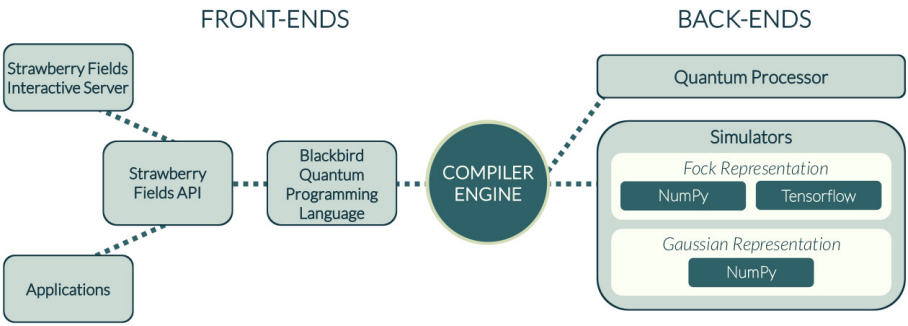
- Used by Qiskit
- Types and Casting
- Gates
- Built-in quantum instructions
- Classical instructions
- Subroutines, directives
- Circuit timing, Pulse-level descriptions of gates and measurements

Blackbird

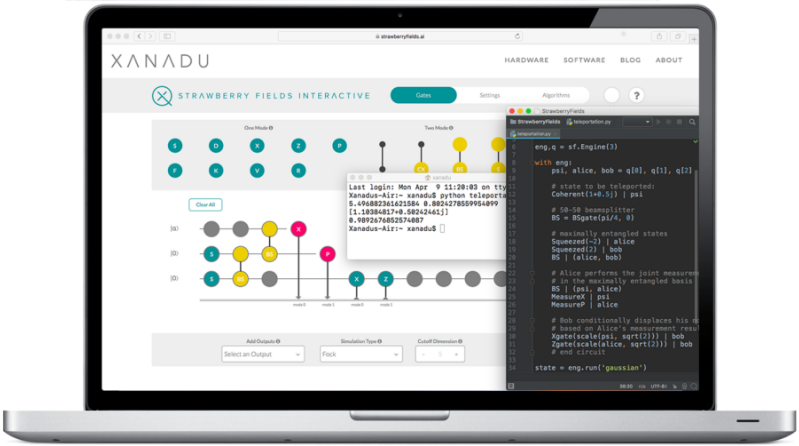
Blackbird is a Quantum Assembly Language for continuous-variable quantum computation, that can be used to program Xanadu's quantum photonics hardware and Strawberry Fields simulator Blackbird (Killoran et al. 2019). The Blackbird repository contains:

- src: The Blackbird grammar specification in enhanced BackusNaur form
- blackbird_python: to develop Blackbird parsers for integration with Python programs
- Blackbird_cpp: libraries and header files needed to develop Blackbird parsers for integration with C++ programs

Blackbird


FRONT-ENDS
BACK-ENDS
Strawberry Fields Interactive Server
Strawberry Fields API
Applications
Blackbird Quantum Programming Language
COMPILER ENGINE
Quantum Processor
Simulators
Fock Representation
NumPy
Tensorflow
Gaussian Representation
NumPy

Blackbird



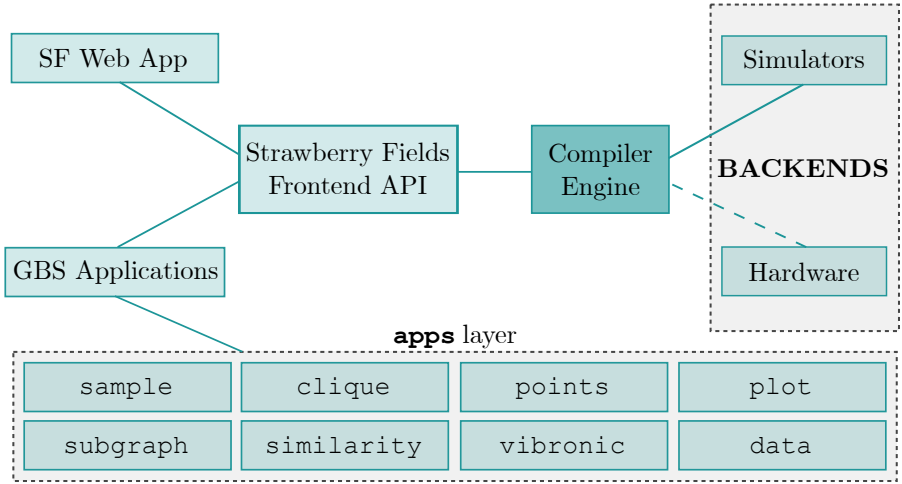
Blackbird features

- Execute photonic quantum algorithms directly on Xanadu's next-generation quantum hardware
- High-level functions for solving practical problems including graph and network optimization, machine learning, and chemistry
- Includes a suite of world-class simulators—based on cutting edge algorithms—to compile and simulate photonic algorithms
- Train and optimize your quantum programs with our end-to-end differentiable TensorFlow backend
- Powers the Strawberry Fields Interactive web app, which allows anyone to run a quantum computing simulation via drag and drop

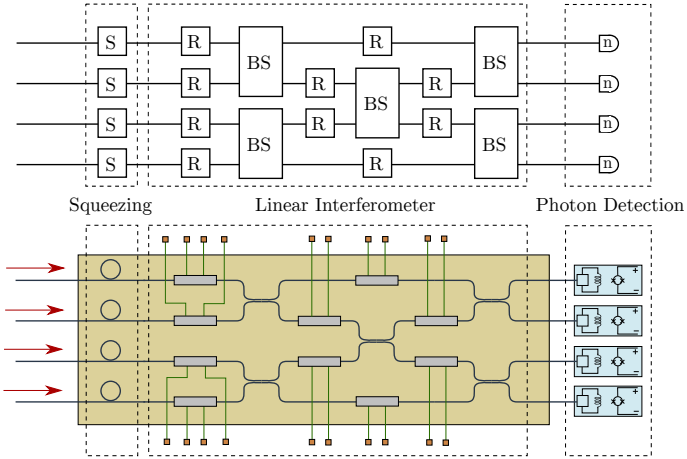
QMASM: A Quantum Macro Assembler

QMASM(Pakin 2016) fills a gap in the software ecosystem for D-Wave's adiabatic quantum computers by shielding the programmer from having to know system-specific hardware details while still enabling programs to be expressed at a fairly low level of abstraction. It is therefore analogous to a conventional macro assembler and can be used in much the same way: as a target either for programmers who want a great deal of control over the hardware or for compilers that implement higher-level languages.

QMASM


SF Web App
Strawberry Fields Frontend API
Compiler Engine
Simulators
BACKENDS
Hardware
GBS Applications
apps layer
sample
clique
points
plot
subgraph
similarity
vibronic
data

QMASM

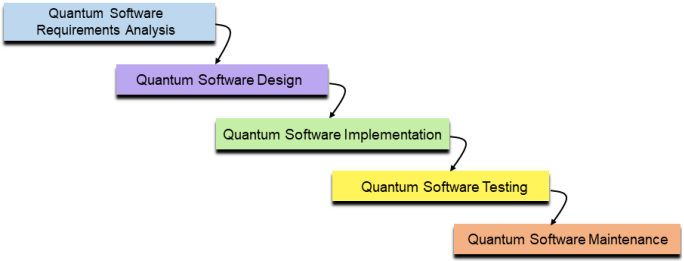


QMASM properties

- Documentation
- Tailored to D-Wave equipment:

High level Quantum programming

- Imperative languages
- Functional lanugages
- Hybrid, Object oriented?
- Making quantum computers easer to program

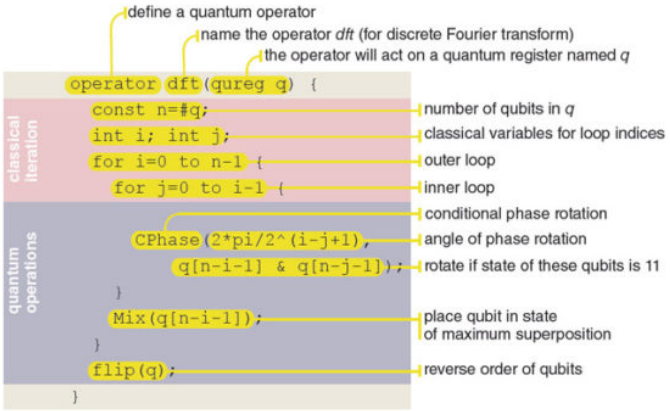


Imperative languages

- Q#
- Qiskit
- cirq,Tutorials(Arute et al. 2019)
- QCL
- Q$|S\,|\rangle$
- Q language
- qGCL: quantum Guarded-Command Language
- Silq : Strong static type system from ETH Zürich) (implemented in the D-language(Dlang, alternative to C++)

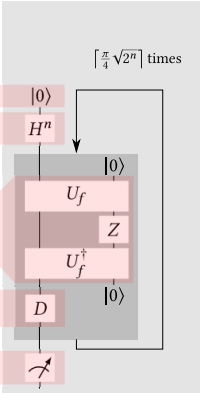
Imperative language example

intro 2021



Example Silq

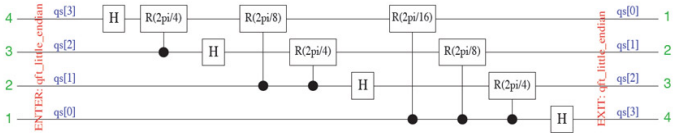

```
def grover[n:!ℕ](f:const int[n]!--qfree-->𝔹){
    nIterations:=⌈π/4 √(2^n)⌉;
    cand:=0:int[n];
    for k in [0..n) { cand[k] := H(cand[k]); }
    for k in [0..nIterations){
      if f(cand) { phase(π); }
      cand:=groverDiff[n](cand);
    }
    return measure(cand);
}
```



Functional languages

- QFC and QPL:classical control flow but can operate on quantum or classical data
- QML: a functional language for quantum computations on finite types
- LIQUIi$|\rangle$: Developed by Microsoft
- Quantum lambda calculi (Tonder 2004)
- FunQ: Functional Quantum Programming

Functional language example



Software Engineering and Quantum Computing

- Software engineering landscape (Zhao 2020)
- eScience Center guide
- Best practices ML software
- Five recommendations for FAIR software
- Victor Eijkhout's Art of High Performance Computing textbooks(includes parallel processing on supercomputers)

Conclusion

- Quantum computing requires program transformations for execution of quantum hardware
- Too many languages but standards are available
- Programming languages still under development
- Quantum software engineering under development

Exercises

Document the following:

- Compare an algorithm in Qiskit tutorials with a similar algorithm in another language (e.g. cirq and related examples Hidary 2021, Github repository
- Evaluate language design inspired by your project :
  - Protocol zoo and security
  - Transpiling and changing parameters
  - Optimization
- Compare at least two Quantum software development tools/Software engineering approaches (classical vs quantum, or different companies)

References I


Abhijith, J. et al. (2020). *Quantum Algorithm Implementations for Beginners*. arXiv: 1804.03719 [cs.ET].


Arute, Frank et al. (2019). "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779, pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5. URL: https://doi.org/10.1038/s41586-019-1666-5.


Bromley, Thomas R et al. (May 2020). "Applications of near-term photonic quantum computers: software and algorithms". In: *Quantum Science and Technology* 5.3, p. 034010. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ab8504. URL: http://dx.doi.org/10.1088/2058-9565/ab8504.


Cross, Andrew W. et al. (2017). *Open Quantum Assembly Language*. arXiv: 1707.03429 [quant-ph].

References II

- Hidary, Jack D. (2019). *Quantum Computing: An Applied Approach*. Springer.
- — (2021). *Quantum Computing: An Applied Approach*. second edition. Springer.
- Khammassi, N. et al. (2018). *cQASM v1.0: Towards a Common Quantum Assembly Language*. arXiv: 1805.09607 [quant-ph].
- Killoran, Nathan et al. (Mar. 2019). "Strawberry Fields: A Software Platform for Photonic Quantum Computing". In: *Quantum* 3, p. 129. ISSN: 2521-327X. DOI: 10.22331/q-2019-03-11-129. URL: http://dx.doi.org/10.22331/q-2019-03-11-129.
- Luo, Xiu-Zhe et al. (Oct. 2020). "Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design". In: *Quantum* 4, p. 341. ISSN: 2521-327X. DOI: 10.22331/q-2020-10-11-341. URL: https://doi.org/10.22331/q-2020-10-11-341.

References III


McKay, David C. et al. (2018). *Qiskit Backend Specifications for OpenQASM and OpenPulse Experiments*. arXiv: 1809.03452 [quant-ph].


Pakin, Scott (2016). "A quantum macro assembler". In: *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8. DOI: 10.1109/HPEC.2016.7761637.


Smith, Robert S., Michael J. Curtis, and William J. Zeng (2017). *A Practical Quantum Instruction Set Architecture*. arXiv: 1608.03355 [quant-ph].


Tonder, André van (Jan. 2004). "A Lambda Calculus for Quantum Computation". In: *SIAM Journal on Computing* 33.5, pp. 1109–1135. ISSN: 1095-7111. DOI: 10.1137/s0097539703432165. URL: http://dx.doi.org/10.1137/S0097539703432165.


Zhao, Jianjun (2020). *Quantum Software Engineering: Landscapes and Horizons*. arXiv: 2007.07047 [cs.SE].