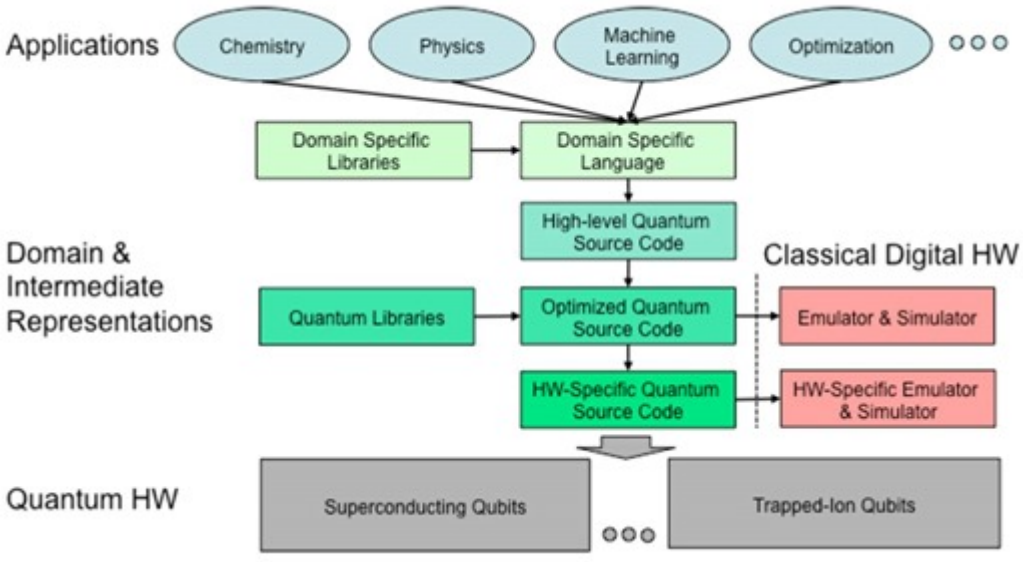
Opinions Libres
le blog d'Olivier Ezratty

Comprendre l'informatique quantique – outils de développement

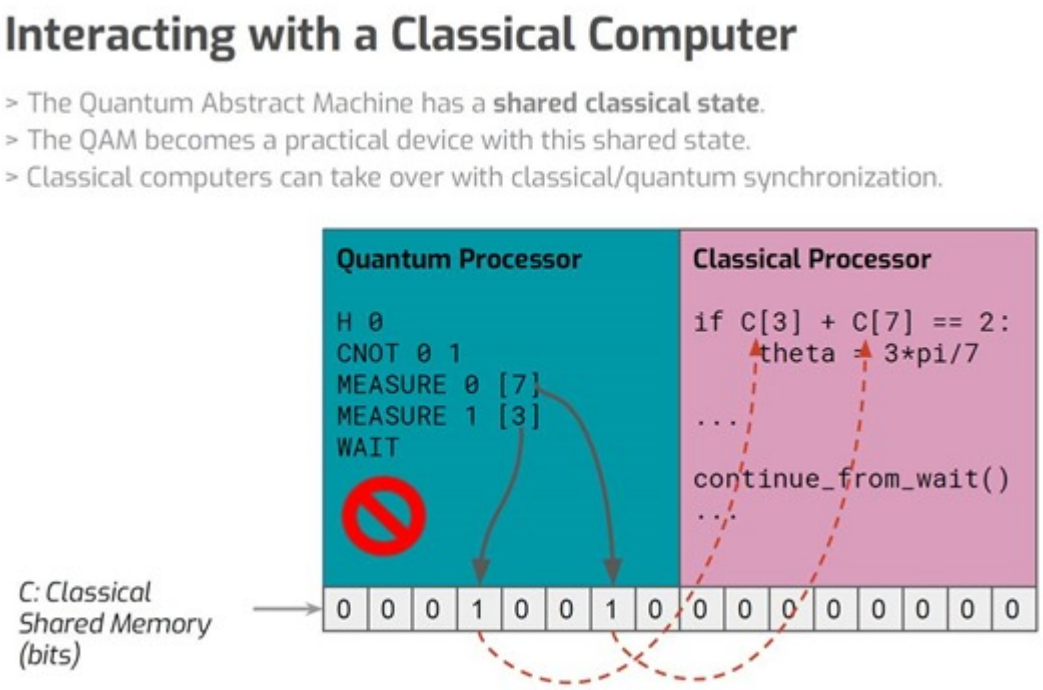
Après avoir fait le tour de **quelques algorithmes quantiques** de base puis des **théories de la complexité** qui permettent de détourer vaguement l'univers du possible pour le calcul quantique, il nous faut maintenant explorer les outils logiciels de l'informatique quantique. Comme pour tout le reste, c'est un monde entièrement nouveau et avec des paradigmes très différents par rapport à la création de logiciels classiques. On peut cependant y retrouver ses petits.

Qui dit algorithme dit programmation, langages de programmation et environnements de développement. Comme l'indique le schéma ci-dessous (dont j'ai perdu la source…), les logiciels quantiques sont organisés en couches superposées avec en partant du bas, les qubits, puis le langage machine permettant de les piloter, puis le "high level quantum source code" qui est en fait une sorte de macro-assembleur, pouvant tirer parti de bibliothèques de fonctions avec des algorithmes prêts à l'emploi (transformée de Fourier quantique, etc.) et enfin, d'éventuels langages de haut niveau adaptés à des besoins métiers spécifiques.



Dans les couches basses entre langage machine et le macro-assembleur se trouvent des fonctions de conversion des portes quantiques en portes quantiques universelles supportées par l'ordinateur quantique ainsi que les systèmes de codes de correction d'erreur qui peuvent demander l'exécution d'un grand nombre de portes quantiques. Un compilateur quantique va aussi faire de l'optimisation en supprimant par exemple les séquences de portes quantiques qui ne changent pas l'état d'un qubit, comme deux portes unitaires de Hadamard ou X (NOT) consécutives.

Les architectures logicielles du quantique sont généralement hybrides et permettent de contrôler un ordinateur quantique à partir de logiciels procéduraux assez traditionnels. Ils gèrent côte à côte l'exécution de logiciels classiques et de logiciels quantiques et manipulant de la mémoire traditionnelle en plus de celle des qubits, comme illustré dans le schéma ci-dessous originaire de la startup Rigetti.



L'ordinateur classique sert au minimum à contrôler l'exécution des algorithmes quantiques, ne serait-ce que pour déclencher les portes quantiques au bon moment, de manière séquentielle. Il peut aussi déclencher plusieurs algorithmes quantiques les uns après les autres. On peut imaginer qu'une application fera appel à plusieurs algorithmes quantiques et pas un seul.

Les classes d'outils de développement

On peut identifier quelques grandes classes d'outils de création de logiciels quantiques : les outils de programmation graphiques, les langages de scripting, les langages intermédiaires, les langages machine et les compilateurs.

Outils de programmation graphique

Ils permettent de spécifier la séquence des portes quantiques à exploiter pour créer des algorithmes directement exploitables dans des ordinateurs quantiques du cloud ou des simulateurs HPC dans le cloud. Ces outils peuvent faire fonctionner et visualiser l'état des qubits lorsque leur nombre est raisonnable. Ils permettent de vérifier la faisabilité de l'exécution de l'algorithme. L'un des exemples de tels outils est l'**IBM Q Experience** qui est proposé dans le cloud depuis 2016 (*ci-dessous*).

On y trouve aussi des simulateurs graphiques de qubits avec lesquels on peut se faire la main pour comprendre comment enchaîner les portes quantiques sur quelques qubits et visualiser le résultat visuellement. C'est notamment le cas de **QuantumPlayground** originaire de Google et de l'outil open source **Quirk**, ce dernier pouvant simuler jusqu'à 16 qubits. Il fonctionne en ligne et on peut le télécharger pour l'exécuter sur son propre ordinateur en local. Sa capacité n'est limitée que par la RAM dont vous disposez. Voici ci-dessous, un **exemple** de transformée de Fourier quantique réalisée en Quirk.

Je ferai en sorte de tester quelques algorithmes de base avec Quirk pour l'édition ebook de cette série d'articles !

Langages de scripting

Ils permettent de programmer en mode texte la structure des portes quantiques d'une solution. Ces outils permettent d'associer de la programmation classique avec enchainement de fonctions quantiques conditionnées par l'état de variables en mémoire classique.

On compte deux principaux types de langages quantiques : les langages impératifs et les langages fonctionnels. Les langages impératifs sont les langages de programmation procéduraux (objets ou pas) où l'on décrit les algorithmes pas à pas. On y range les langages habituels tels que C, C++, PHP ou Java.

Table 1: A selection of some quantum programming languages.

Name	Style	Notes
QCL	Imperative	Has classical sublanguage, multiple high-level programming features.
qGCL	Imperative	Emphasis on algorithm derivation and verification.
LanQ	Imperative	Full operational semantics, proven type soundness.
Quipper	Functional	Focus on scalability, plans to include linear types for static checks (currently done at run-time).
QPL	Functional	Statically typed, denotational semantics in terms of CPOs of superoperators.
QML	Functional	Linearly typed, focused on weakening - not contraction. Quantum control and quantum data.
Qumin	Functional	Two sublanguages (untyped and linearly typed). Focus on ease of use and clean, functional style of programming.

Les langages fonctionnels sont utilisés en définissant des fonctions diverses qui sont appelées de manière ad-hoc par le programme. Les boucles (For, While) sont remplacées par la récursivité de fonctions et il n'y a pas de variables modifiables. Ils permettent d'utiliser des type de données abstraits de haut niveau manipulés par les fonctions. L'ensemble est plus concis.

Une bonne part des langages de programmation traditionnels peuvent être exploités en programmation impérative ou fonctionnelle, notamment dès lors qu'ils disposent de pointeurs de fonctions ou qu'ils supportent une logique événementielle. Dans une certaine mesure, JavaScript et Jquery peuvent-être utilisés comme des langages fonctionnels via leurs *call-back functions*. C'est aussi le cas du C++.

Chez les fournisseurs d'ordinateurs quantiques tels qu'IBM ou Rigetti, deux types de langages sont parfois proposés : un langage intermédiaire (Quil chez Rigetti) et un langage de plus haut niveau sous la forme d'extensions du langage de programmation Python (pyQuil chez Rigetti). Un outil de conversion converti le second dans le premier langage.

Langages machine

Ce sont les langages de plus bas niveau de programmation de l'ordinateur quantique, qui programment l'initialisation des qubits et l'activation des portes universelles qui agissent dessus puis la mesure des résultats. Ils sont généralement spécifiques à chaque type d'ordinateur quantique, voir à chaque ordinateur quantique.

Compilateurs

Les compilateurs exploitent le contenu des précédents outils, et surtout des langages de scripting, pour générer la séquence de contrôle des portes physiques de l'ordinateur quantique cible en langage machine, intégrant au passage les fonctions de corrections d'erreurs quantiques (QEC). Ces compilateurs vont transformer les portes logiques utilisées dans la programmation en portes physiques universelles exploitées par l'ordinateur quantique. Ils vont aussi calculer les temps d'activation des portes et vérifier que l'accumulation de ces temps d'activation est inférieure au temps de cohérence des qubits de l'ordinateur cible.

Comme le aASM d'Atos, ces outils de compilation peut être "multiplateformes" et supporter différentes architectures d'ordinateurs quantiques, au moins universels. La compilation n'est pas la même sur des ordinateurs quantiques topologiques. Ces compilateurs utilisent des langages de programmation quantiques. Les langages de programmation quantique sont généralement capables d'associer de la programmation procédurale classique avec de la programmation de registres et portes quantiques. Ils permettent de gérer parallèlement de la mémoire classique à des registres quantiques. Ils proviennent de la recherche ou de concepteurs d'ordinateurs quantiques comme IBM, Microsoft, Rigetti et D-Wave.

Voir cette présentation qui décrit bien quelques-unes des tâches réalisées par des compilateurs quantiques : **Opportunities and Challenges in Intermediate-Scale Quantum Computing** de Fred Chong, 2018 (34 slides).

Langages de programmation quantiques issus de la recherche

Voici un aperçu des principaux langages quantiques créés à ce jour, avec tout d'abord les langages indépendants des architectures matérielles et qui sont souvent issus de laboratoires de recherche.

Ils présentent l'inconvénient de ne pas être généralement reliés à des offres d'ordinateurs quantiques dans le cloud. Ils ont par contre souvent un certain privilège d'antériorité par rapport aux outils de développement des fournisseurs d'ordinateurs quantiques que nous verrons plus loin. C'est bien normal puisque les chercheurs sont les premiers à s'embarquer dans les domaines émergents, bien avant les acteurs privés. Ils ont souvent conçu les premiers langages de programmation quantique à une époque où l'on n'arrivait à aligner qu'à peine un à deux qubits !

Ce sont un peu les Kernighan et Richie (créateurs du langage C) et Bjarne Stroustrup (créateur du C++) du domaine ! Vous remarquerez au passage qu'un bon nombre de ces langages provient d'Europe.

- **QCL** ou Quantum Computation Language dispose d'une syntaxe et des types de données proches de ceux du langage C. Ce langage est l'un des premiers qui soit pour la programmation quantique, créé en 1998 par le chercheur Autrichien **Bernhard Ömer** de l'Austrian Institute of Technology à Vienne. Il est décrit dans **Structured Quantum Programming**, 2009 (130 pages) qui positionne très bien les différences conceptuelles entre langages de programmation classiques et quantiques (*ci-dessous*).

Classical concept	Quantum analogue
classical machine model	hybrid quantum architecture
variables	quantum registers
variable assignments	elementary gates
classical input	quantum measurement
subroutines	operators
argument and return types	quantum data types
local variables	scratch registers
dynamic memory	scratch space management
boolean expressions	quantum conditions
conditional execution	conditional operators
selection	quantum if-statement
conditional loops	quantum forking

Table 2.1: *Classical and quantum programming concepts*

- **Q Language** est une extension du langage C++ qui fournit des classes permettant de programmer des portes quantiques (Hadamard, CNOT, SWAP, QFT pour transformée de Fourier quantique). Il est documenté dans **Toward an architecture for quantum programming**, 2003 (23 pages), avec comme coauteur un certain Stefano Bettelli du Laboratoire de Physique Quantique de l'Université Paul Sabatier de Toulouse.

- **QFC** et **QPL** sont deux langages fonctionnels définis par le Canadien Peter Selinger, le premier utilisant une syntaxe graphique et le second, une syntaxe textuelle. Ils sont décrits dans **Towards a Quantum**

Programming Language, 2003 (56 pages).

- **QML** est un langage de programmation fonctionnel créé par les Anglais Thorsten Altenkirch et Jonathan Grattage dans **A functional quantum programming language**, 2004 (15 pages) et dont les principes sont bien décrits dans la présentation **Functional Quantum Programming** (151 slides).

- **qGCL** ou Quantum Guarded Command Language a été créé par Paolo Zuliani de l'Université de Newcastle et est décrit dans **Compiling quantum programs**, 2005 (39 pages).

- **Scaffold** est un langage issu de l'Université de Princeton. Il est décrit dans **Scaffold: Quantum Programming Language**, 2012 (43 pages). Il permet notamment de programmer du code traditionnel qui est ensuite transformé automatiquement en portes quantiques via sa fonction C2QG (Classical code to Quantum Gates). Scaffold peut notamment générer du QASM. En voici un exemple de code, presque facile à comprendre ! Son développement a été également financé par l'IARPA.

```
// Pauli X, Pauli Y, Pauli Z, Hadamard, S, and T gates
gate X(qreg input[1]);
gate Y(qreg input[1]);
gate Z(qreg input[1]);
gate H(qreg input[1]);
gate S(qreg input[1]);
gate T(qreg input[1]);

// Daggered gates
gate Tdag(qreg input[1]);
gate Sdag(qreg input[1]);

// CNOT gate defined on two 1-qubit registers
gate CNOT(qreg target[1], qreg control[1]);

// Toffoli (CCNOT) gate
gate Toffoli(qreg target[1], qreg control1[1], qreg control2[1]);

// Rotation gates
gate Rz(qreg target[1], float angle);        //Arbitrary Rotation

// Controlled rotation
gate controlledRz(qreg target[1], qubit control[1], float angle);

// One-qubit measurement gates
gate measZ(qreg input[1], bit data);
gate measX(qreg input[1], bit data);

//One-qubit prepare gates: initializes to 0
gate prepZ(qreg input[1]);
gate prepX(qreg input[1]);

//Fredkin (controlled swap) gate
gate fredkin(qreg targ[1], qreg control1[1], qreg control2[1])
```

- **Quipper** est un langage créé en 2013 qui s'appuie sur le langage classique **Haskell**, créé en 1990, auquel il fournit des extensions sous forme de bibliothèques. Il est documenté dans **An Introduction to Quantum Programming in Quipper**, 2013 (15 pages). Il manipule une version logicielle de QRAM, l'état des registres quantiques, indispensable à l'exécution d'algorithmes quantiques comme celui de Grover. Sa création a été financée par l'IARPA, l'agence fédérale du renseignement américain qui finance de la R&D comme le fait la DARPA dans la défense. L'IARPA est rattachée au DNI (Director of national Intelligence), le coordinateur du renseignement US rattaché à la Maison Blanche et à qui reportent les 17 patrons du renseignement US dont ceux de la CIA et de la NSA. Malgré tout cela, le langage ne semble pas avoir évolué depuis 2016.

- **ProjectQ** est un langage de scripting l'ETH Zurich qui prend la forme d'un framework Python open source,

diffusé sur GitHub depuis 2016. Il comprend notamment un compilateur convertissant le code quantique en langage C++ pour son exécution dans un simulateur quantique à processeur traditionnel. Cf **ProjectQ: An Open Source Software Framework for Quantum Computing** de Damian Steiger, Thomas Häner et Matthias Troyer, 2018 (13 pages) qui explique bien comment le compilateur optimise le code en fonction des portes disponibles dans l'ordinateur quantique (*ci-dessous*). Lancé début 2017, il supporte les ordinateurs quantiques d'IBM via leur langage OpenQASM Ce qui est normal puisque l'ETH Zurich est partenaire de ce dernier, ainsi que la simulation sur ordinateur traditionnel via une implémentation développées en C++ qui supporte jusqu'à 28 qubits. ProjectQ est compatible avec OpenFermion de Rigetti et Google, cité plus loin.

(a) After decomposing. (b) After CNOT mapping. (c) After optimization. (d) After mapping to hardware.

Figure 5: Individual stages of compiling an entangling operation for the IBM back-end. The high-level Entangle-gate is decomposed into its definition (Hadamard gate on the first qubit, followed by a sequence of controlled NOT gates on all other qubits). Then, the CNOT gates are remapped to satisfy the logical constraint that controlled NOT gates are allowed to act on one qubit only, followed by optimizing and mapping the circuit to the actual hardware.

- **QWire** est un autre langage de programmation quantique, issu de l'Université de Pennsylvanie (Upenn), documenté dans **QWIRE: A Core Language for Quantum Circuits** (13 pages) et dans **A core language for quantum circuits**, 2017 (97 slides).

- **Qubiter** est un langage open source développé en Python utilisable au-dessus d'OpenQASM d'IBM et OpenFermion de Google. Il a donc une application industrielle plus directe que les langages cités ci-dessus. Il date aussi de 2017.

- **Qumin** est un langage quantique minimaliste conçu en Grèce en 2017. Cf **Qumin, a minimalist quantum programming language**, 2017 (34 pages). Il est disponible en open source.

- **QuEST** (**Quantum Exact Simulation Toolkit**) est un simulateur quantique développé en langage C et supportant les APIs QUDA et les GPU de Nvidia, créé par des chercheurs de l'Université d'Oxford et open source. Le système permet de simuler de 26 à 45 qubits selon la mémoire RAM disponible, respectivement de 2 Go et 256 Go. Il date aussi de 2017.

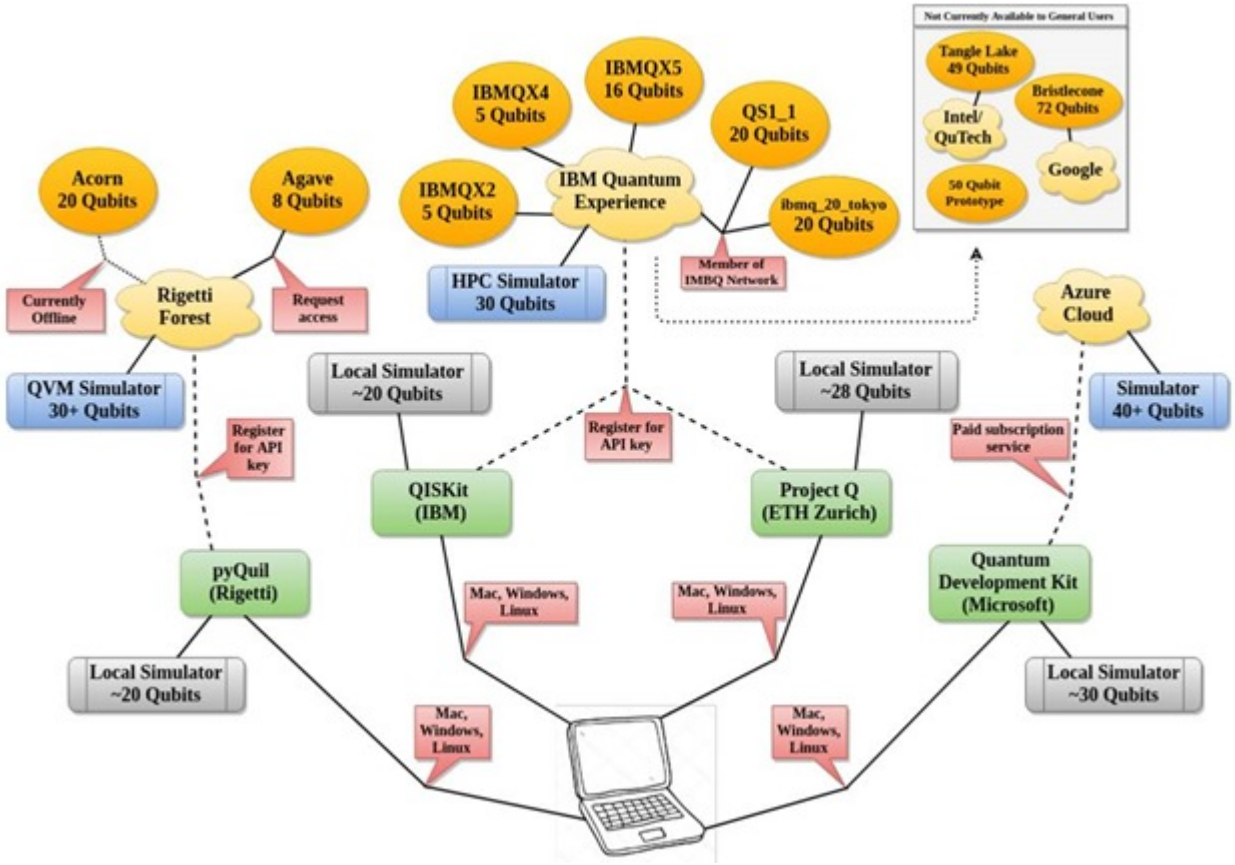
- S'ensuivent des langages de mise en œuvre quantique du **lambda calculus**, conceptualisé par Alonzo Church et Stephen Cole Kleene pendant les années 1930. Traduction en langage naturel ? Ce type de calcul permet de résoudre des problèmes très complexes et de type NP-complet, la classe des problèmes vérifiable en temps polynomial et dont la résolution requiert un temps exponentiel sur ordinateurs classiques et potentiellement polynomial sur ordinateurs quantiques !

Dans la pratique, peu de développeurs d'applications commerciales vont exploiter les langages ci-dessus. Ils vont plutôt s'ancrer dans les langages issus des fournisseurs d'ordinateurs quantiques commerciaux que voici *ci-dessous*.

Outils de développement des concepteurs de calculateurs quantiques

Avant même que les ordinateurs quantiques universels soient opérationnels à une échelle exploitable, la bataille des plateformes est déjà enclenchée. Les grands acteurs de l'ordinateur quantique ont presque tous adopté une approche d'intégration verticale de bout en bout allant de l'ordinateur aux outils de développement. C'est en particulier le cas chez IBM, Microsoft, Rigetti et D-Wave.

C'est bien illustré dans le schéma *ci-dessous* de synthèse découvert dans **Overview and Comparison of Gate Level Quantum Software Platforms** de Ryan LaRose, juin 2018 (18 pages) qui décrit par ailleurs fort bien les principaux environnements de développement d'applications quantiques de Rigetti et IBM et dont est inspirée la partie qui les concerne ici même.

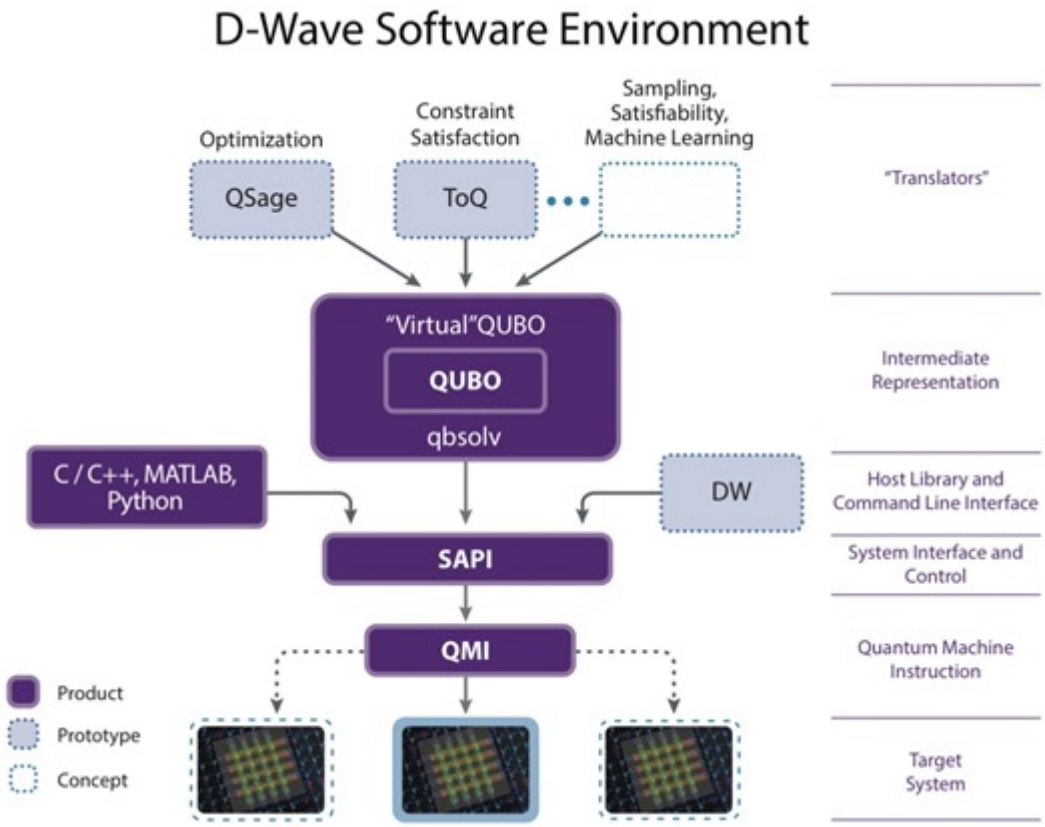


L'offre verticalisée des acteurs cités intègre souvent un langage quantique de bas niveau assimilable au langage machine, puis un langage de plus haut niveau assimilable au macro-assembleur des ordinateurs traditionnels, puis un framework open source exploitable le plus souvent en Python avec des fonctions prêtes à l'emploi, un environnement de développement, éventuellement, un simulateur graphique de portes quantiques et souvent, une offre d'accès à l'ensemble en cloud.

Reste à inventer les langages avec un très haut niveau d'abstraction permettant de s'affranchir des portes quantiques ! Pour l'instant, il n'en existe pas à ma connaissance, sauf dans une certaine mesure autour des ordinateurs quantiques de D-Wave dont le modèle de programmation est particulier.

D-Wave

C'est le plus ancien acteur du marché. Il propose une gamme complète d'outils logiciels qui ont bien évolué depuis sa création. L'architecture n'est d'ailleurs pas évidente à suivre. Le plus bas niveau d'accès aux ordinateurs D-Wave est le langage QMI, sorte de langage machine de définition des liens entre les qubits reliés entre eux dans le processeur quantique de l'ordinateur adiabatique. QMI est exploitable à partir des langages C, C++ Python et même Matlab, via l'interface SAPI (Solver API).



Au-dessus de QMI se situe une surcouche avec un plus haut niveau d'abstraction, qbsolv, qui est une bibliothèque open source depuis fin 2017. Elle permet de résoudre des problèmes d'optimisation en décomposant un problème QUBO (Quadratic Unconstrained Binary Optimization) pour le faire traiter par un ordinateur D-Wave ou un ordinateur traditionnel.

Les développeurs peuvent aussi faire appel au langage open source QMASM (Quantum Macro Assembler) qui est un langage de bas niveau adapté à la programmation sur ordinateur à recuit quantique D-Wave. Comme qbsolv, QMASM permet de décrire un "hamiltonien" fait de relations entre qubits à base de coupleurs qui sont dotés d'un poids comme le poids des synapses dans un réseau de neurone. Lors de l'exécution, l'ordinateur D-Wave cherche ensuite à déterminer un minimum énergétique de ce système pour le faire converger vers une solution. Cette méthode présente un inconvénient : il est préférable d'initialiser le système dans un état voisin de la solution recherchée et cet état ne peut être déterminé que par des calculs traditionnels. Bref, c'est un peu le serpent qui se mort la queue !

C'est en tout cas un modèle de programmation très différente de celui des portes quantiques universelles même s'il existe une équivalence théorique entre les modèles adiabatiques et à portes quantiques universelles comme nous l'avons vu dans la **partie précédente**.

QMASM est aussi intégré dans Quadrant, une plateforme complète pour le développement de solutions D-Wave dans le cloud appliquées au machine learning et lancée par D-Wave fin 2017 (**source**).

Enfin, on peut ajouter des surcouches tierces-parties comme QSage, un framework destiné à la résolution de problèmes d'optimisation et ToQ, un autre framework, pour résoudre des problèmes de satisfaction de contraintes ainsi que le SDK de la startup canadienne 1Qbit. Avec ces surcouches, on commence à se rapprocher des solutions métiers.

A ce jour, D-Wave, ses partenaires et clients ont prototypé un bon nombre d'algorithmes et solutions. Nous les évoquerons dans une partie à venir sur l'offre des différents acteurs du marché.

IBM

IBM propose OpenQASM, un langage de programmation qui complète son outil de programmation graphique en ligne Q Experience. Il est spécifié dans **Open Quantum Assembly Language**, 2017 (24 pages), ce document décrivant au passage les nombreuses tâches réalisées par le compilateur associé. OpenQASM comprend une douzaine de commandes.

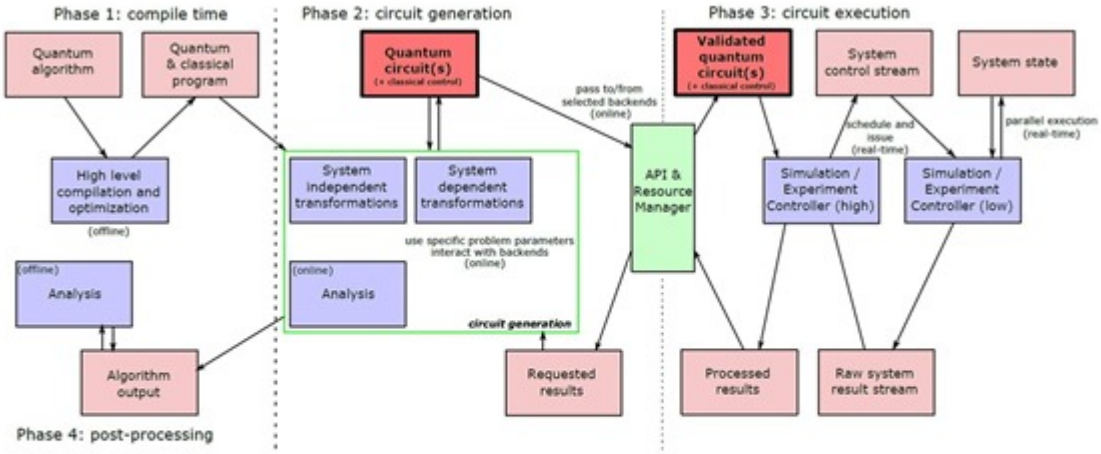
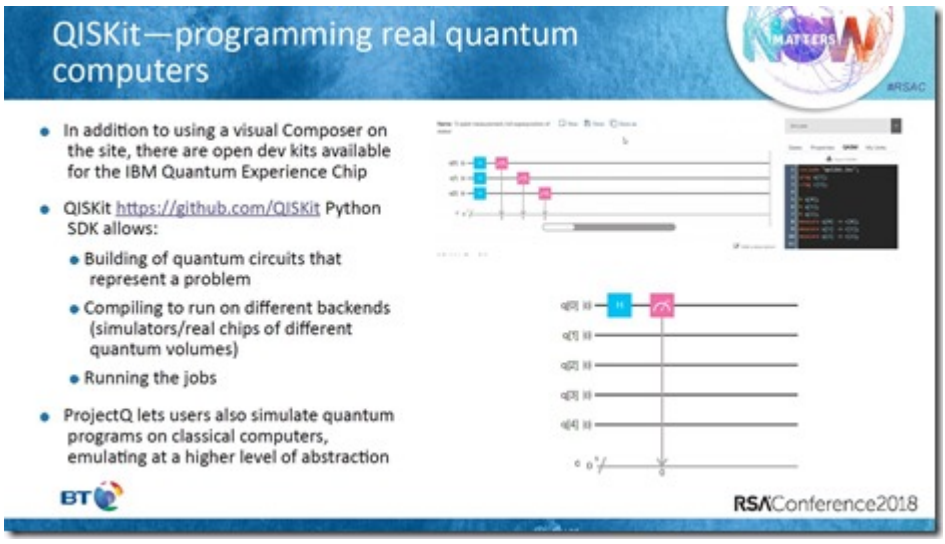


Figure 1: Block diagrams of processes (blue) and abstractions (red) to transform and execute a quantum algorithm. The emphasized quantum circuit abstraction is the main focus of this document. The API and Resource Manager (green) represents the gateway to backend processes for circuit execution. Dashed vertical lines separate offline, online, and real-time processes.

Un langage de scripting de haut niveau associé à OpenQASM est proposé par IBM : **Qiskit**, exploitable en Python, JavaScript et Swift (un langage généraliste d'Apple) et sur Windows, Linux et MacOS. Il a été lancé début 2017 et est en open source. Le slide ci-dessous qui décrit Qiskit est issu de la présentation **Quantum Computing is Here Powered by Open Source** 2018 (41 slides, **vidéo**).



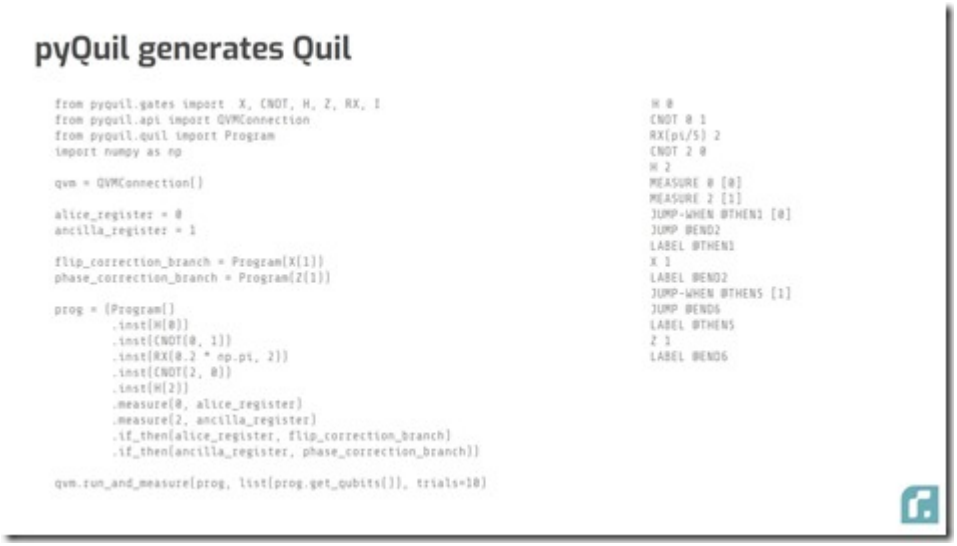
Qiskit est fourni avec de nombreux templates et exemples de codes permettant d'exploiter une vaste gamme d'algorithmes quantiques connus. Il comprend une fonction "circuit-drawer" qui génère une visualisation graphique des circuits quantiques programmés en passant par le langage de composition de documents open source LaTeX. La bibliothèque Aqua de Qiskit permet quant à elle de développer pour des NISQ, les premiers ordinateurs quantiques universels (Noisy Intermediate-Scale Quantum selon l'appellation de John Preskill).

La compilation du code quantique a ensuite lieu pour exécuter l'ensemble soit sur le simulateur HPC classique en cloud d'IBM soit sur un ordinateur quantique universel comme ceux d'IBM qui sont également accessibles dans le cloud.

Rigetti

La startup américaine développe des ordinateurs quantiques universels à base de qubits supraconducteurs dans la lignée de ce que fait IBM. Elle propose aussi une plateforme intégrée avec le langage de bas niveau Quil qui supporte un modèle de mémoire mixte classique et quantique. Il est documenté dans **A Practical Quantum Instruction Set Architecture**, 2017 (15 pages). Il fonctionne sur Windows, Linux et MacOS. Le langage utilise la classe gates pour décrire les opérations à opérer sur les qubits, indexés de 0 à n-1, pour n qubits et avec des portes quantiques. Le langage permet de créer de la programmation conditionnelle en fonction de l'état des qubits.

Il est complété du langage de scripting **pyQuil** open source lancé début 2017 qui est doté de la bibliothèque Grove d'algorithmes quantiques de base (**documentation**). Le pyQuil de haut niveau (assembleur) génère le langage Quil de bas niveau (code machine).



En voici un simple exemple avec un seul qubit activé par une porte de Hadamard qui créé une superposition d'état 0 et 1 permettant de créer un générateur de nombre vraiment aléatoire. Utilisé de manière itérative dans une boucle classique, le programme peut générer une série aléatoire de 0 et de 1 avec 50% de chances d'avoir l'un ou l'autre permettant de créer un code binaire unique complètement aléatoire.

```
1 # random number generator circuit in pyQuil
2 from pyquil.quil import Program
3 import pyquil.gates as gates
4 from pyquil import api
5
6 qprog = Program()
7 qprog += [gates.H(0),
8           gates.MEASURE(0, 0)]
9
10 qvm = api.QVMConnection()
11 print(qvm.run(qprog))
```

porte de Hadamard sur qubit 1

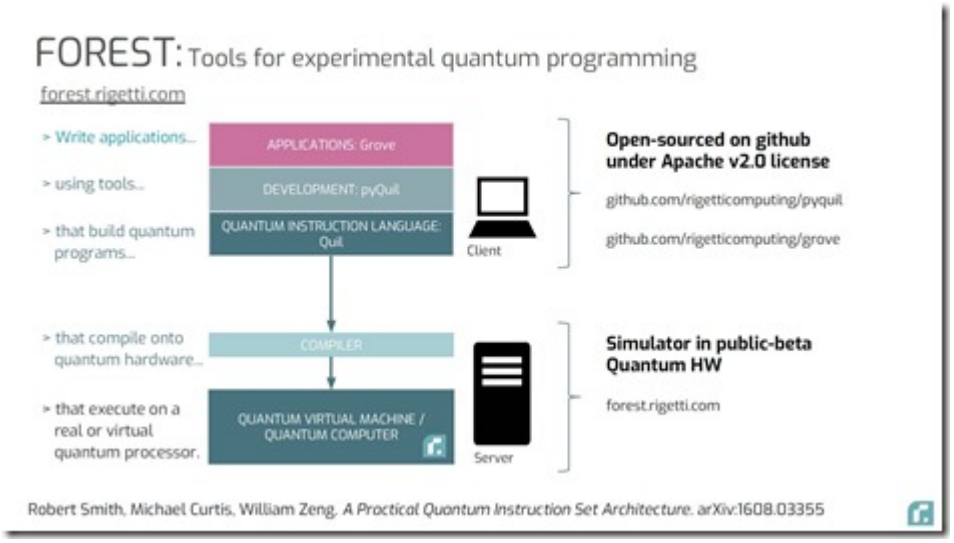
lecture de l'état du qubit superposé

sortie du résultat

Listing 2: pyQuil code for a random number generator.

Rigetti propose l'exécution de programmes quantiques dans ses ordinateurs en cloud et sur des simulateurs classiques via ses QVM, pour **Quantum Virtual Machines**. C'est documenté dans **pyQuil Documentation**, juin 2018 (120 pages) qui contient de nombreux exemples de code comme celui *ci-dessous*.

Quil est utilisable à partir de l'environnement de développement **Forest** proposé par Rigetti. Ces outils sont open source, mais pas multiplateforme. Dommage !



Fin 2017, Google et Rigetti, lançaient l'initiative open source **OpenFermion**. Ce framework développé en Python exploite aussi les travaux des universités de Delft (Pays-Bas) et de Leiden. C'est une solution logicielle de création d'algorithmes quantiques de simulation de fonctions chimiques supportant tout ordinateur quantique, des ordinateurs quantiques universels aux ordinateurs quantiques adiabatiques de D-Wave. C'est une initiative intéressante car elle créé une ouverture multiplateforme sur un domaine d'application clé des ordinateurs quantiques. Elle complète l'approche multiplateforme d'Atos. Voir l'**annonce** en octobre 2017, **OpenFermion: The Electronic Structure Package for Quantum Computers**, 2018 (19 pages) et la **documentation d'Openfermion**.

OpenFermion

Google

En plus d'OpenFermion qui est plutôt un framework de haut niveau, Google lançait le 19 juillet 2018 son propre langage quantique dénommé Cirq en open source. Il cible les NISQ, l'appellation de John Preskill déjà évoquée, des ordinateurs à portes quantiques universelles de taille et performance intermédiaire côté taux d'erreurs. C'est un framework pour Python. Il servira notamment à programmer les ordinateurs quantiques de Google notamment celui de 72 qubits qui a été annoncé en mars 2018 mais n'est toujours pas opérationnel.

Cirq

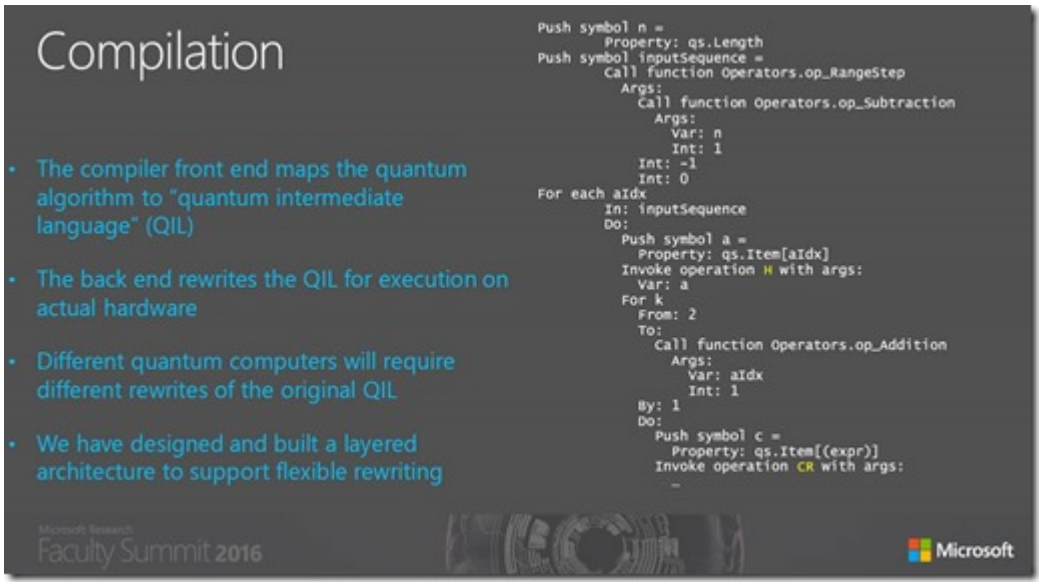
Il doit supporter d'autres ordinateurs quantiques, pas encore précisés à ce stade. Il est aussi accompagné d'un simulateur. Voir les explications dans **Google Cirq and the New World of Quantum Programming** de Jesus Rodriguez, juillet 2018. Un outil de compilation de code OpenFermion en Cirq est aussi proposé.

Microsoft

L'éditeur propose trois briques pour les développeurs avec l'extension **LIQUi|>** du langage de scripting **F#** qui permet de faire de la simulation de programme quantique.

Le tout est complété depuis décembre 2017 par le langage **Q#** qui semble surtout adapté à la programmation d'ordinateurs quantiques topologiques, qui n'existent pas encore, mais sont simulables sur ordinateurs classiques dans le cloud avec jusqu'à 30 qubits.

Q# a une syntaxe dérivée du langage C# de Microsoft. Le tout est fourni sous la forme d'extensions de l'environnement de développement Visual Studio et dans le QDK, pour Quantum Development Kit. Un langage intermédiaire est généré par le compilateur, QIL. Il est censé être multiplateforme mais ne l'est pas encore. Microsoft aura certainement intérêt à rendre multiplateformes ses outils de développement pour capter l'attention et le temps des développeurs.



IonQ

La startup issue de l'Université de Maryland planche sur la création d'ordinateurs quantiques à base d'ions piégés. Nous détaillerons cela plus tard. Comme Rigetti, ils veulent aussi créer une offre logicielle "full stack" adaptée à leur architecture d'ordinateur quantique, et proposée en cloud.

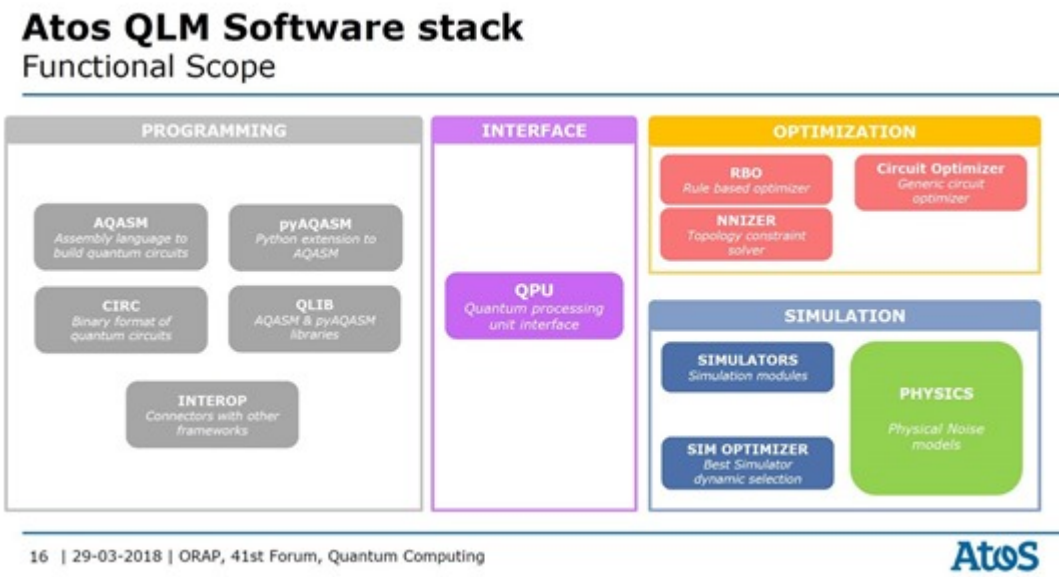
Intel

A ce stade, Intel n'est pas très avancé côté développement de logiciels quantiques. Ils ont créé à ce stade un logiciel de simulation quantique pour ordinateurs classiques, qui est documenté dans **qHiPSTER: The Quantum High Performance Software Testing Environment** de Mikhail Smelyanskiy, Nicolas Sawaya, et Alan Aspuru-Guzik, 2016 (9 pages), les deux premiers auteurs travaillant chez Intel et de dernier à Harvard. Il peut simuler jusqu'à une quarantaine de qubits.

Atos

Atos n'est pas encore un fabricant d'ordinateurs quantiques même si leurs partenariats avec le CEA laissent indiquer que cela pourrait les intéresser. Ils proposent pour l'instant un simulateur de logiciels quantiques à base de supercalculateur à processeurs Intel et avec leur propre architecture mémoire optimisée, les aQML. Ils simulent de 30 à 40 qubits.

aQASM (Atos Quantum Assemby Programming Language) est un langage de programmation complétant Python qui permet de créer des algorithmes quantiques exécutables sur les simulateurs aQML ou sur toute architecture physique d'ordinateur quantique universel (à terme). Le langage permet de définir des portes quantiques utilisant d'autres portes quantiques, l'équivalent des objets, fonctions ou des macros dans la programmation traditionnelle. **Source** du schéma *ci-dessous*.



aQASM est une déclinaison du langage standard OpenQASM, évoqué plus haut. Ce langage est complété par une bibliothèque Python dénommée PyAQASM qui permet de générer des fichiers aQUASM. Le langage permet de programmer l'exécution répétitive de portes en boucles et de créer des fonctions réutilisables. Le compilateur du code aQASM génère un code binaire CIRC qui est le langage pivot de bas niveau, ensuite converti dans le langage de contrôle d'ordinateurs quantiques universels spécifiques ou pour des supercalculateurs de simulation via l'interface QPU (Quantum Processing Unit Interface). Il est complété de plugins d'optimisation divers qui vont éliminer les portes qui ne servent à rien et adapter le code à l'architecture matérielle ciblée.

Vue d'ensemble

J'ai essayé de consolider une vue d'ensemble des offres "propriétaires" de programmation quantique, en mettant de côté les langages de programmation fonctionnels et impératifs issus des laboratoires de recherche. Cela donne le schéma suivant qui positionne les outils de développement en fonction de leur niveau. Comme je ne suis pas sûr de toutes les composantes intégrées, je corrigerai au fil de l'eau si besoin est ! Mes articles sur ces sujets nouveaux sont ainsi des puzzles qui se construisent dynamiquement !

principales plateformes logicielles intégrées

	conception visuelle	plateforme	bibliothèques	langage de haut niveau	langage intermédiaire	langage machine	cloud
D-Wave	N/A	Quadrant (ML)	Qsage, ToQ, …	Qbsolv	QMASM	QMI	oui
Google	Quantum Playground	?	OpenFermion (chemistry)	Cirq	?	?	oui
IBM	Q Experience	QisKit	QisKit	QisKit	OpenQASM	?	Q Experience
rigetti	N/A	Forest	OpenFermion (chemistry)	pyQuil	N/A	Quil	QVM
Microsoft	N/A	LIQUi|>	?	Q#	?	N/A	oui
IONQ	oui	?	?	?	?	?	oui
Atos	N/A	QML	QLIB	pyAQASM	AQASM	CIRC QPU	

(cc) Olivier Ezratty, 2018

Le plus intéressant dans tout cela est que nombreux sont les outils de développement qui permettent de se faire la main sur des algorithmes quantiques à petite échelle avant que de “gros” ordinateurs quantiques soient disponibles. A vous de jouer si le cœur vous en dit !

Dans la **prochaine partie**, nous irons faire un tour du côté des applications quantiques par secteur d’activité avec notamment la chimie, l’énergie, l’environnement, la santé, le marketing, la finance et le renseignement. Nous en aurons alors terminé avec la partie “logicielle” de cette série et reviendront au matériel avec un panorama des ordinateurs quantiques classés par technologie de qubits.

Cet article a été publié le 31 juillet 2018 et édité en PDF le 15 mars 2024.