PROGRAMMEZ!

Le magazine des dévs - CTO & Tech Lead

LES RÉVOLUTIONS DE L'INFORMATIQUE QUANTIQUE

EN PARTENARIAT AVEC

EVIDEN intel

SPÉCIAL HIVER 2023

M 01642 - 13 - F: 6,99 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7,50 € P928345 - Canada 10,55 $ CAN - SUISSE 14,10 FS - DOM Surf 8,10 € - TOM 1100 XPF - MAROC 59 DH

Plutôt qu'offrir un livre, offrez-en des milliers !

Accédez à la plus riche bibliothèque informatique de France

29€ le 1er mois (au lieu de 49€)*

avec le code :

Programmez29

eni

www.editions-eni.fr

*Offre valable jusqu'au 31/12/2023. 29€ le 1er mois au lieu de 49€, avec tacite reconduction à 49€/mois, les mois suivants. Abonnement résiliable à tout moment depuis votre espace client Editions ENI ou en contactant notre service client. Non cumulable avec d'autres promotions en cours. Non valable pour les webpros.

Contenus

Programmez! est une publication bimestrielle de Nefer-IT. Adresse : 57, rue de Gisors 95300 Pontoise – France. Pour nous contacter : redaction@programmez.com

#HS13

Les révolutions de l'informatique quantique

Quel est l'état de cet édito ? 0 ou 1 ? Mort ou vivant ?
Réponse en ouvrant les pages de ce numéro 100 % quantique !

Nous remercions tous les contributeurs de ce numéro exceptionnel ainsi que les partenaires (Eviden et Intel). Ce hors-série n'a pas été le numéro le plus facile à créer. L'idée a germé durant la DevCon quantique du printemps 2023 à l'ESGI.

Les promesses du quantique sont immenses mais de quoi parle-t-on réellement ? De quelle quantique ? Matériel ? Logiciel ? Les deux ?

J'espère que ce numéro vous donnera envie d'aller plus loin et de découvrir par vous-même l'univers fascinant du quantique.

François Tonic
État indéfini

PROCHAIN NUMÉRO

PROGRAMMEZ! N°261

Disponible à partir du
26 janvier 2024

NUMÉRO EN COURS

PROGRAMMEZ! N°260

Toujours disponible

Les événements Programmez!

Meetups Programmez!

Les meetups 2024 :

9 janvier 2024
6 février 2024

Où : The Coding Machine
56 Rue de Londres, 75008 ParisParis
A 5 minutes de la gare Saint Lazarre : ligne L, H,
métro 3 / 9 / 12 / 14

Conférence DevCon #21

100 % IA

Une conférence développeur exceptionnelle sur l'IA, l'IA générative : usage, les modèles de développement, comment concevoir et déployer, la partie matérielle.

7 mars 2024 : au campus de l'école 42 !

Informations & inscription : programmez.com

Décembre 2023

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
				1	2	3
4	5	6	7	8	9	10
		Open Source Experience / Paris		DevFest Dijon		
		API Days / Paris				
11	12	13	14	15	16	17
			DevCon Sécurité / Paris			
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Janvier 2024

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
	Meetup Programmez!					
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				
		SnowCamp / Grenoble				

Février 2024

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
			1	2	3	4
			SnowCamp / Grenoble			
			AgileLeMans			
5	6	7	8	9	10	11
	DevFestParis / Paris					
	Meetup Programmez!					
12	13	14	15	16	17	18
			Touraine Tech / Tours			
19	20	21	22	23	24	25
26	27	28	29			

1er semestre 2024

FOSDEM : 3&4 février / Belgique
FlowCon : 6-7 mars / Paris
pgDayParis : 14-15 mars / Paris
KubeCon+CloudNativeCon Europe : 19-22 mars / Paris
SymfonyLive Paris : 28-29 mars / Paris
Devoxx France : 17-19 avril / Paris
MiXiT : 25-26 avril / Paris
Android Makers : 25-26 avril / Paris
AFUP Day : 4 mai / Nancy, Poitiers, Lille, Lyon
DevFest Lille : 6-7 juin / Lille

Merci à Aurélie Vache pour la liste 2023/2024, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

PROCHAIN NUMÉRO

PROGRAMMEZ! N°261

Disponible à partir du
26 janvier 2024

NUMÉRO EN COURS

PROGRAMMEZ! N°260

Toujours disponible

Former les talents de demain a l'informatique quantique et aux technologies quantiques

Axel Ferrazzini
Chef de la majeure Informatique & Technologies Quantiques à l'EPITA

Les étudiants de la majeure quantum de l'EPITA découvrent la technologie des qubits de chat chez la startup française Alice & Bob

Changement de paradigme

Les ingénieurs en informatique classique connaissent des carrières riches, diverses et évoluant au fur et à mesure des avancées technologiques. Cependant passer de l'informatique classique qui utilise des bits (qui peuvent prendre comme valeur 0 ou 1) à l'informatique quantique qui utilise des qubits (qui ne prennent pas comme valeur 0 ou 1, mais une superposition de 0 et de 1) n'est pas une évolution simple, c'est un changement de paradigme qui requière une formation spécifique.

Création de la formation

L'informatique quantique utilise des propriétés de la physique quantique telles que la superposition et l'intrication quantiques. C'est pourquoi l'EPITA proposait depuis plusieurs années une mineure Quantique animée par un des spécialistes les plus réputés du domaine, Olivier Ezratty, afin de permettre aux ingénieurs en informatique (i.e. Computer Science) d'avoir une acculturation quantique et d'éveiller en eux l'envie de s'impliquer dans l'informatique quantique, les technologies quantiques et les technologies habilitantes. À la suite du succès de celle-ci, l'EPITA a décidé en 2022 de créer une nouvelle majeure dédiée au quantique. C'est ainsi que la majeure 'Informatique & Technologies Quantiques' a été lancée en février 2023 en devenant la seizième majeure proposée par l'EPITA.

Les conditions nécessaires pour un élèves désirant suivre cette majeure sont : d'aimer les mathématiques, la physique et la programmation. Les élèves débutant leur deuxième année du cycle ingénieur possèdent d'excellentes compétences en développement logiciel à la suite de création de projets complexes impliquant l'apprentissage de différents langages de programmation. L'un des buts était aussi de faire simple et le plus agile possible : pas d'investissement dans du matériel coûteux (matériel d'optique par exemple) ou indisponible (ordinateurs quantiques). La consultation des entreprises principales du quantique nous a donné raison et nous avons donc centré cette formation autour du développement logiciel quantique avec une diversité d'architectures d'ordinateurs quantiques et un accès aux ordinateurs quantiques via les fournisseurs de cloud.

La formation détaillée

Pour répondre aux besoins de l'industrie, la majeure a été créée en partenariat avec les sociétés les plus importantes du quantique en France, en Europe et au niveau international. Cela a aussi permis de créer une formation avec une très grande majorité d'intervenants venant directement de l'industrie du quantique : vingt-deux sociétés ont accepté de rejoindre l'aventure avec l'aide d'universitaires renommés

Pendant de nombreuses années, les étudiants désirant apprendre le quantique s'engageaient dans des formations très théoriques dans lesquelles les principes fondamentaux de la physique quantique étaient enseignés afin de comprendre, entre autres, le comportement de la matière à l'échelle atomique, c'est-à-dire l'infiniment petit. Les prérequis principaux étaient d'avoir de bonnes connaissances d'analyse mathématique, d'algèbre linéaire et de physique classique. De nombreuses avancées technologiques fascinantes ont été découvertes grâce à cette théorie, telles que les lasers, l'imagerie médicale ou encore les nanotechnologies. Après un mastère, les élèves désirant approfondir le sujet, et peut-être faire de la recherche fondamentale, continuaient en doctorat.

Les dix dernières années ont vu l'arrivée des technologies de rupture (i.e. deeptech) dans l'intelligence artificielle, la biologie, la robotique, l'électronique, la photonique et, bien sûr, le quantique : cela a créé de nouveaux besoins et la nécessité d'inventer de nouveaux profils d'ingénieurs. Cela a fait évoluer les formations liées au quantique, tant d'un point de vu du nombre de talents nécessaires que du contenu des formations.

Certains problèmes ayant un niveau de complexité plus important nécessitent de passer d'un ordinateur classique à un supercalculateur afin d'être résolus. Mais certains problèmes complexes sont impossibles à résoudre même avec les supercalculateurs les plus puissants d'aujourd'hui, et c'est là que l'ordinateur quantique devrait apporter un avantage.

QUANTUM
EPITA

afin de créer l'un des programmes les plus complets et ambitieux, de France et d'Europe.
Sans compter le tronc commun, le programme se compose de plus de 600 heures d'enseignement purement quantique auquel s'ajoute des cours de cryptographie classique, de réseaux et de télécommunication (axé 5G advance afin de préparer la création de la 6G qui aura une partie quantique).
La majeure s'étend sur deux semestres puis est suivie par un stage de fin d'études de six mois. Les élèves ont également un projet en entreprise pendant les deux semestres qui leur permet de passer une demi-journée par semaine au sein d'une entreprise et d'y appliquer leurs connaissances.
La formation débute par une remise à niveau en mathématiques, physique classique / quantique et en parallèle commence une acculturation à l'écosystème des technologies quantiques avec une approche business. Il est très important que les élèves aient des exemples concrets tout au long du programme afin d'illustrer, comprendre et structurer leur apprentissage.
Une fois tous les concepts de base maitrisés, s'ensuivent les premiers développements en utilisant python et Q# avec une mise en application des enseignements des principaux algorithmes quantiques (Bernstein Vazirani, Grover, Shor, HHL...), auxquels s'ajoutent l'estimation de ressources, l'informatique quantique hybride et la cohabitation avec les ordinateurs à hautes performances. En parallèle les élèvent apprennent la théorie de l'information, la correction d'erreurs quantiques et sont initiés aux différentes architectures (qubit supraconducteur, qubit à ions piégés, qubit photonique, qubit silicium, qubit à base de nanotubes de carbone, qubit à atome neutre et qubit de chat).
Les élèves ont également la chance d'apprendre les différentes technologies de capteurs quantiques (atomes froids, centres NV, Josephson, terres rares et autres). L'accent est également mis sur l'importance de la cryptographie post-quantique, qui nécessite de comprendre les menaces liées à l'informatique quantique (en particulier bien comprendre l'algorithme de Shor) pour être mise en application. La compréhension de l'utilisation de la normalisation (NIST, ETSI...) et des rôles des propriétés intellectuelles et industrielle sont aussi enseignés.
Le développement logiciel quantique étant au cœur de la formation, il y a un enchainement d'utilisation de Qiskit, Braket, Q# ainsi que des plateformes Black Opal et Classiq. Nous mettons aussi l'accent sur l'apprentissage de l'utilisation d'émulateurs tels que Callisto et Perceval. Enfin l'enseignement de la cryptographie quantique, de la distribution et des solutions de déploiement de clefs quantiques mais aussi des solutions quantiques de génération de nombres aléatoires viennent consolider les connaissances des étudiants. L'enseignement du Quantum Machine Learning a aussi une place prépondérante dans la formation.
Pour conclure, les élèves ont également plusieurs enseignements couvrant les applications de l'informatique quantique et des algorithmes quantiques aux secteurs de la finance, de la logistique et de la pharmaceutique. Ce programme a été élaboré en collaboration avec les acteurs majeurs de l'industrie quantique et évoluera en fonction des besoins afin de former les meilleurs ingénieurs en informatique et technologies quantiques d'aujourd'hui et de demain.

Introduction à l'informatique quantique

Cet article est destiné à donner les premiers éléments de compréhension de l'informatique quantique. Il reprend largement des publications passées, à commencer par un article introductif publié dans Programmez! 239, mais aussi le support de présentation d'Introduction to Quantum Computing présenté lors de BarCamp Yerevan 2023. Son objectif est clairement de faire saisir les quelques notions-clés nécessaires pour commencer à explorer les différentes solutions quantiques disponibles.

Benoit Prieur

est auteur aux éditions ENI pour lesquelles il a publié une vidéo de vulgarisation de l'informatique quantique en 2023. Il a publié plusieurs articles de vulgarisation sur le sujet et participe régulièrement à des conférences quantiques. Il a également donné des cours d'informatique quantique en écoles d'ingénieurs.
https://www.benoit-prieur.fr/

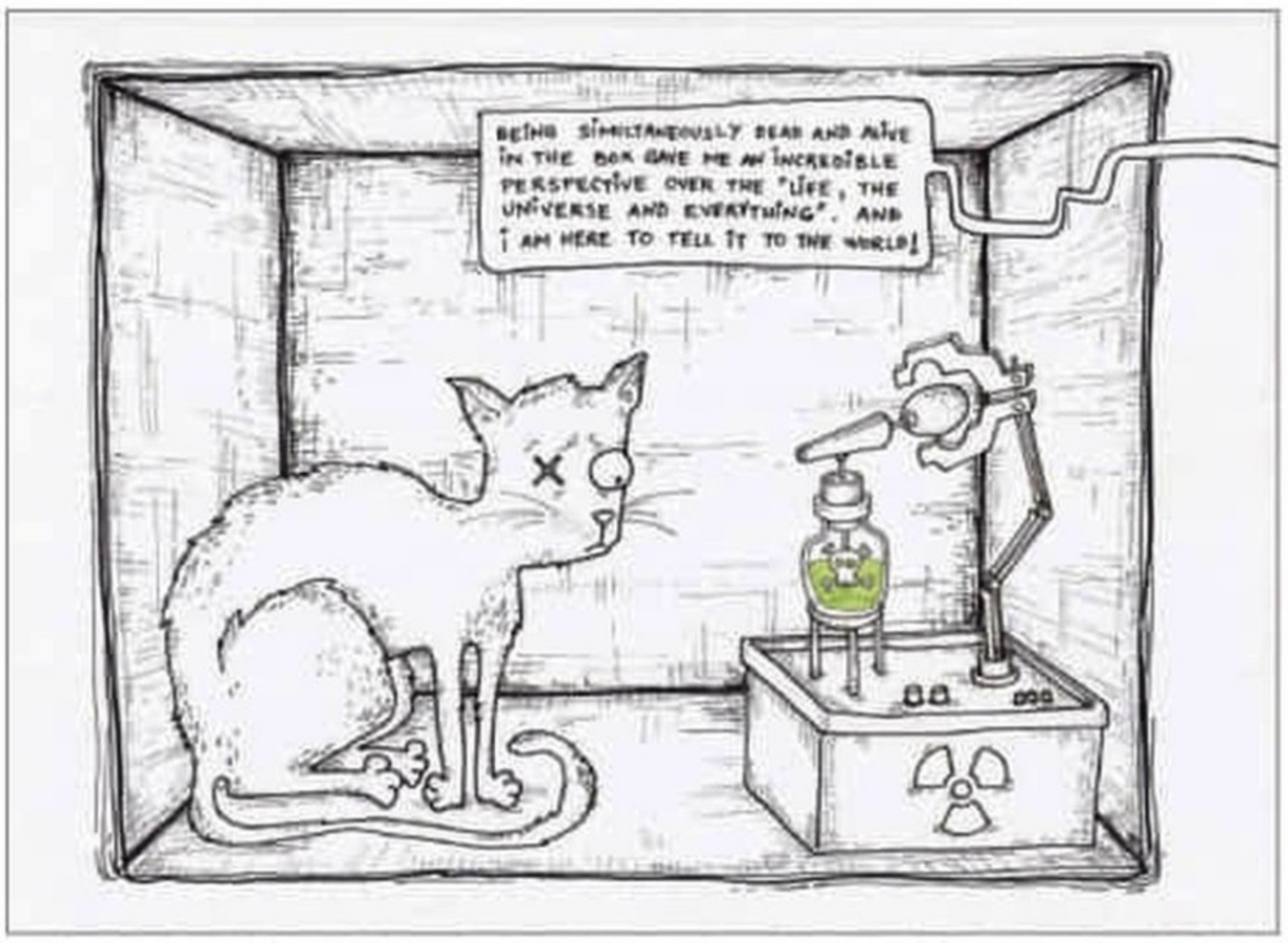
Premiers éléments de physique quantique

Considérations générales

L'informatique quantique se base sur quelques principes de physique quantique. La première question à évacuer est de définir la physique quantique. C'est la **physique de l'infiniment petit** au niveau nanoscopique. Exprimé autrement, c'est la physique des **atomes** et de ce qu'ils contiennent, en particulier les **électrons**. Elle couvre également les **photons**, qui sont des particules de lumière, et qui constituent l'aspect corpusculaire de la lumière, qui est également une onde. On parle d'ailleurs à propos de la lumière de la **dualité onde-corpuscule**.

La physique quantique se distingue également par son **caractère non intuitif**. Chaque personne a au contraire l'intuition, par exemple de la gravitation, dans le monde de la mécanique classique : si l'on pousse un objet posé sur une table, tout le monde a l'intuition que quand l'objet sera proche du rebord de la table, il tombera. Nous avons donc l'intuition de sa chute avant même qu'elle ne survienne, car nous avons une connaissance intuitive des lois de la mécanique classique ; a contrario chaque phénomène quantique échappe complètement à notre intuition.

Figure 1 :
Le chat Couscous dans la boîte, face au cruel dispositif.



Le dessin du chat de Schrödinger : Cristineagoe sur Wikipédia anglais, CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

Autre caractéristique de la physique quantique, sa **nature non déterministe**, ou **probabiliste**. Au contraire de la mécanique classique qui est, elle, déterministe. En effet, si on a toutes les données d'entrée, la position précise du ballon, sa vitesse, la force qui est exercée sur lui, dans quelle direction, on pourra calculer précisément sa nouvelle position. La mécanique classique est déterministe ; la mécanique quantique, elle, ne l'est pas. C'est une physique probabiliste.

Mentionnons enfin, le **principe d'Heisenberg** en physique quantique, qui explicite le fait que si on connaît, grâce à une mesure, une donnée relative à une particule, sa vitesse par exemple, on ne peut pas connaître une autre de ses caractéristiques, sa localisation précise par exemple. Ceci nous amène à la notion de **mesure quantique**, que nous expliciterons ci-après, qui est liée à une autre notion importante, la **réduction du paquet d'onde**, qui explicite que dans le monde quantique, après une mesure, une particule, ou un système physique en général, voit son état réduit à celui qui a été mesuré. En d'autres termes, la mesure quantique fait passer un état quantique de la physique quantique à la physique classique. De plus, on considère que la mesure quantique perturbe la valeur mesurée. C'est comme si dans le monde classique, quand vous mesurez la vitesse d'un véhicule à l'aide d'un radar routier, cette mesure modifie la vitesse dudit véhicule. Absolument contre-intuitif.

Détaillons plus avant cette notion d'état quantique, ainsi que les trois notions fondamentales pour commencer à faire de l'informatique quantique : la **superposition quantique**, l'**intrication quantique** et la **mesure quantique**, dont nous venons déjà de dire un mot.

Superposition, intrication et mesure quantiques

L'état quantique

Une particule se trouve dans un état quantique, constitué par une superposition quantique des différents états mesurables par une mesure quantique et pondéré chacun par la probabilité que cet état soit mesuré. En informatique classique, on parle de bit : l'état est 0 ou 1. Il faut voir un **état quantique** comme un état superposant à la fois un état 0 et un état 1, chacun pondéré par la probabilité respective d'être mesuré. Après mesure, l'état "classique" est égal strictement à 0 ou à 1. Cette première explication paraît sans doute obscure au premier abord, nous allons donc détailler l'**expérience de**

l'esprit dite du chat de Schrödinger pour rendre plus concret ce fonctionnement éminemment contre-intuitif.
Un mot avant cela à propos de la manière de représenter un état quantique, en informatique quantique : on parle de **bit quantique**, ou **qubit** ou **qu-bit**. Le bit quantique représente l'unité de base en informatique quantique à l'instar du bit en informatique dite classique.

L'expérience du chat de Schrödinger

L'expérience consiste à enfermer le chat Couscous (aka Cacahuète, aka Titus) dans une boîte opaque, et la refermer. Dans cette boîte se trouve un dispositif létal qui peut tuer le chat Couscous à n'importe quel moment. **Figure 1**
De l'extérieur, nous n'avons aucun moyen de savoir si le chat Couscous est vivant ou mort. En réalité, d'un point de vue quantique, il est à la fois mort et vivant, les deux états superposés (mort et vivant) étant équiprobables. L'état quantique de Couscous est donc une superposition de deux états : l'état mort et l'état vivant. Seule manière de savoir réellement ce qu'il en est : ouvrir la boîte, et ce faisant, procéder à une mesure quantique. L'état mesuré du chat Couscous sera soit mort, soit vivant.
On peut tenter de mettre en équation, cet état quantique, grâce à une notation très utilisée en informatique quantique, la **notation bra-ket**. L'état quantique de Couscous est représenté par cette combinaison linéaire impliquant l'état quantique mort, et l'état quantique vivant, chacun pondéré par une sorte de probabilité d'être la valeur finalement mesurée.

$$|\psi_{chat}\rangle = a_{vivant}.|vivant\rangle + a_{mort}.|mort\rangle$$

Si l'on considère les deux états superposés du chat (vivant et mort) comme étant équiprobables alors l'état quantique du chat s'écrit comme ci-dessous.

$$|\psi_{chat}\rangle = \frac{1}{\sqrt{2}}.|vivant\rangle + \frac{1}{\sqrt{2}}.|mort\rangle$$

Note sans importance pour la compréhension : le coefficient multiplicateur devant chaque état dans la précédente équation est égal à la racine carrée de la probabilité associée (ici chaque probabilité est égale à 0.5). En effet :

$$probabilité(|vivant\rangle) = probabilité(|mort\rangle) = (a_i)^2 = \frac{1}{2}$$

Figure 2

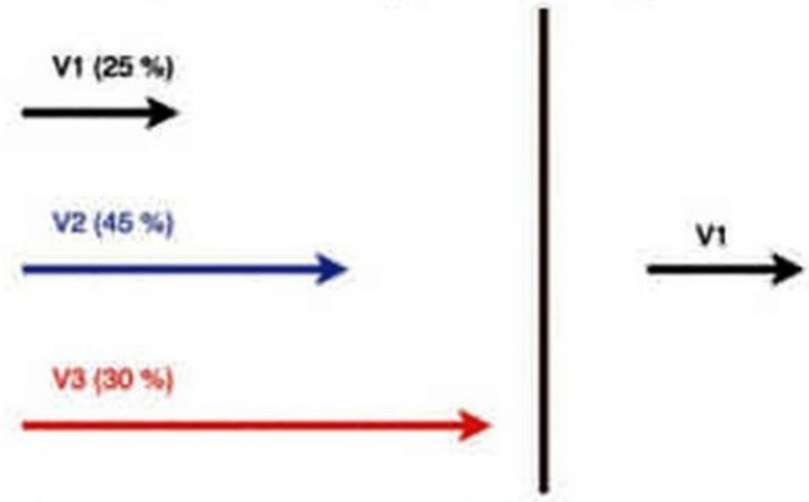
Cette expérience de la pensée permet de mettre en exergue deux notions-clés : la superposition quantique (quand le chat est dans un état superposé, à la fois mort et vivant) et la mesure quantique (quand on ouvre la boîte).

La notation bra-ket

Imaginons une particule donnée. On s'intéresse particulièrement à sa vitesse. Sa vitesse est une superposition quantique de trois vitesses différentes.
On peut écrire l'état quantique associée ainsi :

$$|\Psi\rangle = 25\%\ |V1\rangle + 45\%\ |V2\rangle + 30\%\ |V3\rangle$$

Modulo quelques approximations en termes de probabilité, on peut donc considérer que l'état quantique est une superposition de trois vitesses, V1, V2 et V3. La vitesse V1 a 25% de chance d'être mesurée, la vitesse V2 45% et la vitesse V3 30%. Une fois la mesure effectuée, ce ne sera qu'une des trois vitesses qui sera mesurée, et on se retrouvera alors dans le domaine de la mécanique classique.
Dans le schéma suivant, le segment vertical représente la mesure : une seule vitesse est consistante post-mesure, ici la vitesse V1, celle d'ailleurs qui était la moins probable.



Allons à présent un petit peu plus loin. On considère ici que l'on s'intéresse à un seul qubit. En notation bra-ket, son état quantique s'écrit ainsi :

$$|\psi\rangle = \alpha.|0\rangle + \beta.|1\rangle$$

On peut d'ailleurs écrire chacune des deux valeurs mesurables ainsi.

$$|0\rangle = 1.|0\rangle + 0.|1\rangle$$
$$|1\rangle = 0.|0\rangle + 1.|1\rangle$$

Si l'on passe à un schéma à deux qubits, on a quatre valeurs possibles mesurables. Ce sont les valeurs suivantes.

$$|00\rangle$$
$$|01\rangle$$
$$|10\rangle$$
$$|11\rangle$$

On peut en effet écrire l'état quantique à l'aide de l'équation suivante, utilisant les coefficients α, β, γ et δ.

$$|\psi\rangle = \alpha.|00\rangle + \beta.|01\rangle + \gamma.|10\rangle + \delta.|11\rangle$$

Comme précédemment on peut exprimer sous forme d'équations chacune des valeurs finalement mesurables.

$$|00\rangle = 1.|00\rangle + 0.|01\rangle + 0.|10\rangle + 0.|11\rangle$$
$$|01\rangle = 0.|00\rangle + 1.|01\rangle + 0.|10\rangle + 0.|11\rangle$$
$$|10\rangle = 0.|00\rangle + 0.|01\rangle + 1.|10\rangle + 0.|11\rangle$$
$$|11\rangle = 0.|00\rangle + 0.|01\rangle + 0.|10\rangle + 1.|11\rangle$$

Si l'on passe à trois qubits on aura huit valeurs possibles. De manière générale, une solution à n qubits verra la possibilité de travailler en 2 exposant n.

Figure 2 : *Le chat Couscous, bien vivant, mais courroucé par cette drôle d'expérience.*

On peut ensuite modifier l'état quantique de chaque qubit à l'aide d'un **circuit quantique**, qui représente ni plus ni moins un **algorithme quantique**. Ce circuit quantique est composé de **portes quantiques** et se termine en général par une mesure quantique.

L'informatique classique et son algèbre de Boole manipulent des portes logiques. En informatique quantique, on utilise des portes quantiques. Les états quantiques étant finalement des états probabilistes, on soumet en général plusieurs fois les mêmes entrées à un circuit quantique. On analyse ensuite la répartition du résultat des 10000 (par exemple) mesures, une mesure par utilisation du circuit quantique.

Le propos qui précède nous a permis d'éclaircir les notions de superposition et de mesure, ainsi que ce qui lie ces deux notions. Il nous reste à aborder la troisième grande notion : l'intrication (ou l'enchevêtrement) quantique.

L'intrication quantique

L'intrication quantique est un phénomène physique assez étonnant dont l'existence fut longtemps controversée au sein de la communauté scientifique. Il a été depuis démontré expérimentalement bien après sa mise en évidence théorique.

L'**intrication quantique** consiste en un lien entre deux particules, pourtant possiblement très éloignées l'une de l'autre. Deux particules A et B qui sont intriquées ont le même état quantique. Si l'une des deux change d'état, l'état de la seconde particule sera le même. Cela implique enfin que le résultat de la mesure respective de deux particules intriquées est théoriquement le même.

Cette notion est fréquemment utilisée en informatique quantique, pour coder ce que l'on appelle la **téléportation quantique** ou encore pour simuler les **états de Bell,** qui seront l'objet de notre exemple pratique avec la solution **IBM Qiskit**.

Les états de Bell

Un des moyens d'implémenter l'intrication de deux particules, ici entre deux qubits, est de recourir aux états de Bell. Ci-dessous les quatre états de Bell :

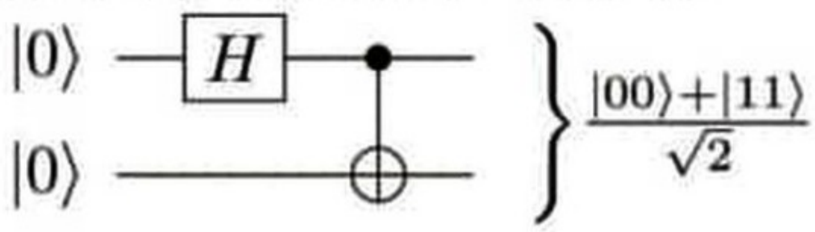
$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B + |1\rangle_A \otimes |1\rangle_B) \text{ (1)}$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |0\rangle_B - |1\rangle_A \otimes |1\rangle_B) \text{ (2)}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B + |1\rangle_A \otimes |0\rangle_B) \text{ (3)}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B - |1\rangle_A \otimes |0\rangle_B) \text{ (4)}$$

Ceux-ci permettent de maximiser l'intrication entre les deux qubits. Un circuit quantique bien déterminé et bien connu permet d'implémenter un des quatre états de Bell.



Il implique l'utilisation de deux portes quantiques, la porte de Hadamard et la porte C-NOT et se termine par une mesure de chacun des deux qubits.

La plate-forme IBM QisKit

La plate-forme permet de s'essayer à l'informatique quantique de plusieurs manières. En premier lieu, Qiskit s'installe et s'utilise localement indifféremment sous Linux, Windows ou macOS. L'URL suivante détaille la procédure à suivre pour installer QiSkit sur votre machine.

```
https://qiskit.org/documentation/install.html
```

La plate-forme permet également de travailler entièrement en ligne. C'est la solution que nous utiliserons ici. D'une part est disponible un éditeur graphique de circuits quantiques nommé Circuit Composer (nous en dirons un mot) et d'autre part, il est possible de coder et d'exécuter tous ses programmes quantiques en Python dans des notebooks Jupyter que nous allons également utiliser ici.

Une fois défini votre circuit quantique (soit avec Circuit Composer soit au sein d'un notebook Jupyter) on peut exécuter le programme. IBM Qiskit permet de soumettre le programme à deux types de solveurs quantiques.

- des simulateurs de machines quantiques conçus avec un ordinateur classique qui sont supposés donner un résultat exact d'un point de vue quantique.
- de vraies machines quantiques. L'exécution se retrouve alors dans une file d'attente

Note : *les notebooks Jupyter permettent la programmation interactive faisant alterner du code Python et le résultat de son exécution. L'extension d'un fichier de notebook Jupyter est .ipynb. Un tel fichier peut facilement être partagé, y compris sur des sites de gestion de version, comme Github, qui savent afficher un fichier .ipynb.*

Pour utiliser Qiskit en ligne, commencez par vous créer un compte IBM en ligne à l'URL suivante.

```
https://quantum-computing.ibm.com/
```

Une fois votre compte créé, vous accédez à votre espace, muni d'une barre latérale à gauche de l'écran qui vous permet de naviguer parmi les différents outils mis à disposition. Ci-contre une copie d'écran de la barre latérale incluant les liens suivants.

- Dashboard qui fait office de page d'accueil.
- Results qui permet de consulter les exécutions passées et leurs résultats.
- Circuit Composer, qui permet de réaliser un circuit quantique de manière graphique.
- Qiskit Notebooks, qui permet de travailler avec des notebooks Jupyter. **Figure 3**

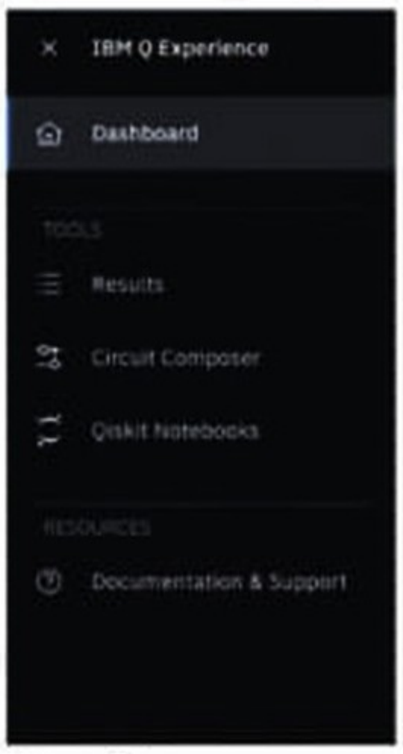


Figure 3

Le Circuit Composer de Qiskit

Cet outil permet de composer un circuit quantique en faisant glisser les différentes portes quantiques de notre programme. Ainsi en quelques clics on réalise le schéma suivant en faisant glisser une porte de Hadamard (H), une porte C-NOT ainsi que deux portes de mesure. **Figure 4**

Au-dessus du schéma ainsi composé se trouve un bouton Run dans lequel on peut sélectionner un solveur ainsi que le nombre d'utilisations du circuit quantique. Nous choisissons donc de lancer 1024 essais à l'aide d'un des simulateurs quantiques mis à disposition.

Après quelques secondes on obtient un résultat graphique qu'il s'agit d'interpréter. **Figure 5**

On voit qu'on obtient 49,121 % des essais (503 essais) qui donnent le résultat suivant dans lequel les deux qubits de sortie sont identiques (à zéro) :

$$|00\rangle$$

On obtient également 50,879 % des essais (521 essais) qui donnent le résultat suivant dans lequel les deux qubits de sortie sont identiques (à un) :

$$|11\rangle$$

Nous n'obtenons jamais, comme prévu, les deux situations suivantes :

$$|10\rangle$$

$$|01\rangle$$

Les notebooks Jupyter dans Qiskit

Passons à présent à l'utilisation de Jupyter. Nous allons reproduire le même exemple, mais cette fois avec du code Python. Nous lancerons une simulation similaire à celle qui précède puis nous exposerons notre programme quantique sur une vraie machine quantique située à Melbourne. Commençons à créer un nouveau notebook Jupyter en cliquant sur le bouton New Notebook.

On commence ensuite à coder une première section du notebook que nous détaillons tout de suite. On ajoute la clause %matplotlib inline nécessaire dès l'instant que l'on travaille avec des notebooks Jupyter.

```
%matplotlib inline
```

Puis on déclare les différents modules dont on aura besoin, à commencer évidemment par le module qiskit. On importe également numpy et plot_histogram pour pouvoir ensuite générer un résultat graphique.

```
import qiskit
from qiskit import(
 IBMQ,
 ClassicalRegister,
 QuantumCircuit,
 QuantumRegister,
 QuantumCircuit,
 execute,
 Aer)
```

```
import numpy as np
```

```
from qiskit.visualization import plot_histogram
```

Puis on définit notre circuit quantique à l'aide de la fonction QuantumCircuit dont la documentation est à cette URL.

```
https://qiskit.org/documentation/api/qiskit.circuit.QuantumCircuit.html
```

On a besoin de deux bits quantiques en entrée et de deux bits classiques de sortie.

```
circuit = QuantumCircuit(2, 2)
```

Puis on adjoint une porte de Hadamard sur le premier bit quantique.

```
circuit.h(0)
```

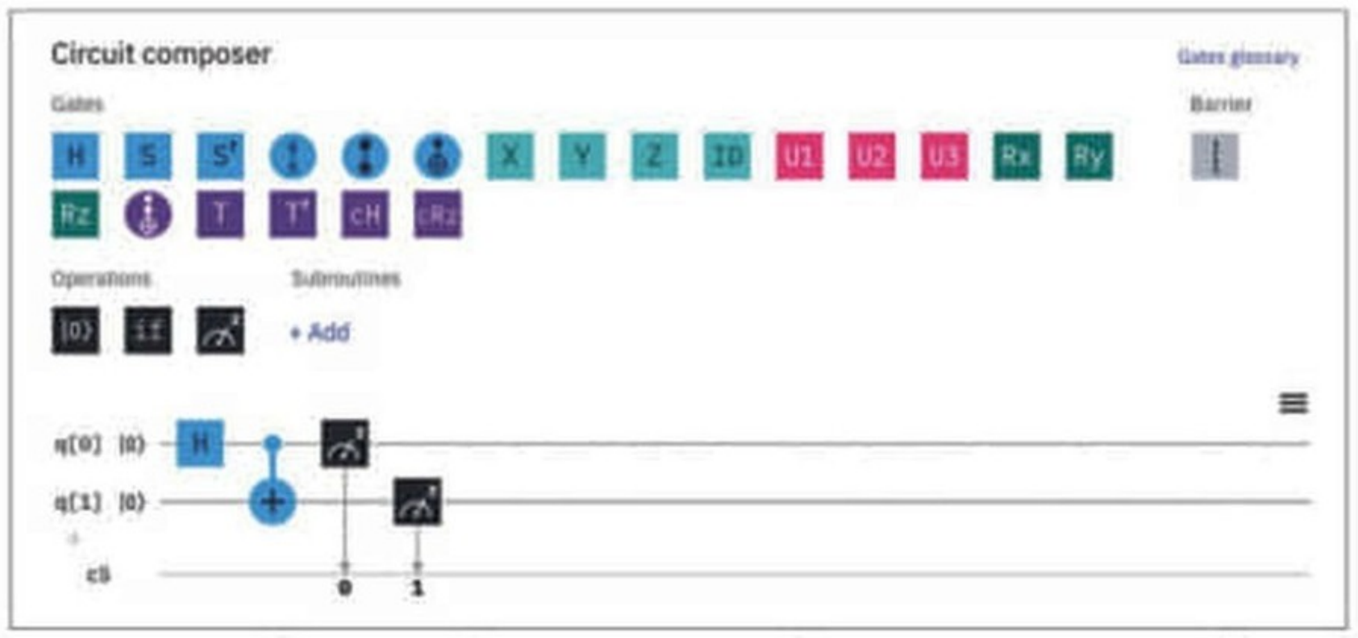


Figure 4

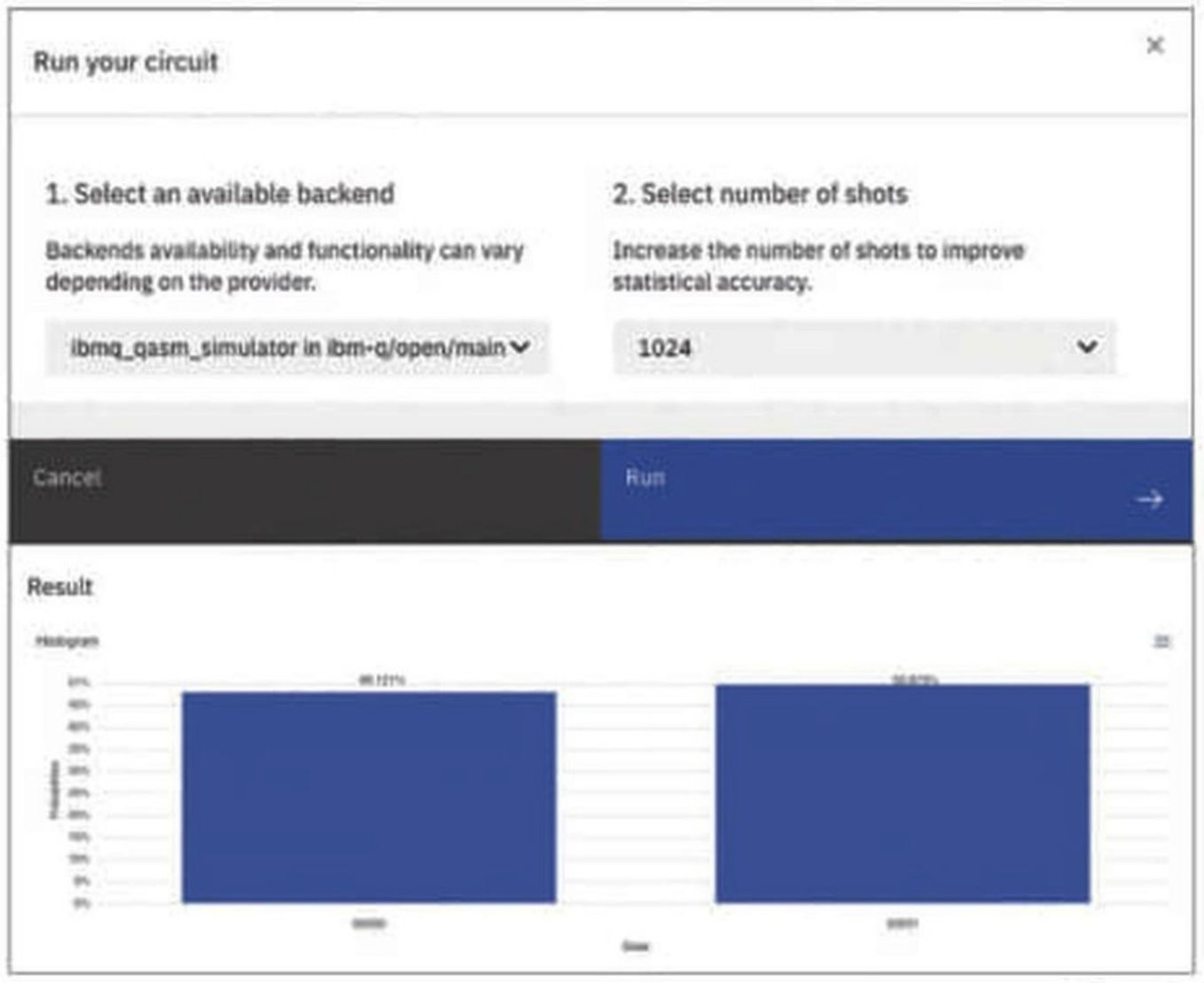


Figure 5

On ajoute ensuite notre porte quantique C-NOT entre le premier et le deuxième bit quantique.

```
circuit.cx(0, 1)
```

Puis on ajoute les deux unités de mesure quantique.

```
circuit.measure([0,1], [0,1])
```

Pour l'instant nous avons seulement défini un circuit quantique que nous allons dessiner à l'écran avant de l'exécuter.

```
circuit.draw()
print(circuit)
circuit.draw(output='mpl', filename='circuit.png')
```

La copie d'écran de la première section de notre notebook. **Figure 6**

Quand on l'exécute, on obtient deux représentations de notre circuit quantique. **Figure 7**

Nous allons à présent exécuter ce programme quantique. La seconde section du notebook concerne l'exécution du programme sur un simulateur quantique, puis la troisième section concerne l'exécution sur une réelle machine quantique.

Voici le code commenté de la deuxième section.
On déclare un simulateur qui sera chargé de l'exécution.

```
simulator = Aer.get_backend('qasm_simulator')
```

On exécute notre circuit quantique pour 1000 essais.

```
job = execute(circuit, simulator, shots=1000)
```

On obtient le résultat de la simulation.

```
result = job.result()
counts = result.get_counts(circuit)
```

Enfin on affiche un histogramme synthétisant les résultats.

```
print("\nRésultats :", counts)
plot_histogram(counts)
```

La copie d'écran de la deuxième section de notre notebook ainsi que son résultat obtenu quasi immédiatement.

Figure 8

Les résultats uniquement en 00 ou en 11 confirment que l'on est bien en intrication maximale de manière parfaite (en effet aucun artefact 01 ou 10 n'est présent même de façon minime dans les résultats). Passons à présent à la troisième section qui utilise une vraie machine quantique. On déclare cette machine quantique localisée à Melbourne.

```
provider = IBMQ.get_provider(group='open')
device = provider.get_backend('ibmq_16_melbourne')
```

Puis on exécute notre circuit quantique sur cette machine.

```
job_exp = execute(circuit, device, shots=1024)
```

Et comme dans la deuxième section on obtient puis on met en forme les résultats.

```
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)
print("\nRésultats expérimentaux:", counts_exp)
plot_histogram(counts_exp)
```

Les résultats ne sont cette fois pas immédiats. Ce nouveau calcul est placé dans la file d'attente de la machine – nous ne sommes pas les seuls à la solliciter. On peut voir peu après son lancement, son état et sa présence dans la file d'attente dans l'onglet Results. **Figure 9**

La copie d'écran de la troisième section de notre notebook ainsi que son résultat obtenu après quelques minutes.

Figure 10

On voit donc que la machine quantique fonctionne plutôt bien, mais a quelques défauts et approximations. Ainsi un peu moins de 6% des tirages ont conduit à obtenir 01 ou 10, ce qui est en contradiction avec la définition de cet état de Bell. Le simulateur quantique utilisé précédemment ne produisait pas de tels artefacts. Certes cette exécution montre un défaut (faible), mais elle a été lancée sur une vraie machine quantique, ce qui est tout à fait exaltant.

```
%matplotlib inline
import qiskit
from qiskit import(
  IBMQ,
  ClassicalRegister,
  QuantumCircuit,
  QuantumRegister,
  QuantumCircuit,
  execute,
  Aer)
import numpy as np
from qiskit.visualization import plot_histogram

circuit = QuantumCircuit(2, 2)
circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0,1], [0,1])

# Circuit
circuit.draw()
print(circuit)
circuit.draw(output='mpl', filename='circuit.png')
```

Figure 6

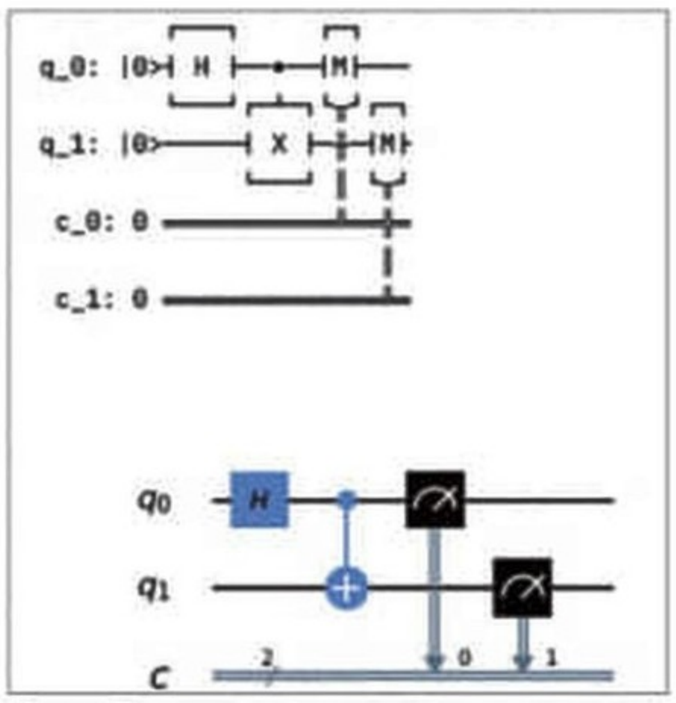


Figure 7

```
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1000)
result = job.result()
counts = result.get_counts(circuit)
print("\nRésultats :", counts)
plot_histogram(counts)
```

```
Résultats : {'11': 472, '00': 528}
```

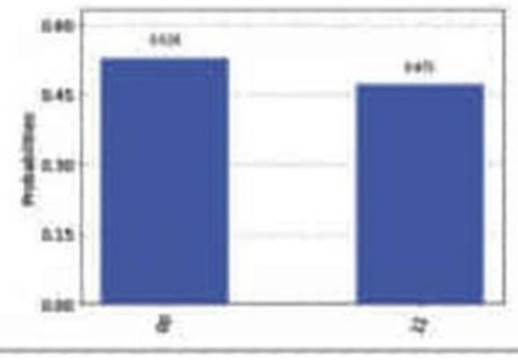


Figure 8

```
# Machine quantique
provider = IBMQ.get_provider(group='open')
device = provider.get_backend('ibmq_16_melbourne')
job_exp = execute(circuit, device, shots=1024)
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)
print("\nRésultats expérimentaux:", counts_exp)
plot_histogram(counts_exp)
```

```
Résultats expérimentaux: {'10': 28, '11': 449, '00': 519, '01': 28}
```

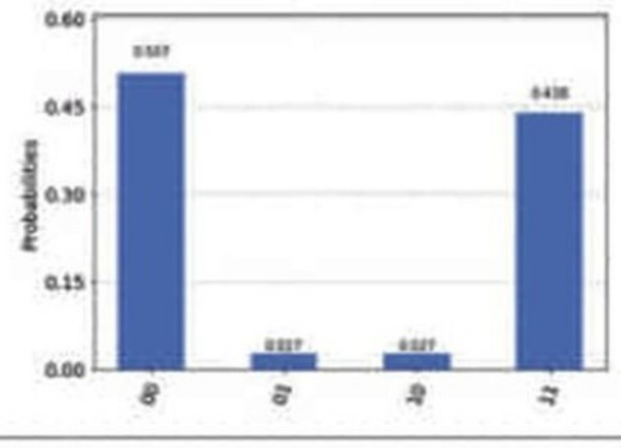


Figure 10

Figure 9

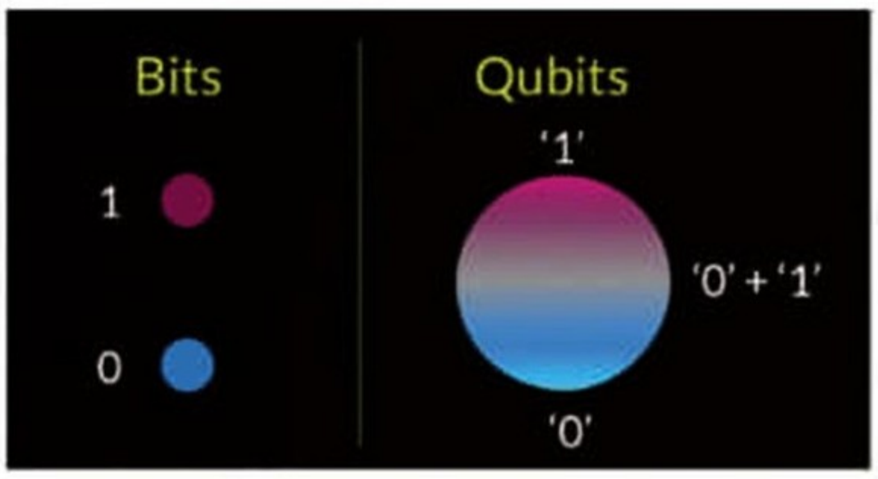
Programmation quantique : naviguer dans le futur de l'informatique

Tamuz Danzig
Ingénieur software quantique, et CTO Office chez Classiq

Dans l'univers de l'informatique quantique, les projecteurs se braquent souvent sur le hardware (ou le matériel) : qubits, circuits supraconducteurs, etc. Mais il est temps de porter notre attention sur le véritable héros de cette révolution : le software quantique, le maestro silencieux orchestrant la symphonie des qubits. Les logiciels quantiques jouent un rôle essentiel, allant de la transformation d'algorithmes quantiques abstraits en code exécutable à l'optimisation des conceptions de circuits.

En abordant les bases de la programmation quantique, on découvre rapidement ses nuances par rapport à l'informatique classique. Les langages quantiques, avec leur rôle distinct, offrent une perspective fascinante sur cette technologie en plein essor.

Programmation quantique vs. Programmation classique : les différences fondamentales



Au cœur de l'univers de l'informatique quantique, il existe des différences radicales de celui de l'informatique classique. Et elles vont bien au-delà du hardware et touchent l'essence même de la programmation.

Les ordinateurs classiques, que la plupart d'entre nous utilisent au quotidien, fonctionnent sur des données binaires. Cela signifie qu'ils traitent l'information en "bits", qui sont soit dans un état 0 soit dans un état 1. Les programmes classiques tournent donc autour de la manipulation de ces bits à l'aide d'opérations logiques.

Les ordinateurs quantiques, en revanche, fonctionnent différemment. Ils exploitent les particularités de la physique quantique pour traiter l'information via des "qubits". Contrairement aux bits, un qubit peut exister dans plusieurs états simultanément, grâce à un phénomène appelé superposition. De plus, les qubits peuvent également être intriqués, ce qui signifie que l'état d'un qubit peut affecter instantanément l'état d'un autre, quelle que soit la distance qui les sépare.

Par conséquent, programmer un ordinateur quantique nécessite une nouvelle approche, une nouvelle logique et un tout nouvel ensemble de langages de programmation. Les développeurs de logiciels quantiques ne donnent pas simplement une séquence d'opérations ; ils chorégraphient une danse de qubits, exploitant les propriétés particulières de la physique quantique pour résoudre des problèmes complexes. La beauté de la programmation quantique réside dans sa capacité à tisser un ballet de superpositions et d'intrications pour obtenir des solutions exponentiellement plus rapides que l'informatique classique.

L'informatique quantique ne remplace pas l'informatique classique. Elle la complète, en abordant des problèmes actuellement insolubles avec des ordinateurs classiques en raison du type de calcul et de sa complexité. Le logiciel quantique nécessite donc une solide compréhension des principes classiques et quantiques pour exploiter efficacement les forces de chacun et naviguer à travers leurs défis respectifs.

Les fondamentaux de la programmation quantique

La programmation quantique requiert un lexique distinct.

Celui-ci permet de traiter les éléments constitutifs d'un programme quantique, et nous aide à décrire et à naviguer dans l'univers multidimensionnel de l'informatique quantique.

Trois termes doivent être mis en évidence, car ils servent de base à la compréhension de la programmation quantique : les portes quantiques, les circuits quantiques et les algorithmes quantiques.

- **Portes Quantiques** : Tout comme les ordinateurs classiques utilisent des portes logiques (ET, OU, NON), les ordinateurs quantiques fonctionnent avec des portes quantiques. Mais contrairement à leurs homologues classiques, les portes

Electronic circuit design

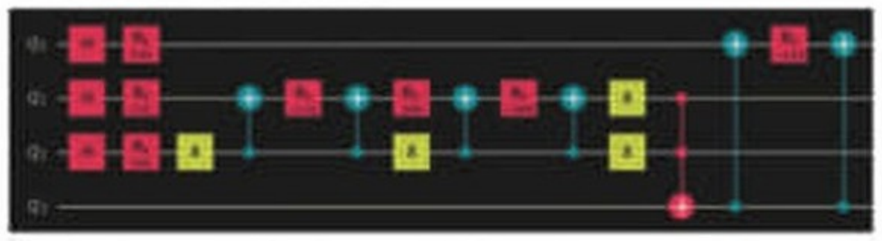
input x	input y	output
0	0	0
0	1	1
1	0	1
1	1	0

Quantum circuit Design

input x	input y	output
|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|1⟩
|1⟩	|0⟩	|1⟩
|1⟩	|1⟩	|0⟩

quantiques sont réversibles et traitent des probabilités. Elles manipulent l'état des qubits pour effectuer des opérations quantiques. Quelques exemples incluent les portes Pauli-X, Pauli-Y, Pauli-Z, Hadamard et CNOT.

- **Circuits Quantiques** : Une séquence de portes quantiques forme un circuit quantique. Le circuit quantique définit les transformations que subissent les qubits pour résoudre un problème donné. Cependant, le comportement du circuit est intrinsèquement probabiliste en raison de la nature de la physique quantique.

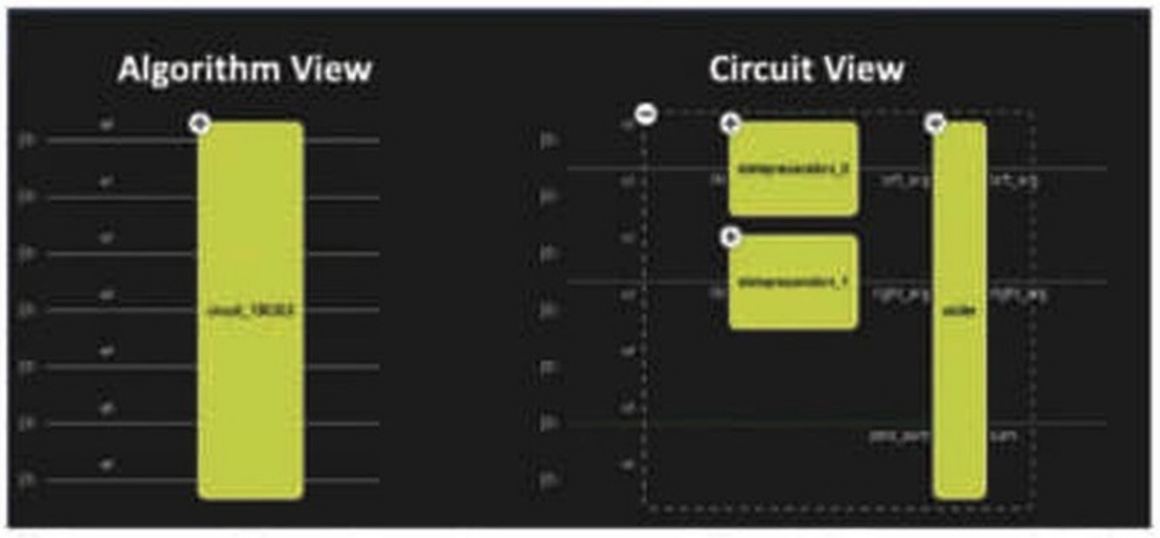


- **Algorithmes Quantiques** : Les algorithmes quantiques sont des séquences de circuits quantiques conçus pour effectuer une tâche spécifique ou résoudre un problème spécifique, tout comme une séquence d'instructions forme un algorithme classique. Certains algorithmes quantiques populaires incluent l'algorithme de Shor pour la factorisation de grands nombres, et l'algorithme de Grover pour la recherche dans des bases de données non triées. Les algorithmes quantiques exploitent les phénomènes de superposition et d'intrication pour surpasser les algorithmes classiques pour certains types de problématiques.

Dans le domaine de la programmation quantique, nous concevons essentiellement une séquence chorégraphiée qui manipule les qubits à travers ces portes quantiques, formant des circuits quantiques pour exécuter des algorithmes quantiques. Le tout, pour résoudre des problèmes que les machines classiques trouvent insurmontables. **Figures A**

Le paysage de la programmation quantique

Le monde de la programmation quantique est aussi diversifié que l'ensemble des problèmes qu'il cherche à résoudre. Divers langages de programmation quantique et plateformes logicielles ont émergé pour répondre à différents besoins, chacun avec sa propre approche et ses points forts. Ce riche paysage comprend :

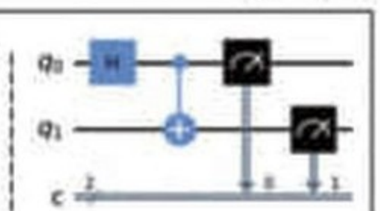


Figures A

- **Des langages de programmation quantique** : tout comme l'informatique classique a son C++, Python et Java, l'informatique quantique a également ses langages. Par exemple, Q# de Microsoft et Qiskit d'IBM sont deux des langages de programmation quantique les plus populaires aujourd'hui. Ils permettent de définir et de manipuler des états quantiques, d'appliquer des portes quantiques et de mesurer les résultats.

Dans ce code qiskit, un registre quantique de deux qubits est créé. Ensuite, une porte de Hadamard est appliquée au premier qubit et une porte CNOT est utilisée sur les deux qubits. À la fin, une mesure est effectuée sur chaque qubit.

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])
qc.draw('mpl')
```



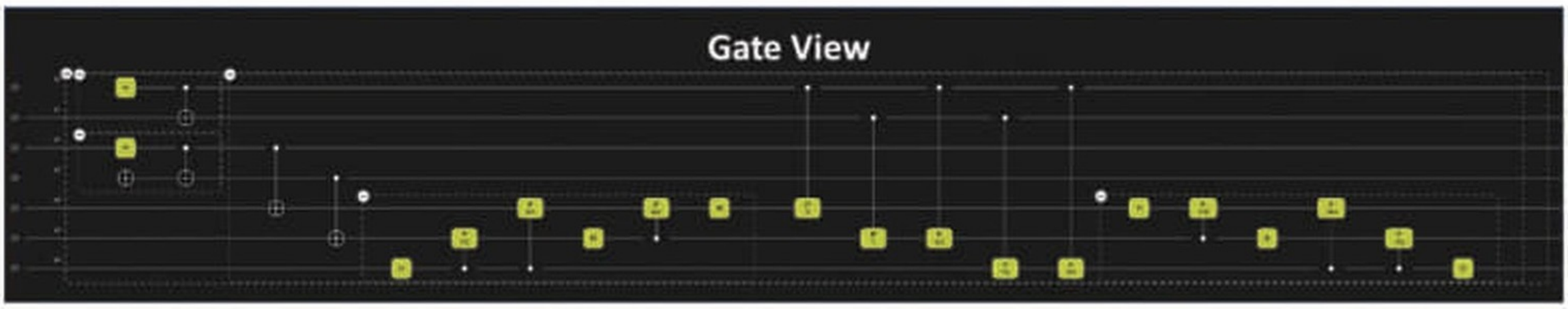
- **Des plateformes logicielles** : outre les langages de programmation autonomes, il existe des plateformes logicielles conçues pour aider au développement quantique. Par exemple, notre plateforme offre une manière intuitive et visuelle de concevoir des circuits et des algorithmes quantiques. C'est cette abstraction de haut niveau qui permet aux développeurs quantiques, débutants et experts, d'exploiter la puissance de cette technologie sans se perdre dans les détails techniques de la définition des portes.

Il est à noter que chaque outil et langage a ses points forts, et le choix dépend souvent de la problématique abordée. Il s'agit de choisir le bon outil pour la bonne mission, tout comme pour l'informatique classique.

Le processus de la programmation quantique

Bien que programmer un ordinateur quantique puisse sembler intimidant au départ, le processus reste relativement classique :

1. **La formulation de la problématique** : la première étape de la programmation quantique consiste à définir le problème que vous souhaitez résoudre. Il peut s'agir d'optimiser un portefeuille financier, de simuler une réaction chimique ou de casser un code de cryptage. Il est essentiel de comprendre que toutes les problématiques ne sont pas adaptées aux solutions quantiques. Certaines tâches peuvent être traitées plus efficacement par des ordinateurs classiques. Par conséquent, bien identifier les problématiques devient crucial.



2 La sélection de l'algorithme : une fois la problématique définie, l'étape suivante consiste à choisir un algorithme quantique capable de la résoudre. Il existe une bibliothèque croissante d'algorithmes quantiques, chacun conçu pour traiter un type de problème particulier. Certains algorithmes sont bien adaptés aux tâches d'optimisation, tandis que d'autres sont conçus pour la simulation ou le Machine Learning.

3 L'implémentation : une fois la problématique définie et l'algorithme en main, il devient possible de passer à l'implémentation. C'est là que les langages de programmation quantique et les plateformes entrent en jeu. Il faudra traduire l'algorithme choisi en code quantique à l'aide du langage ou plateforme sélectionnés. C'est souvent la partie la plus technique du processus, et elle peut impliquer des tâches complexes comme la conception de circuits et la gestion des états quantiques.

4 L'exécution et l'analyse : enfin, nous arrivons à l'exécution du programme quantique sur un ordinateur quantique ou un simulateur. L'analyse des résultats devient palpable. Comme cette technologie est probabiliste, il est nécessaire d'exécuter le programme plusieurs fois pour obtenir un résultat statistiquement significatif. L'analyse implique souvent d'interpréter les résultats quantiques dans le contexte de la problématique initiale.

Tout comme apprendre à programmer au sens classique, le chemin pour devenir compétent en programmation quantique nécessite de la pratique, de la patience et beaucoup de curiosité. **Figure B**

L'avenir de l'informatique quantique

Les implications de l'informatique quantique sont vastes et prometteuses. À mesure que nous affinerons notre capacité à exploiter et manipuler les phénomènes quantiques, nous verrons cette prouesse déverrouiller des solutions à certains des défis les plus complexes et actuellement insolubles du monde.

Innovation dans de multiples domaines :

L'informatique quantique a le potentiel de révolutionner diverses industries. Les entreprises pharmaceutiques, par exemple, pourraient utiliser des systèmes quantiques pour simuler et analyser des structures moléculaires complexes, conduisant à de nouvelles découvertes de médicaments. Le secteur financier pourrait exploiter des algorithmes quantiques pour une meilleure évaluation des risques, une optimisation de portefeuille et une détection des fraudes.

Sécurité améliorée des données :

La perspective que les ordinateurs quantiques brisent les méthodes de cryptage actuelles est préoccupante, mais elle présente également une opportunité. À mesure que nous progresserons dans l'informatique quantique, nous développerons simultanément des techniques de cryptage résistantes, créant ainsi une nouvelle ère de sécurité des données.

Découverte scientifique :

L'informatique quantique promet de dynamiser la découverte scientifique. Dans des domaines tels que la science des matériaux, les simulations quantiques peuvent faciliter la découverte de nouveaux matériaux aux propriétés souhaitées. Pour l'environnement, elle pourrait offrir des prédictions climatiques plus précises en modélisant mieux les systèmes complexes.

Bien que ces possibilités excitantes se profilent à l'horizon, il est important de se rappeler que le périple quantique ne fait que commencer. Il s'agit d'un domaine propice à l'exploration et à l'innovation.

À mesure que nous passerons de la théorie à la pratique, de l'abstraction à l'application, la programmation quantique deviendra une pierre angulaire de l'évolution technologique. En apprenant les principes de cette science aujourd'hui, nous nous préparons non seulement à un avenir alimenté par le quantique, mais nous posons également les fondements et participons activement à sa création.

Naviguer dans le royaume quantique peut sembler intimidant, mais avec la bonne compréhension et les bons outils, chacun a le potentiel de laisser sa marque dans ce domaine passionnant. Nous n'en sommes qu'au début de la démystification des subtilités du software quantique, et chaque jour nous apporte une meilleure compréhension du potentiel révolutionnaire de l'informatique quantique. Cette technologie, bien qu'encore à ses balbutiements, promet de bouleverser notre manière de traiter et d'interpréter les données, ouvrant la voie à des avancées sans précédent dans divers domaines, de la recherche fondamentale à l'industrie. En plongeant profondément dans ses mystères, nous nous rendons compte que les défis sont nombreux, mais les opportunités le sont tout autant. L'avenir de l'informatique quantique pourrait bien redéfinir non seulement notre compréhension de l'informatique, mais aussi la manière dont nous abordons et résolvons les défis les plus complexes de notre époque.

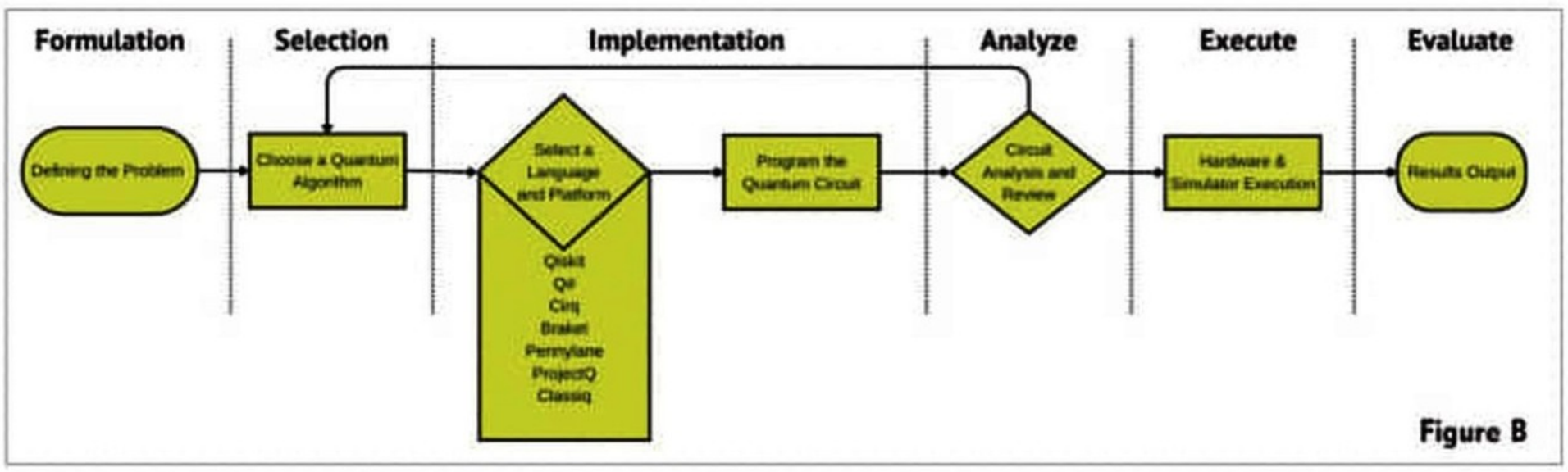


Figure B

James Clarke (JC),

Director of Quantum Hardware at Intel Corporation

Jim Clarke est le directeur du groupe de recherche Quantum Hardware au sein de l'Intel Components Research Organization. Auparavant, Jim a dirigé un groupe axé sur la recherche et les performances des interconnexions au niveau des nœuds technologiques avancés. Il est co-auteur de plus de 100 articles et possède plus de 150 brevets. Avant de rejoindre Intel en 2001, Jim a obtenu un B.S. en informatique et en chimie à l'Université d'Indiana. Il est membre de l'IEEE.

intel

INTERVIEW DE JIM CLARKE

Réinventer le traitement des données avec l'informatique quantique

Tunnel Falls est la puce qubit de spin en silicium la plus avancée d'Intel à ce jour et s'appuie sur des décennies d'expertise de l'entreprise en matière de conception et de fabrication de transistors. La sortie de la nouvelle puce constitue la prochaine étape de la stratégie à long terme d'Intel visant à construire un système informatique quantique commercial complet. Même s'il reste encore des questions et des défis fondamentaux à résoudre avant de parvenir à un ordinateur quantique tolérant aux pannes, la communauté universitaire peut désormais explorer cette technologie et accélérer le développement de la recherche.

Bonjour Jim, pouvez-vous vous présenter à nos lecteurs ?

JC : Je m'appelle James Clarke, je suis le directeur du matériel Quantique chez Intel. Je suis ici depuis près de 23 ans et j'ai occupé divers postes. En 2014, j'ai indiqué aux dirigeants d'Intel que nous avions besoin d'un programme d'informatique quantique, et je gère cela depuis.

Il n'est pas toujours simple d'expliquer à quoi va servir un ordinateur quantique. Pouvez-vous donner des exemples de domaines dans lesquels l'informatique quantique sera utile ?

JC : Le premier exemple qui me revient est basé sur ce qu'on appelle l'algorithme de SHOR. Il a été développé par Peter Shor, professeur dans les années 90, et il s'agissait essentiellement d'un algorithme utilisé pour factoriser de grandes clés de cryptographie. Ainsi, vous pouvez utiliser un ordinateur quantique pour déchiffrer des e-mails ou des transactions commerciales. C'est dans une certaine mesure la raison pour laquelle tous les gouvernements de la planète aimeraient disposer d'un ordinateur quantique. Au-delà de cela, nous pensons que les ordinateurs quantiques seront utiles pour le développement de matériaux, la chimie, l'analyse financière, le changement climatique et l'optimisation. Ce sont des applications qui sont actuellement très difficiles à étudier avec un ordinateur ou un supercalculateur hautes performances.

Pourquoi Intel investit-il dans l'informatique quantique ?

JC : Intel est une société informatique. L'informatique quantique est une forme de calcul haute performance ou de calcul de centre de données, et il est donc naturel qu'Intel y travaille. Le type d'ordinateur quantique qu'Intel étudie est très similaire à nos transistors en se basant sur des Qubits à base de silicium. Nous utilisons ces connaissances et notre expertise et essayons de les appliquer aux ordinateurs quantiques dans la fabrication de puces quantiques.

Quelle est la stratégie d'Intel autour de l'informatique quantique ?

JC : La stratégie d'Intel à court terme est de développer un ordinateur quantique full stack comprenant du matériel et des logiciels. Pour ce faire, nous utilisons nos usines pour nous aider à fabriquer des puces quantiques. Nous avons récemment annoncé Tunnel Falls, une puce à 12 spin-qubits. En outre, nous développons un système électronique de contrôle quantique qui fonctionnerait à très basse température et nous permettrait de contrôler les puces qubit. Et au printemps dernier, nous avons publié le kit de développement Intel Quantum Software permettant aux développeurs d'apprendre à programmer un ordinateur quantique.

Revenons au matériel, pouvez-vous expliquer ce que sont les « spin qubits » et pourquoi Intel va dans cette direction ?

JC : Quand je décris ce qu'est un ordinateur quantique, je décris souvent une pièce de monnaie qui tourne. Imaginez si vous avez une pièce de monnaie dans votre main et que vous regardez cette pièce, c'est pile ou face, c'est l'un ou l'autre. Mais si j'ai une pièce de monnaie et qu'elle tourne sur une table, elle peut représenter deux états en même temps. Cette pièce est à la fois pile et face. Si j'ai plusieurs pièces, j'ai une combinaison de pile et face pour chacune d'elles. C'est un nombre exponentiel d'états pour le nombre de pièces que je possède. Si j'avais 50 pièces qui tournaient sur une table en même temps, je pourrais en représenter 250 états, ce qui représente plus d'états que n'importe quel superordinateur sur Terre. Bien entendu, nous n'utilisons pas de pièces ici. Nous utilisons les pièces comme métaphore pour les qubits. Certaines personnes utilisent des qubits supraconducteurs. Chez Intel, c'est le cas et nous développons aussi des « spin qubits ». Nous prenons un transistor, mais au lieu de faire passer un courant d'électrons à travers le dispositif, nous avons un seul électron dans le dispositif à très basse température. Un électron a soit un état de rotation ascendant, soit un état de rotation descendant. C'est une propriété fondamentale de l'électron. Cet état de rotation ascendant ou descendant est essentiellement les deux faces d'une pièce de monnaie, et donc ce que nous faisons est de créer le qubit et de développer ces deux états, nous enchaînons ensemble un tas de transistors à électron unique. Essentiellement, nous

manipulons un seul électron ou des paires d'électrons pour représenter tous les états ou combinaisons qu'utiliserait un ordinateur quantique.

Alors, quels sont les avantages des qubits de spin ?

JC : Du point de vue des performances, les spin-qubits ont des performances similaires à celles des autres types de qubits. Les spin-qubits sont fabriqués à partir de transistors. Et chez Intel, nous savons fabriquer de bons transistors. Nous savons comment fabriquer des puces contenant plusieurs dizaines de milliards de transistors. En fait, d'ici la fin de la décennie, notre PDG a prédit que nous disposerions de puces pouvant contenir un trilliard de transistors. Cela montre que nous savons comment mettre à l'échelle les transistors. Il existe une analogie selon laquelle si nous savons comment mettre à l'échelle les transistors et que nos spin-qubits sont comme des transistors, alors nous devrions avoir une voie plus claire pour mettre à l'échelle des systèmes quantique avec des millions de Qubits. Et c'est l'approche que nous adoptons chez Intel.

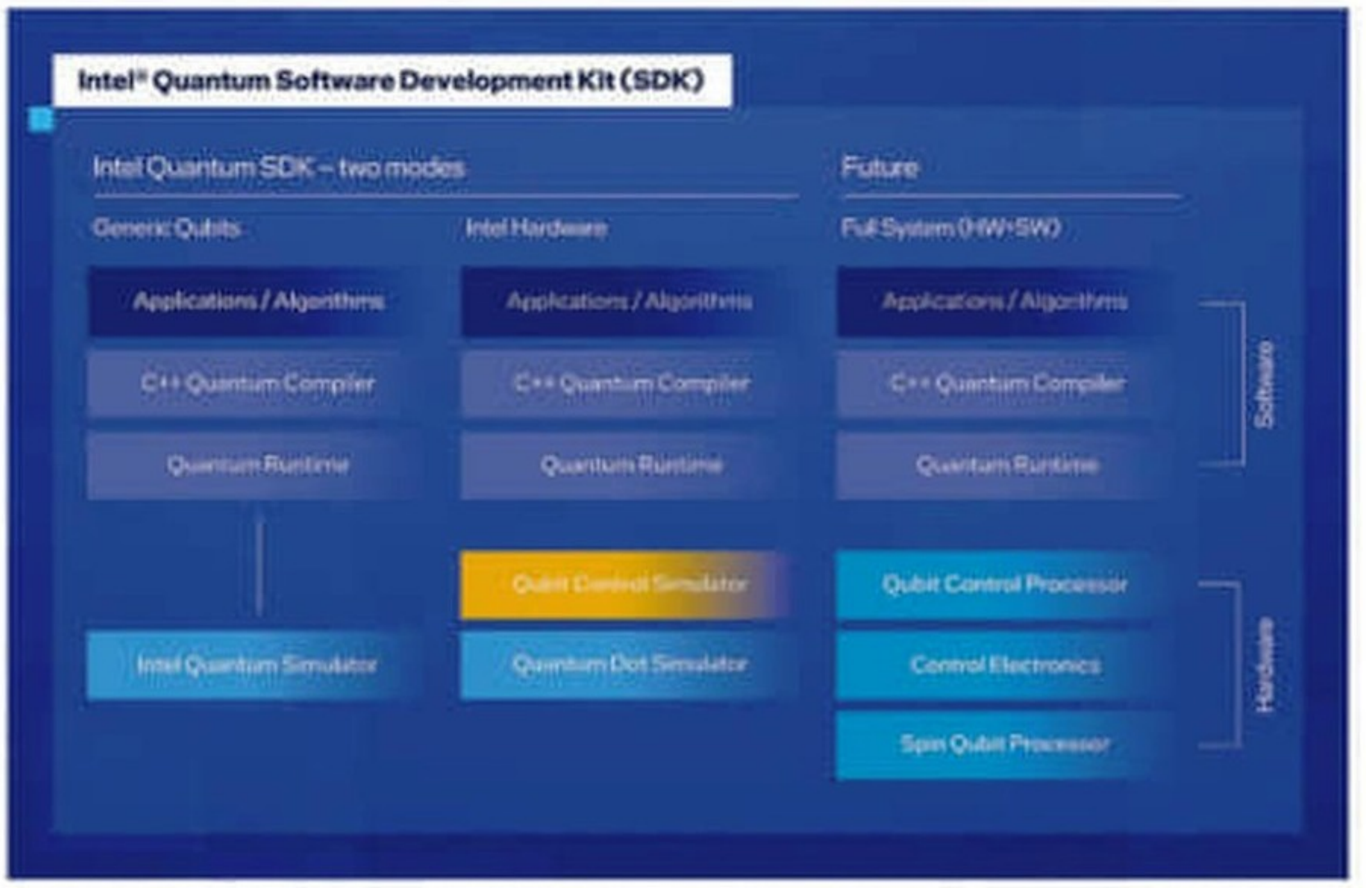
Pouvez-vous parler un peu de Horse Ridge ? De quoi s'agit-il et pourquoi est-ce si important dans le cadre de votre stratégie d'informatique quantique ?

JC : Lorsque vous disposez d'une puce à qubits, vous devez contrôler ces électrons uniques. La question est : comment les contrôler ? Chez Intel, non seulement nous souhaitons développer notre propre électronique de contrôle, mais nous souhaitons que cette électronique de contrôle soit très proche de la puce qubit à l'intérieur de notre réfrigérateur. Et lorsque nous faisons cela, nous pouvons essentiellement travailler à mieux faire évoluer la puce. Nous pouvons contrôler le nombre de câbles utilisés. Nous pouvons contrôler le facteur de forme de la puce. Nous avons une puce de contrôle surnommée « Horse Ridge » qui fonctionne très bien à quatre Kelvin, donc à seulement quatre degrés au-dessus du zéro absolu et qui se trouve à proximité immédiate de notre puce qubit. Et avec cela, nous avons prouvé à travers plusieurs générations de Horse Ridge que nous pouvons contrôler efficacement nos qubits.

Intel a travaillé avec QuTech autour de l'informatique quantique. Qu'est-ce que cela représente en terme de travail et de partenariat ?

JC : Nous avons démarré cette initiative fin 2015. Nous recherchions un partenaire académique. QuTech qui est une collaboration entre l'Université de technologie de Delft et l'Organisation néerlandaise pour la recherche scientifique appliquée (TNO), possède une expertise dans plusieurs types de qubits et dans l'électronique de contrôle. Nous entretenons un merveilleux partenariat qui dure maintenant depuis huit ans. Au début, QuTech a formé Intel à l'informatique quantique, puis à un stade ultérieur, Intel envoyait nos meilleurs dispositifs à QuTech pour caractérisation. Et nous avons récemment convenu d'étendre la collaboration QuTech tout en nous concentrant sur l'électronique cryogénique. J'ajouterais que ce que nous constatons aujourd'hui dans le monde entier, c'est que les gouvernements investissent de plus en plus dans l'informatique quantique et que de nombreux pays et leurs professeurs universitaires se tournent vers Intel pour s'associer. Cela représente une opportunité car nous pouvons fournir notre dernière puce, appelée Tunnel Falls, aux professeurs pour leur utilisation. Cela apporte deux avantages. D'une part, ils peuvent étudier des aspects de notre puce pour lesquels nous n'avons tout simplement pas la bande passante, afin d'accélérer notre apprentissage. Et d'un autre côté, cela permet aux étudiants de se familiariser avec nos appareils Intel et nous espérons qu'ils voudront un jour venir travailler pour Intel en tant qu'experts en quantique.

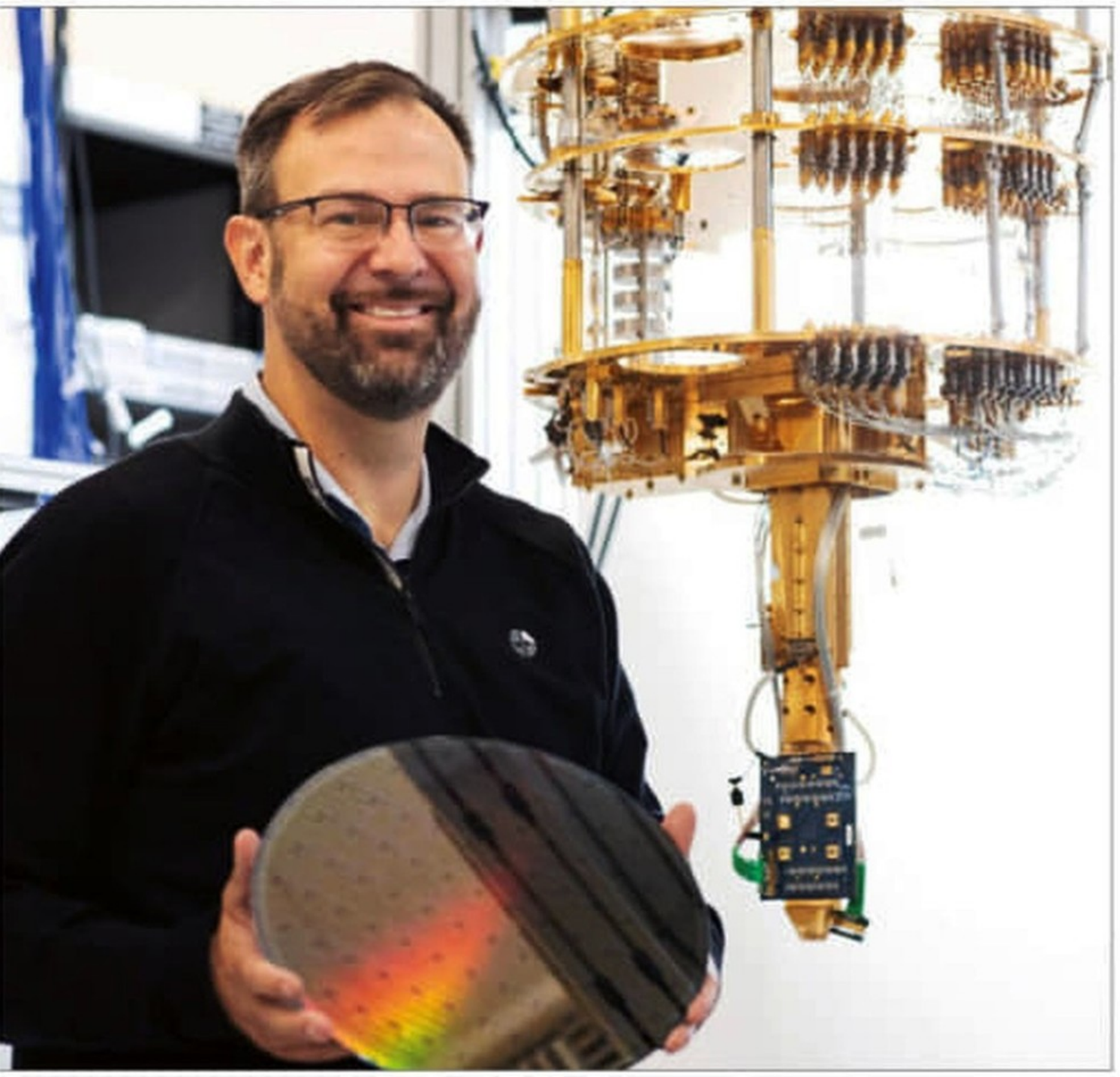
La recherche quantique d'Intel couvre l'ensemble de la pile de calcul, depuis les dispositifs qubit jusqu'à l'architecture matérielle, logicielle et les applications. Le SDK Intel Quantum est un ordinateur quantique complet en simulation qui peut également s'interfacer avec le matériel quantique d'Intel, notamment la puce de contrôle Horse Ridge II d'Intel et la puce spin-qubit d'Intel lorsqu'elle sera disponible plus tard cette année.



Le SDK Intel Quantum offre aux développeurs le choix entre deux backends cibles pour simuler les qubits : 1) un simulateur de qubit générique open source hautes performances, le Intel Quantum Simulator (IQS) ; et 2) Un backend cible qui simule le matériel Intel Quantum Qubit Dot et permet une simulation de modèle compact des spin-qubits de silicium d'Intel.

Pouvez-vous décrire ce qu'est votre Tunnel Falls ?

JC : Tunnel Falls est notre deuxième génération de puce quantique. C'est la première génération que nous diffusons dans la communauté universitaire ou dans les laboratoires nationaux.

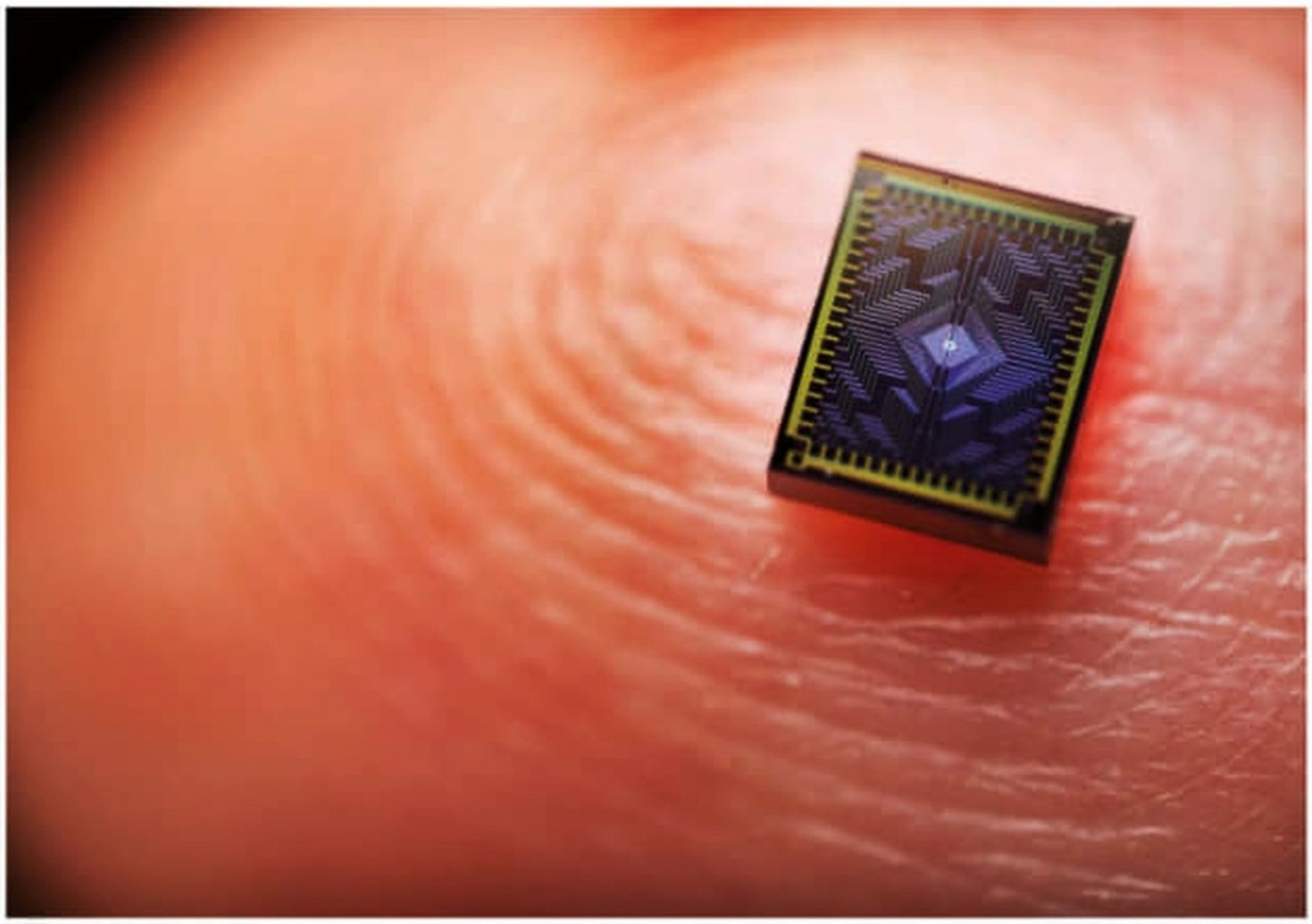


Intel s'engage à faire progresser le domaine de l'informatique quantique, notamment en créant une communauté de développeurs. Les utilisateurs bêta du SDK Intel Quantum explorent divers cas d'utilisation potentiels, notamment la dynamique des fluides, l'astrophysique et la conception de matériaux.

Il s'agit d'un appareil de 12 spin-qubits. Il est fabriqué dans l'usine D1 à Hillsboro, dans l'Oregon. Il utilise la lithographie EUV pour plusieurs couches. Puisque nous utilisons notre usine de transistors pour construire Tunnel Falls, cela s'appuie sur ce que nous faisons chaque jour. Notre rendement a été impressionnant pour cette puce. Lorsque nous faisons des choses avec cette puce du point de vue des performances de conception, c'est quelque chose que nous n'avons pas vu jusqu'à présent dans la communauté universitaire.

Sommes-nous encore au stade de la recherche fondamentale ou pouvons-nous maintenant passer de la théorie au matériel et aux logiciels réels ?

JC : C'est quelque part entre les deux. Il reste encore des défis fondamentaux à relever concernant les performances d'un qubit individuel. Dans le même temps, nous devons commencer à faire évoluer ces systèmes. Parce que lorsque nous faisons évoluer un système, nous allons rencontrer d'autres problèmes, tels que : comment faire le câblage, comment corriger les erreurs ou comment mapper les algorithmes sur un système à grande échelle. Il faut 10 à 15 ans pour mettre sur le marché une véritable nouvelle technologie. Ainsi, d'une part, nous essayons d'améliorer nos qubits et, d'autre part, nous essayons de les faire évoluer et de résoudre ainsi des problèmes d'ingénierie. Il est maintenant temps pour les développeurs de se demander comment programmer un ordinateur quantique ; comment créer des algorithmes très efficaces sur un ordinateur quantique ? Donc, même si cela prendra encore plusieurs années, nous devons maintenant nous concentrer sur le matériel, les logiciels et l'utilisateur final.



La puce quantique Tunnel Falls sur un doigt. Les spin-qubits de silicium sont jusqu'à 1 million de fois plus petits que les autres types de qubits.

INTERVIEW D'ANNE MATSUURA

Intel met une nouvelle puce quantique et un SDK complet à la disposition des universités et de la communauté pour accélérer la recherche en informatique quantique

Après avoir lancé sa version bêta en septembre 2022, Intel a publié la version 1.0 du kit de développement logiciel (SDK) Intel Quantum. Le SDK est un ordinateur quantique complet en simulation qui peut également s'interfacer avec le matériel quantique d'Intel, notamment la puce de contrôle Horse Ridge II d'Intel et la puce qubit de spin quantique d'Intel. Le kit permet aux développeurs de programmer des algorithmes quantiques en simulation et comprend une interface de programmation intuitive écrite en C++ à l'aide d'une chaîne d'outils de compilateur de machine virtuelle de bas niveau (LLVM) standard de l'industrie.

Pouvez-vous vous présenter ?

AM : Je m'appelle Anne Matsuura, je suis senior principle engineer et je suis la directrice des applications quantiques et de l'architecture système dans la division Intel Labs depuis 9 ans. Je suis à l'origine une physicienne et je travaille dans le domaine de la physique quantique depuis plusieurs décennies maintenant.

On ne sait toujours pas clairement à quoi servira un ordinateur quantique ou une programmation quantique. Quel est votre point de vue à ce sujet ?

AM : Ma formation technique initiale porte sur l'étude des supraconducteurs à haute température, dont l'objectif est de développer un jour des matériaux supraconducteurs à température ambiante. Je pense que l'informatique quantique sera utile pour simuler ce type de matériaux et d'autres systèmes quantiques. D'autres systèmes quantiques qui incluent des éléments tels que la compréhension des réactions chimiques et la création de nouveaux catalyseurs chimiques pour des réactions chimiques plus efficaces et plus propres. Simuler des matériaux avant de les créer en laboratoire rationalise la recherche de matériaux possédant des propriétés magnétiques ou électroniques inhabituelles, ce qui est bien sûr très intéressant pour les entreprises d'électronique.

Vous vous concentrez sur la partie logicielle de l'informatique quantique. Pourquoi est-ce essentiel au monde de l'informatique quantique ?

AM : Comme pour tout système informatique, vous avez besoin d'un logiciel d'application qui inclut un compilateur qui prend ce que le programmeur a écrit et l'optimise en quelque chose de plus petit, plus rapide et plus efficace à exécuter. Chez Intel, nous avons développé une chaîne d'outils logiciels qui comprend le compilateur Intel Quantum et un runtime qui fonctionne avec le système de contrôle matériel et la puce qubit. Et tout cela ensemble constitue le système informatique quantique. Sans cette partie logicielle, il n'existe pas de système quantique.

En quoi les logiciels développés pour les ordinateurs quantiques diffèrent-ils de ceux développés pour les ordinateurs classiques ?

AM : L'informatique quantique est un type de calcul totalement différent, car il est programmé d'une manière différente. Quand on parle de compilateur quantique, cela signifie compiler et optimiser un algorithme très différent d'un algorithme pour ordinateur classique. Mon équipe construit la pile logicielle, et c'est un exercice difficile car le matériel en est encore à ses balbutiements. Il existe des programmes quantiques purement quantiques tandis que d'autres sont partiellement quantiques et partiellement classiques. Notre chaîne d'outils logiciels est bien adaptée pour exécuter l'un ou l'autre de ces types de programmes.

> ***« L'Intel Quantum SDK est une pile informatique quantique complète en simulation qui offre un environnement de développement personnalisable pour un large éventail de développeurs. »***

Quelles sont les complexités du développement de logiciels pour le quantique ?

AM : Les qubits sont sujets au bruit quantique. Ils sont très sensibles aux conditions environnementales. Les systèmes aujourd'hui très bruyants et lorsque nous exécutons des algorithmes, aucune correction d'erreur n'est actuellement mise en œuvre sur les ordinateurs quantiques. Un grand nombre des programmes que nous gérons doivent donc fonctionner pendant la durée de vie d'un qubit qui peut être des fractions

Anne Matsuura,
Director of Quantum & Molecular Technologies, Intel Labs at Intel Corporation

Le Dr Anne Matsuura est directrice des applications et de l'architecture quantique chez Intel Labs. Auparavant, elle a occupé les postes de scientifique en chef de la Société d'Optique (OSA) et de directrice générale de l'Installation européenne de spectroscopie théorique (ETSF). Elle a été un investisseur stratégique dans des start-ups technologiques chez In-Q-Tel. Le Dr Matsuura était chercheur à l'Université de Lund en Suède, à l'Université de Stanford, à l'Université de Tokyo et professeur adjoint au département de physique de l'Université de Boston. Anne a obtenu son doctorat en physique de l'Université de Stanford.

intel.

de secondes. Cela rend donc la tâche très, très difficile à la fois pour les développeurs de logiciels et d'algorithmes. Mais les développeurs doivent commencer dès maintenant à développer des algorithmes quantiques et des domaines d'application quantiques afin qu'à l'avenir – lorsque nous disposerons d'ordinateurs quantiques à grande échelle tolérants aux pannes – nous puissions être opérationnels.

Quels outils avez-vous développés pour les développeurs ?

AM : Chez Intel, nous avons construit une pile logicielle et une chaîne d'outils complète conçue pour être le sommet d'un ordinateur quantique évolutif et commercial. Nous avons un compilateur très performant inclus dans notre SDK Intel Quantum qui effectue de nombreuses optimisations d'algorithmes. Il décompose votre algorithme et votre programme en quelque chose qui peut être exécuté sur la puce quantique d'Intel. Nous disposons également d'un simulateur de qubits performant accessible aux développeurs sur lequel nous pouvons simuler jusqu'à environ 30 qubits sur un ordinateur portable et plus de 40 qubits sur plusieurs nœuds. La simulation de qubits est très coûteuse en termes de calcul, car chaque fois que vous ajoutez un autre qubit, vous devez doubler la taille de la mémoire. Nous essayons de créer un écosystème de développeurs quantiques et d'utilisateurs d'ordinateurs quantiques habitués à travailler avec la technologie quantique d'Intel. En utilisant le SDK Intel Quantum, vous vivrez la même expérience que lorsque vous utiliserez notre logiciel sur notre véritable puce qubit.

> ***« Intel s'engage à faire progresser le domaine de l'informatique quantique et s'efforce de créer une communauté de développeurs. »***

Pouvez-vous nous en dire plus sur le SDK Quantum et nous donner un aperçu ?

AM : Le SDK Intel Quantum se compose du compilateur Intel Quantum, qui optimise les algorithmes quantiques et décompose les algorithmes en opérations - opérations physiques disponibles sur notre puce qubit. Il est basé en C++. Nous avons fait ce choix intentionnellement à des fins de performances. Notre compilateur est basé sur un LLVM standard de l'industrie, typique des compilateurs classiques que nous avons étendu avec des extensions quantiques. Nous avons fait cela intentionnellement afin de pouvoir open source le frontal du compilateur. Et ce faisant, les utilisateurs peuvent contribuer avec leurs propres passes de compilateur.

Je suis développeur de logiciels, comment puis-je accéder au SDK ?

AM : Il est disponible aujourd'hui, soit directement dans notre Intel Developer Cloud, soit sous forme d'image conteneurisée. Avec chaque distribution, vous écrivez votre code en C++ et le compilez depuis la ligne de commande. Le SDK Intel Quantum est livré avec un certain nombre d'exemples que vous pouvez exécuter directement. Vous pouvez commencer en interrompant ou en étendant ces exemples ou en implémentant votre propre programme. Pour savoir comment accéder au SDK, visitez notre site web Intel Developer Zone à l'adresse https://developer.intel.com/quantumsdk

Quels sont les cas d'utilisation déjà existants pour l'informatique quantique et les logiciels quantiques sur lesquels les développeurs peuvent travailler ?

AM : Nous en avons déjà parlé plus tôt, comme la chimie, la simulation des matériaux ou la dynamique des fluides. De nombreuses entreprises se tournent aujourd'hui vers l'informatique quantique et développent de petits algorithmes quantiques pour leur secteur d'activité. Il y a du travail dans divers domaines tels que la finance et les produits pharmaceutiques. De nombreuses entreprises commencent à réfléchir à l'utilité future de l'informatique quantique pour leur stratégie d'entreprise. Mais il s'agit encore d'un domaine naissant, et les travaux sur les algorithmes quantiques et les applications quantiques sont encore exploratoires.

Il y a quelques années, Intel travaillait sur le projet de simulateur quantique Intel ; est-ce que celui-ci est toujours d'actualité ?

AM : Oui, ça l'est. En fait, le simulateur Intel Quantum, qui est un simulateur de qubits hautes performances, est intégré au SDK Quantum comme l'un des back-ends ; c'est-à-dire l'un des simulateurs de qubits utilisés après le travail de notre compilateur et de notre environnement d'exécution. Il fait désormais partie du SDK même s'il est toujours disponible seul en tant que projet open source. Si vous souhaitez que l'algorithme quantique soit réellement optimisé, construisez-le avec le compilateur et exécutez-le avec le moteur d'exécution sur le simulateur Intel Quantum.

Pourquoi les développeurs devraient-ils désormais se soucier de développer du code pour l'informatique quantique ?

AM : L'informatique quantique va être un nouveau changement de paradigme pour l'informatique. Elle ne remplacera pas l'informatique classique, mais elle sera utilisée en parallèle avec l'informatique classique. Je pense que cela nous ouvrira la porte à la possibilité de résoudre certains de ces problèmes complexes que les ordinateurs classiques ne peuvent pas résoudre. Je pense que tant pour les développeurs que pour les entreprises, il est temps de commencer à réfléchir au type de problèmes qu'ils souhaiteraient qu'un ordinateur quantique résolve. Et il est temps pour nous de commencer à rechercher des algorithmes quantiques et des applications quantiques, car nous ne pouvons pas attendre d'avoir un ordinateur quantique à grande échelle pour ensuite essayer de savoir quoi en faire.

Démonstration d'un algorithme quantique variationnel

Résumé en Français : le SDK Intel Quantum comprend des fonctionnalités facilitant l'écriture et l'exécution d'algorithmes exprimés dans une combinaison de logique de programmation traditionnelle et quantique et d'algorithmes hybrides quantiques-classiques. Cet exemple est une petite démonstration du Variational Quantum Eigensolver. Cette classe d'algorithmes est fréquemment utilisée dans les calculs de chimie et de physique quantiques qui modélisent des systèmes en interaction de particules quantiques. Cependant, elle est applicable à des problèmes plus généraux pouvant être cartographiés sur un système de particules. L'exemple utilise deux qubits pour modéliser deux particules en interaction. L'algorithme fonctionne en associant les atouts de l'informatique quantique à ceux de l'informatique classique. L'action de l'algorithme fonctionne comme suit : la logique quantique, représentée par un circuit quantique, prépare une estimation (un ansatz) de l'état ayant l'énergie la plus basse pour le système à deux particules et effectue des mesures sur cet état. La logique classique utilise les résultats des mesures pour calculer l'énergie totale pour cet état, puis utilise des techniques d'optimisation pour former une estimation améliorée. L'algorithme se répète ensuite jusqu'à ce que les résultats de la routine d'optimisation convergent en préparant l'ansatz amélioré, en calculant l'énergie totale de cet état et en calculant une nouvelle estimation.
Code source : sur programmez.com et GitHub

Anne Matsuura
Director of Quantum & Molecular Technologies, Intel Labs at Intel Corporation

Of the applications to possibly allow quantum advantage using near-term systems, variational quantum-classical algorithms are considered to be among the most promising [1]. The Intel Quantum SDK includes features to facilitate writing and running algorithms expressed in a combination of traditional and quantum programming logic, hybrid quantum-classical algorithms.

This example is a small, but non-trivial demonstration of the Variational Quantum Eigensolver [2]. This class of algorithms is frequently used in quantum chemistry and physics calculations that model interacting systems of quantum particles. However, it is applicable to more general problems that can be mapped onto a system of particles [3].

The example uses two qubits to model two interacting particles. The algorithm works by pairing the strengths of quantum computing with those of classical computing. The action of the algorithm works as follows: The quantum logic, represented by a quantum circuit, prepares a guess (an *ansatz*) of the state with the lowest energy for the two-particle system and performs measurements on that state. The classical logic uses results of the measurements to compute the total energy for that state, and then uses optimization techniques to form an improved guess. The algorithm then repeats until the results of the optimization routine converge by preparing the improved ansatz, computing the total energy of that state, and calculating a new guess.

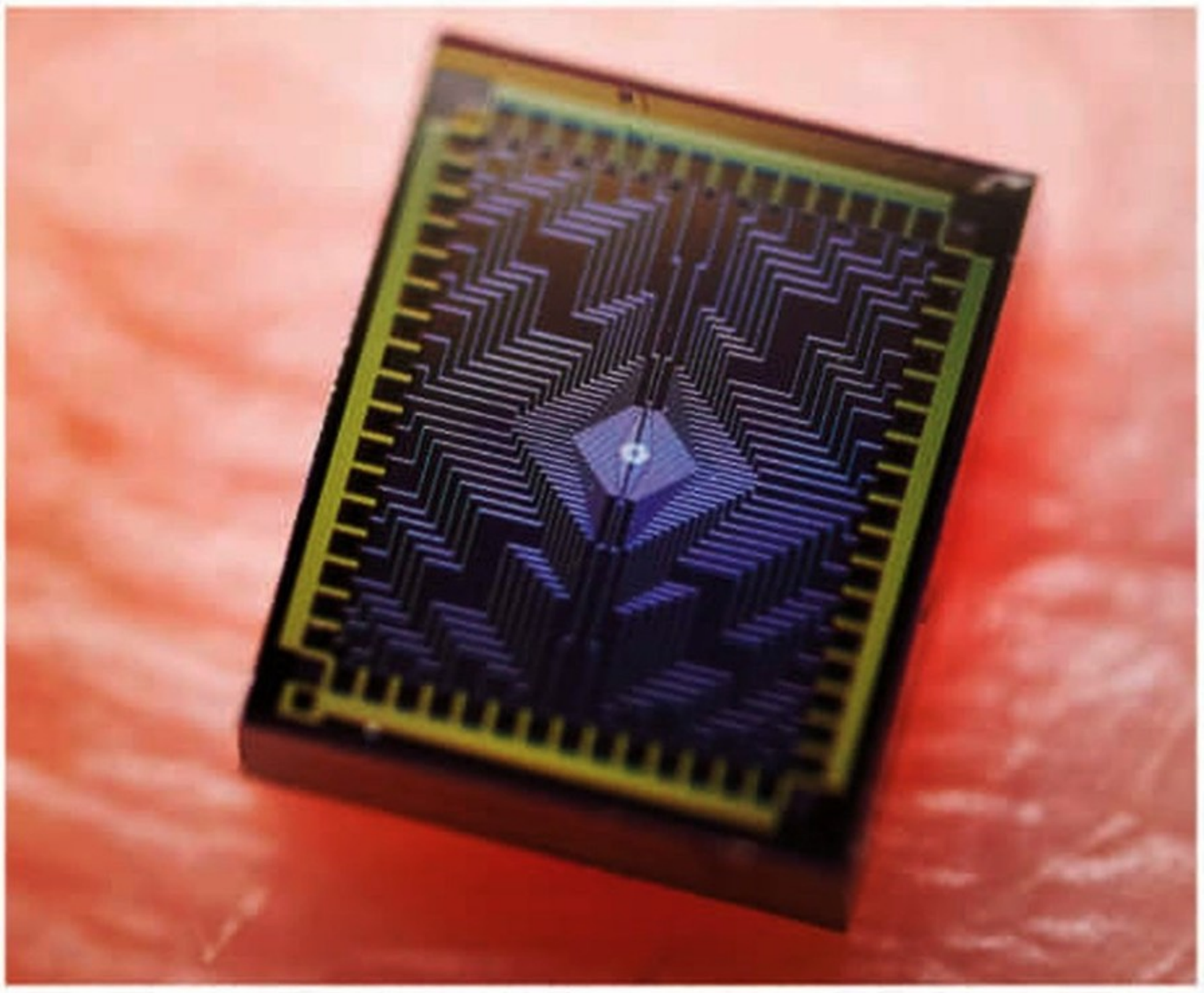
The source code begins with a comment that gives the energy equation (referred to by physicists as the system's Hamiltonian) for this test system in terms of the quantum mechanical operators. This is the cost function that will be optimized.

Immediately following that comment are the header files for several of the classes in the Intel Quantum SDK as well as the header for dlib, an open-source optimization library. These header inclusions are followed immediately by the declaration of a variables used to define the problem: a qubit register of two qubits and a set of four parameters to describe the state of the two-qubit system. dlib defines its own data structures to represent the optimization parameters, and so a typedef is prepared to more simply express variables of that type. Lastly, the steps_count will record how many iterations through the algorithm are required before it converges.

The functions labeled with the quantum_kernel keywords contain the instructions defining the quantum algorithm. The first two, prep_z_all and ansatz_linear, represent the two steps of our quantum logic, initializing the qubits and preparing them in the specified state. The fourth quantum_kernel calls these first two sequentially (this is not strictly necessary, here it illustrates that a quantum_kernel can be composed of other quantum_kernels). The third quantum_kernel applies a Hadamard gate to all the qubits; this will change the measurement basis from Z-axis to X-axis so that we can collect the results for the energy equation terms that include X operators.

The next set of functions represent those individual terms of the energy equation (albeit all in terms of Z operators because the hadamard_all function will be used with it for the terms involving X). The matrices in the comments are the tensor product results of the two operators, and the matrix acting on the qubit-register's state is encoded in the returned quantity of each function.

Now that all the quantum logic has been defined, the main function details the classical logic of our hybrid algorithm. The first act is to create an instance of the FullStateSimulator class that will represent the quantum backend. The ready function signals the quantum runtime that this object will run the next quantum_kernel called.



To compile this example, invoke the compiler and use the '-I' (that's a capital "eye") flag to indicate to the compiler where you installed dlib.

```
intel-quantum-compiler -I./dlib simple_vqe.cpp
```

Because the source code set the FullStateSimulator to run in verbose mode, running the executable produces a sequence of all the quantum instructions run during each iteration. The final output shows the values that parameterize the final state as well as the computed total energy:

```
Optimized Parameters using BOBYQA:
1.3557292800237959
-2.1747206308722609
-0.25940742245165149
0.14624749801162643
BOBYQA execution count: 95
Total energy using BOBYQA: -0.90138781764713471
```

The next block of code prepares references to the qbit variables. These references will be used as input to the methods of the FullStateSimulator to gather simulation data about the relevant qubit states. In this implementation, the simulation data will be used in place of measurement observations. One reality of quantum hardware is that at the end of a quantum circuit, the state of the qubit isn't a knowable quantity. Instead, one measures each qubit and observes either a 0 or 1 as the outcome. Many such measurements would be needed to reconstruct the probability distributions that we can extract from the simulation of qubits. In this way, the simulation of qubits isn't just a stand-in for quantum hardware; it's a complementary tool in the quantum developer's toolkit.

The next block of code prepares a lambda expression and its inputs. In C++, a lambda expression is a function that returns a function. This lambda expression sets the new values for the input parameters describing the state to be prepared, runs the quantum circuit, and then computes the total energy from qubit simulation's probability distribution.

This total energy is returned by the lambda expression as a function, and in the final block of code, that function is given as an input to an optimization method from dlib to minimize the total energy. This method works because of the Variational Principle, a theorem of physics that shows that calculating the energy of the Hamiltonian using the wrong ground state will only over-estimate the total energy, never under-estimate. So in a sense, each time a state is created that produces a lower energy, one knows that state is closer to the actual ground state energy than previous states. For a sophisticated version of this algorithm, the selection of the state preparation affects the quality of the result, and the choice of optimization technique can lead to faster or slower convergence.

As written, this example represents only the first step of the path to running an application on a quantum computer. There are several ways it can be extended as a hypothetical next step. As mentioned earlier, this implementation used data from the simulation of the qubits in a way that isn't reproducible on quantum hardware. For this program to run on quantum hardware, it would need to compute those probability distributions from sequences of measurements. This change can be written and validated with help from a method provided by the FullStateSimulator class, getSamples. Another factor to consider is how many iterations are needed to converge on the solution. A different choice of optimization routine might use fewer iterations, representing an overall faster solution, or might be more robust in the presence of noisy qubits.

Lastly, to write out the terms of the cost function, a developer would need to have some prior knowledge about working with quantum operators. The Intel Hybrid Quantum-Classical Library does some of this work for the developer by calculating the individual terms required. Furthermore, the library can group measurements and set the necessary basis mappings to collect multiple allowed measurements simultaneously. For an example of how this library works in conjunction with a variational quantum-classical algorithm, see the post written by a member of our team at https://community.intel.com/t5/Intel-Quantum-SDK/How-would-I-use-the-Intel-Quantum-SDK-to-write-a-complete/td-p/1473142

[1] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, Nature Reviews Physics 3, 625 (2021).

[2] Peruzzo, A., McClean, J., Shadbolt, P. et al. A variational eigenvalue solver on a photonic quantum processor. Nat Commun 5, 4213 (2014). https://doi.org/10.1038/ncomms5213

[3] Lucas A (2014) Ising formulations of many NP problems. Front. Physics 2:5. doi: 10.3389/fphy.2014.00005

Intelligence Artificielle et Calcul Quantique : quels rapports ?

Jean-Michel TORRES
IBM Quantum Ambassador, Qiskit advocate

IA et Calcul Quantique : deux histoires parallèles qui pourraient se rejoindre.

Il y a seulement quelques mois, l'Intelligence Artificielle (IA) a soudainement défrayé la chronique avec les « IA génératives » d'images (Dall-E, Stable-Diffusion) ou de textes (tels que GPT-3, BERT...). Les résultats sont spectaculaires, facilement accessibles et utilisables par le grand public. Enthousiasme d'un côté, mises en garde éthiques et sociétales, certains rappellent qu'il ne s'agit en fait que de mathématiques et de statistiques. Le débat redémarre sur la définition de l'intelligence, la conscience et le computationnalisme(1) entrent à nouveau en scène... Toujours est-il que la technologie de l'apprentissage automatique a franchi une étape et en promet d'autres.

On pourrait par exemple imaginer que l'IA pourra elle-même développer les algorithmes quantiques pour l'IA. Et ce serait la fin de cet article !

L'IA est une idée ancienne, mais naît au milieu du 20e siècle du point de vue de la science et de la technologie (avec entre autres les travaux d'Alan Turing et la notion de neurone artificiel : le perceptron de Rosenblatt), à peine cinquante ans après l'observation et la description des neurones humains par Santiago Ramòn y Cajal en 1906.

Et il a fallu encore un demi-siècle avant que les machines acquièrent la capacité suffisante pour réaliser l'entraînement de réseaux de neurones à une taille intéressante et utile. La reconnaissance visuelle, les systèmes de recommandation, les moteurs de traduction automatique développent l'industrie numérique, les véhicules autonomes apparaissent.

De manière un peu similaire, le développement de la mécanique quantique s'est surtout déroulé au début du 20e siècle pour proposer une théorie au début des années 1930.

Celle-ci précède de 50 ans environ l'idée du calcul quantique. C'est en effet au début des années 1980 que Richard P. Feynman fait le constat que l'informatique ne peut et ne pourra jamais effectuer les calculs nécessaires à la chimie et à la science des matériaux. Il propose alors l'idée selon laquelle l'observation d'objets régit par les lois de la mécanique quantique peut nous révéler des informations sur d'autres systèmes quantiques. Cette intuition est à l'origine du calcul quantique et invite la communauté scientifique à maîtriser l'infiniment petit, du point de vue expérimental, c'est le début de ce que l'on appelle la seconde révolution quantique :

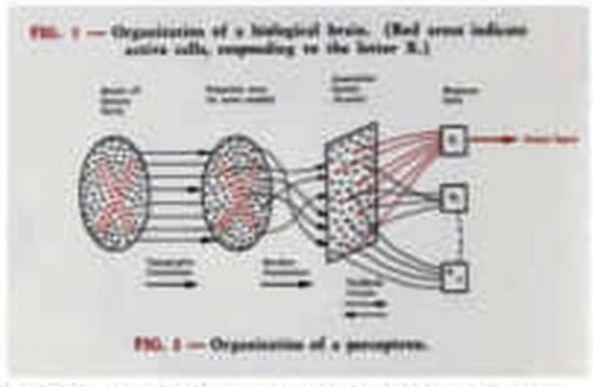
« I'm not happy with all the analyses that go with just the classical theory, because nature isn't classical, damnit, and if you want to make a simulation of nature, you'd better make is quantum mechanical » **Figure 1**

C'est une nouvelle théorie de l'information qui émerge, les vingt années suivantes accumulent les progrès algorithmiques et technologiques, il faut toutefois attendre plus de dix ans pour que les premiers résultats apparaissent.

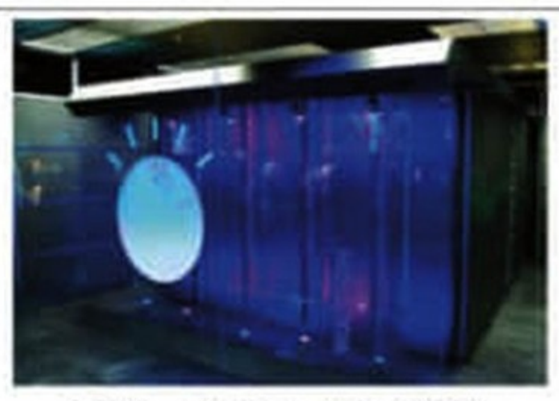
David Deutsch montre en 1992 comment les principes du cal-

(1) Le computationnalisme est une théorie fonctionnaliste en philosophie de l'esprit qui, pour des raisons méthodologiques, conçoit l'esprit comme un système de traitement de l'information et compare la pensée à un calcul (en anglais, computation) et, plus précisément, à l'application d'un système de règles. (Source Wikipedia)

1906 – Santiago Ramon y Cajal : prix Nobel, l'observation du neurone



1958 –le perceptron de Rosenblatt



2011 – Watson, l'IA d'IBM

1927 – Congrès de Solvay
L'avènement de la mécanique quantique

1981 – Conférence MIT + IBM : The Physics of Computation

2023 - La feuille de route d'IBM pour un calcul quantique utile

Crédits photo : Wikipedia Cornell University (Perceptron), CERN(R.P.Feynman, IBM(Quantum decade)

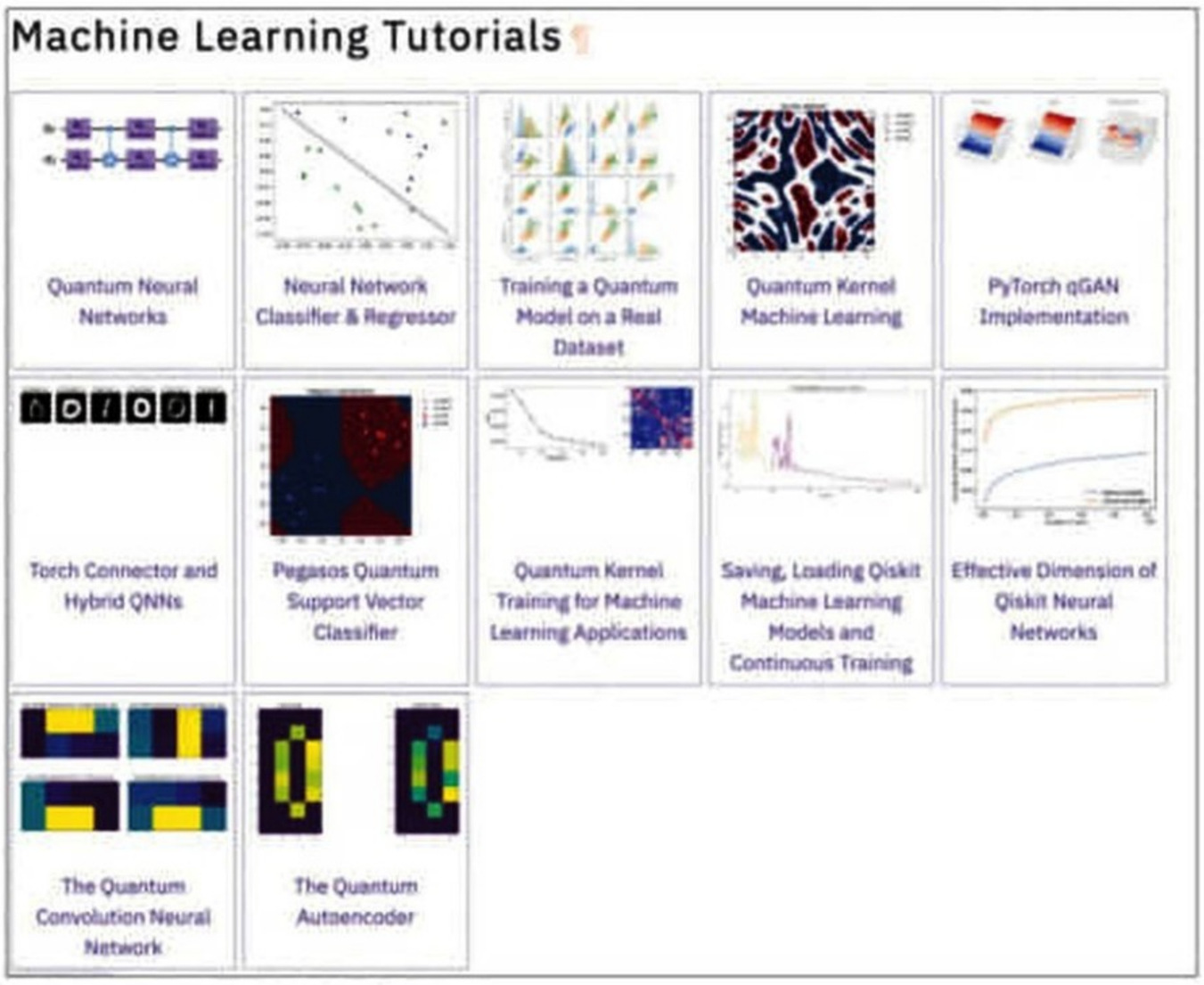


Figure 2
Quantum AI : quatre quadrants

cul quantique permettent de décider en une seule fois une certaine propriété d'une fonction, alors qu'il faut deux itérations pour parvenir au résultat avec les principes de l'informatique classique. C'est une première preuve d'avantage, mais le problème résolu n'a pas d'application pratique. En 1994 l'algorithme de Shor, puis l'algorithme de Grover en 1996 apportent des résultats promettant une certaine « utilité », le développement des algorithmes quantiques a démarré.
De nombreuses pistes ont été explorées et ce n'est plus seulement la physique et la chimie, mais aussi d'autres types de calculs qui sont en jeu : arithmétique, optimisation et cette généralisation ouvre évidemment la question de l'IA quantique. J'y reviendrai bien sûr.
Dans le même temps, le développement technologique, la maîtrise de plus en plus fine des phénomènes physiques à l'échelle réellement microscopique permet de construire et d'utiliser les premiers ordinateurs quantiques. On parle de « qubit » (quantum bit) comme unité de manipulation de l'information.
À l'heure actuelle, plusieurs possibilités sont explorées pour fabriquer ces qubits : IBM, Google, D-Wave, Rigetti travaillent sur des puces de silicium à des températures extrêmement basses, en France la startup QUANDELA utilise des photons tandis que la startup PASQAL manipule directement des atomes et il existe bien d'autres voies, y compris peut-être celles que nous ne connaissons pas encore.
Le domaine se développe de plus en plus rapidement, mais, ayant démarré environ 30 ans après celui de l'intelligence artificielle, il est loin d'avoir acquis une maturité comparable. En fait, nous disposons aujourd'hui de machines NISQ(2), en clair des ordinateurs de « petite » taille et « bruités », c'est-à-dire sujet à des erreurs. On espère des résultats réellement utiles à relativement court terme avec de tels ordinateurs par exemple dans le domaine des sciences des matériaux. Mais pour des usages plus généraux et en particulier pour les calculs pratiqués en Intelligence Artificielle, c'est l'ère du FTQC(3) qui apportera des accélérations substantielles. Ces machines, incluant des qubits de bonne qualité que l'on peut associer à des systèmes de correction d'erreur, sont attendues en particulier par IBM sous un horizon de 10 ans, elles ouvriront alors la voie à une nouvelle classe de calcul.
Les ordinateurs quantiques pourront alors pleinement exprimer leur potentiel : la possibilité d'agir en un même temps sur un très grand nombre de valeurs : un processeur possédant N qubits peut traiter 2 à la puissance N valeurs simultanées. Aujourd'hui le plus grand processeur d'IBM contient 433 qubits : 2 à la puissance 433 est un nombre tout à fait gigantesque (dépassant de très loin 2 à la puissance 275 qui représente approximativement le nombre de particules dans l'univers observable). C'est en cela que l'ordinateur quantique est particulièrement adapté aux calculs nécessitant une complexité d'ordre exponentiel.
Concernant l'IA, dans sa conception actuelle, l'utilisation du calcul quantique est entravée par ces deux limitations :
L'une est fondamentale : on ne peut pas cloner un état quantique, on peut le déplacer (d'un qubit à l'autre), mais pas le copier (plus précisément on ne peut pas en général en faire une copie exacte),
L'autre est d'ordre technologique : il est très difficile actuellement (mais pas impossible) de fabriquer des mémoires pour des états quantiques (qRAM(4)). En clair, autant Big-Data et AI vont ensemble, autant le calcul quantique n'est pas à l'aise dans le domaine des mégadonnées.
La qRAM fait bien entendu l'objet d'une recherche active, mais la conséquence immédiate est qu'il n'est pas facile de projeter les structures classiques de l'AI dans le calcul quantique. Cependant rien n'est perdu, il faut explorer d'autres manières de traiter le problème et accepter l'injonction de Jay Gambetta, Vice-Président en charge de la Recherche Quantum chez IBM : « *You're Thinking too classically* ».

Les quatre quadrants

Voici donc différentes manières de discuter le sujet de l'AI et du calcul quantique, représentées dans ce tableau : **Figure 2**
Le premier quart « **CC** » correspond au traitement en IA de données classiques avec des algorithmes classiques. Pourquoi parler de cette catégorie, qui n'a pas sa place ici a priori ? En fait la recherche dans le domaine des algorithmes quantiques n'a rien à voir avec le « *refactoring* »(5) d'un calcul, et encore moins d'une application, il s'agit de repartir sur les principes mêmes du calcul et de le reconcevoir selon les propriétés fondamentales du calcul quantique (superposition et intrication d'états quantiques par exemple). Ce faisant, il peut arriver que l'on remarque qu'une nouvelle idée dans un algorithme quantique puisse aussi être utilisée de manière classique. On découvre ainsi un algorithme classique plus

(2) *NISQ : Noisy Intermediate Scale Quantum.*

(3) *FTQC : Fault Tolerant Quantum Computing.*

(4) *qRAM: Quantum random-access memory.*

(5) *Le « refactoring » ou « réusinage » de code est l'opération consistant à retravailler le code source d'un programme informatique – sans toutefois y ajouter des fonctionnalités ni en corriger les bogues – de façon à en améliorer la lisibilité et, par voie de conséquence, la maintenance, ou à le rendre plus générique (afin par exemple de faciliter le passage de simple en multiple précision) ; on parle aussi de « remaniement ». Cette technique utilise quelques méthodes propres à l'optimisation de code, avec des objectifs différents. (source Wikipedia)*

performant que l'état de l'art, il s'agit alors de ce que l'on pourrait appeler un « bénéfice collatéral » de la recherche en calcul quantique. Certains parlent de « *déquantisation* », ou encore d'algorithmes « *quantum inspired* ».

Et ceci est arrivé précisément dans le champ du Machine Learning quantique. Voici l'histoire : en 2008 Aram Harrow, Avinatan Hassidim et Seth Lloyd démontrent que l'ordinateur quantique peut être exponentiellement plus efficace pour réaliser l'inversion d'une matrice. Il s'agit de l'algorithme HHL[6]. Il se trouve que ceci est particulièrement utile dans le domaine de l'entraînement des réseaux de neurones et ce résultat a enthousiasmé la communauté de la recherche dans ce domaine. De nombreux résultats (théoriques) ont suivi, comme qGAN(7) et autres qNLP(8)... et par exemple qREC(9) en 2016, c'est-à-dire une version quantique des algorithmes de recommandation, au cœur de très nombreux sites de ventes ou de streaming. qREC utilise un cas restreint de HHL. Il reste à prouver « l'optimalité » (c'est-à-dire la version quantique est-elle la meilleure?), et c'est dans cette étape justement, qu'en 2018, qREC est « déquantisé » par une doctorante dans un laboratoire Canadien.

Il y a aussi le quadrant « **CQ** » : ces algorithmes classiques au service du développement du calcul quantique, je ne m'étendrai pas trop sur le sujet, il s'agit de techniques classiques en vue d'analyser les données pour améliorer la qualité les processeurs, ou de rendre les compilateurs plus efficaces. Bien sûr, l'IA peut apporter beaucoup à la recherche pour le développement du calcul quantique, naturellement partout où il y a des données en grande quantité (par exemple dans l'analyse des paramètres fins de pilotage des qubits afin d'en améliorer la précision des algorithmes). Et également dans les recherches d'optimisations des circuits quantiques dans leur phase de transpilation vers le processeur quantique (il s'agit de trouver comment réarranger les qubits et les opérations pour minimiser le nombre d'étapes), pour ne citer que quelques exemples.

Le quadrant « **QC** » : les programmes quantiques pour les problèmes et les données classiques constituent la majorité des recherches actuellement. Ici, ce que l'on attend du processeur quantique, c'est sa capacité à effectuer des calculs dans de **grands** espaces, sur de **grands** nombres de possibilités simultanément. Tout est dans le « grands », car ici on parle de tailles définitivement inconcevables pour des ordinateurs classiques. Les résultats sont nombreux autour de la modélisation des données, l'idée est d'encoder différemment et très efficacement des données dans des états de qubits. Un certain nombre de preuves ont été apportées pour des cas retreints, d'autres restent à démontrer. Certaines pistes seront peut-être abandonnées, mais personne n'a jamais affirmé que l'ordinateur quantique pourrait surpasser le classique dans l'ensemble des domaines d'application, au contraire, on sait que seulement certains problèmes seront traités plus efficacement sur les ordinateurs quantiques, mais l'accélération attendue est telle que le jeu en vaut la chandelle.

La généralisation semble difficile, mais on n'a pas non plus prouvé qu'elle sera impossible, en particulier lorsque l'on disposera d'ordinateurs quantiques assez vastes et dotés de correction d'erreur.

Pour terminer le tableau, le quadrant « **QQ** » évoque de nouveaux horizons : le traitement de problèmes de nature quantique, avec des données directement quantiques, par des algorithmes quantiques, c'est un champ encore peu exploré, mais qui pourrait, selon l'intuition de Richard P. Feynman, apporter des avancées spectaculaires, lesquelles? Justement cela reste à découvrir.

En attendant, pour les plus curieux d'entre vous, je recommande une visite du site de qiskit.org, dans lequel on trouve de nombreux exemples de programmes quantiques en quelques lignes de code, avec des tutoriels, des jeux de données restreints et des résultats qui illustrent les « grands classiques » : SVM(10), qGAN... **Figure 3**

Conclusion

Le domaine de l'IA comme celui du calcul quantique est en ébullition. Chacun sera en mesure de profiter des développements de l'autre. Il est très difficile de prédire l'avenir des technologies au-delà de quelques années, mais les conditions sont en place pour des innovations aussi passionnantes que prometteuses.

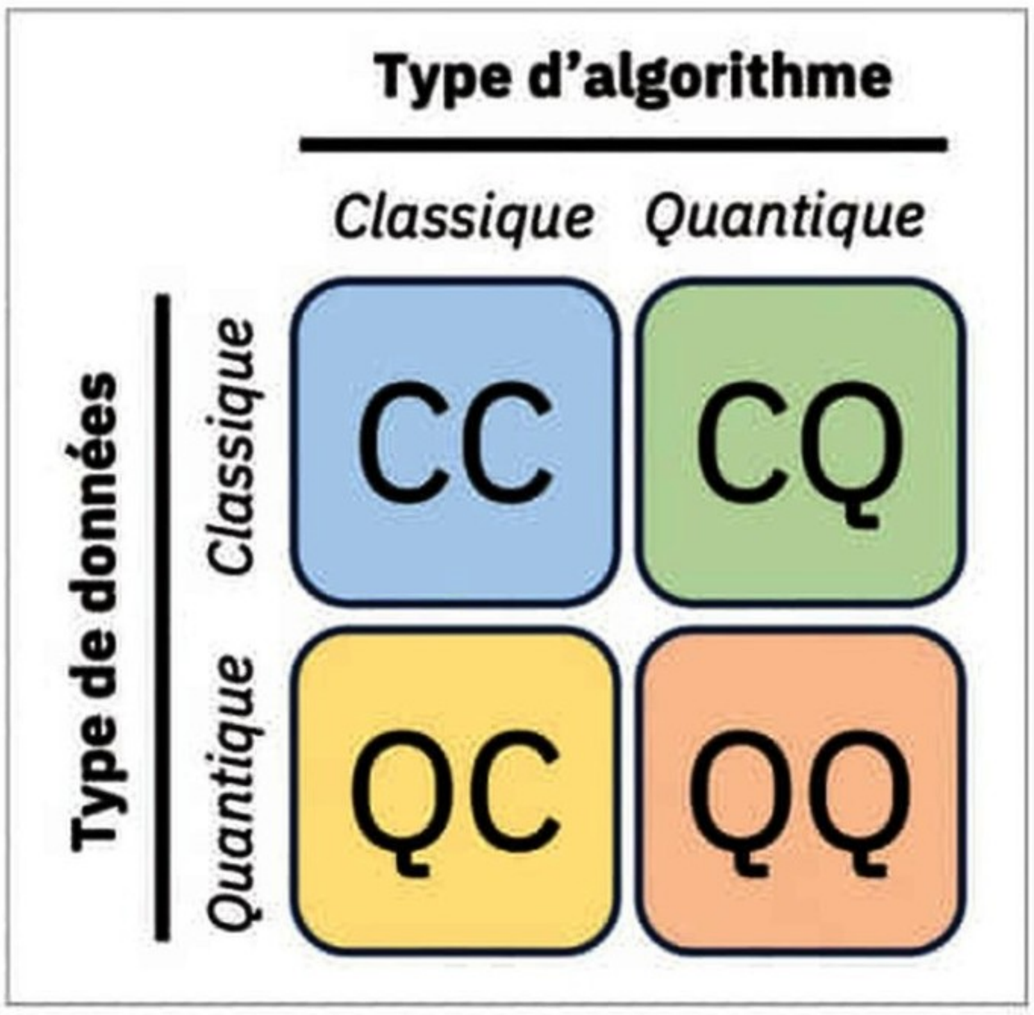


Figure 3
qiskit.org / Documentation / Machine Learning

(6) HHL : Harrow, Hassidim & Lloyd.

(7) qGAN : Quantum Generative Adversarial Network.

(8) qNLP : Quantum natural language processing.

(9) qREC : quantum RECommandation. Dans le domaine de l'IA : REC est une référence aux algorithmes de RECommandation (le cas d'école est Netflix dont le but est de présenter à l'utilisateur le prochain film qu'il appréciera probablement le plus, par analyse des contenus qu'ont appréciés un groupe d'utilisateurs possédant un historique de visionnage commun), il y a aussi du REC dans Youtube et Amazon.... Et donc qREC fait référence à la version quantique d'un algorithme de recommandation.

(10) SVM : Support Vector Machine.

Comment faire de la mitigation d'erreurs quantique ?

Aziz Ngoueya Akanji Benga

Ayant obtenu un diplôme généraliste en informatique, je me suis spécialisé assez vite dans le domaine de l'informatique quantique en rejoignant l'équipe support du réseau IBM Quantum. Notre objectif est d'aider et résoudre les problèmes des clients qui ont accès aux machines fabriquées par IBM. Qiskit en fait partie.

Les ordinateurs quantiques sont très prometteurs et ont le potentiel de révolutionner plusieurs domaines, car ils peuvent résoudre des calculs encore inaccessibles à la puissance de calcul classique (ordinateurs classiques) en un temps décisif. Il existe déjà des algorithmes quantiques qui démontrent cet avantage : c'est le cas de l'algorithme de Grover, Shor, etc.

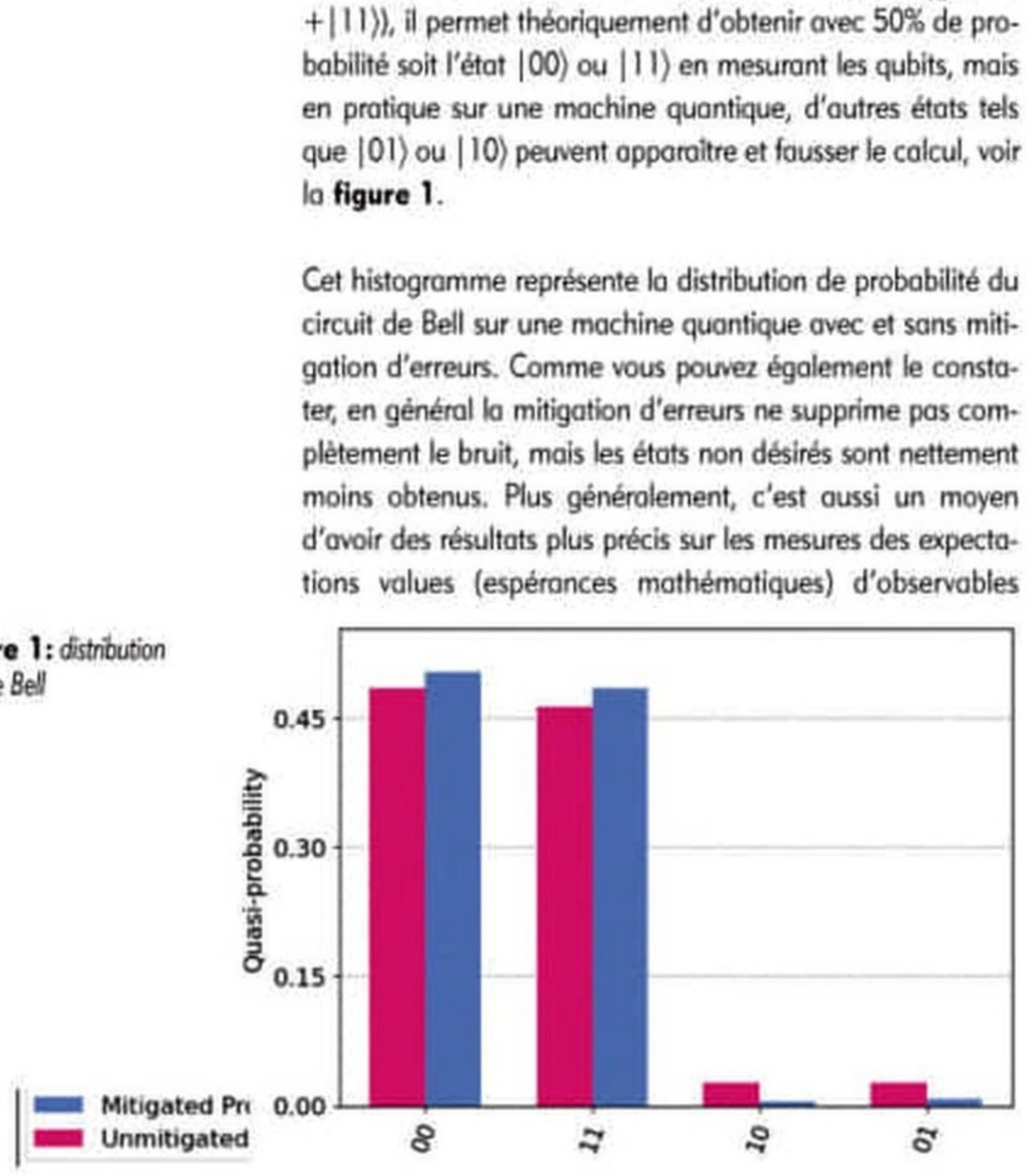
Seulement, les ordinateurs quantiques actuels sont naturellement sensibles au bruit de par leurs interactions avec l'environnement, mais aussi de par les erreurs des portes quantiques et de mesures. Il existe plusieurs techniques pour gérer le bruit quantique et l'atténuer et le corriger. Elles peuvent être rassemblées dans trois catégories : l'error suppression, l'error mitigation et l'error correction. Dans cet article, l'error mitigation sera uniquement abordé.

Error mitigation

L'error mitigation ou mitigation d'erreurs est un ensemble de techniques permettant de réduire les erreurs affectant les résultats des programmes quantiques. C'est un peu similaire aux casques antibruit qui fonctionnent en caractérisant le bruit dans l'environnement, puis utilisent ces données du bruit pour adapter et injecter du bruit supplémentaire. Ajouter plus de bruit au système permet d'annuler précisément le bruit d'origine et d'obtenir une absence de bruit. Prenons le très célèbre état de Bell sur deux qubits $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, il permet théoriquement d'obtenir avec 50% de probabilité soit l'état $|00\rangle$ ou $|11\rangle$ en mesurant les qubits, mais en pratique sur une machine quantique, d'autres états tels que $|01\rangle$ ou $|10\rangle$ peuvent apparaître et fausser le calcul, voir la **figure 1**.

Cet histogramme représente la distribution de probabilité du circuit de Bell sur une machine quantique avec et sans mitigation d'erreurs. Comme vous pouvez également le constater, en général la mitigation d'erreurs ne supprime pas complètement le bruit, mais les états non désirés sont nettement moins obtenus. Plus généralement, c'est aussi un moyen d'avoir des résultats plus précis sur les mesures des expectations values (espérances mathématiques) d'observables (valeurs physiques mesurables). Pour arriver à diminuer les erreurs, le bruit de la machine est caractérisé grâce à la sortie des circuits spécifiques. Il existe plusieurs méthodes, en voici quelques-unes.

Figure 1: *distribution état de Bell*



Readout error mitigation

Le readout error mitigation (REM), en français mitigation d'erreurs de lecture, a pour but de réduire les erreurs introduites lors de la mesure des qubits. Une façon de modéliser les erreurs de mesure est de considérer d'abord qu'une mesure parfaite donnant l'état $|\psi\rangle$ est réalisée sans aucun type de bruit, puis les résultats obtenus passent par un canal bruyant qui bruite certains bits créant des erreurs aléatoires et résultant en un état bruité $|\psi'\rangle$

La première étape de la mitigation d'erreurs de lecture est de mesurer les circuits dans les différents états de base et voir les probabilités de chaque résultat. Ceci génère une matrice M appelée matrice de confusion ou d'assignement. Si $P_{idéal}$ et $P_{bruité}$ sont considérés comme étant les vecteurs de probabilité respectifs des résultats du circuit bruité et du circuit sans bruit, le modèle de bruit en considérant que les erreurs de lecture s'écrit mathématiquement de la forme $\mathbf{P_{bruité} = AP_{idéal}}$

La matrice d'assignement **M** caractérise les taux d'erreurs de mesure du hardware et qui est générée à l'aide de circuits de calibration. Elle est indépendante du circuit et plus précisément elle permet d'évaluer la fidélité des portes de mesures en calculant la probabilité d'obtenir l'état $|x\rangle$ sachant que l'état attendu est $|v\rangle$ pour toute paire $|u\rangle$, $|v\rangle$ d'états de base. Avec deux qubits la matrice d'assignement équivaut à la **figure 2** :

$$\begin{bmatrix} Pr(00|00) & Pr(00|01) & Pr(00|10) & Pr(00|11) \\ Pr(01|00) & Pr(01|01) & Pr(01|10) & Pr(01|11) \\ Pr(10|00) & Pr(10|01) & Pr(10|10) & Pr(10|11) \\ Pr(11|00) & Pr(11|01) & Pr(11|10) & Pr(11|11) \end{bmatrix}$$

Figure 2: *Matrice d'assignement sur deux qubits*

Pr(00|11) corresponds à la probabilité d'avoir le bitstring 00 alors que le résultat devrait être 11. Avec un ordinateur quantique parfait, la matrice d'assignement aurait uniquement des 1 sur la diagonale principale et des 0 ailleurs. Les probabilités en dehors de la diagonale principale caractérisent le bruit causé par les opérations de mesures, les autres types d'erreurs sont négligés.

Une approche directe pour estimer cette matrice est de créer des circuits pour chaque état de base et calculer leur distribu-

tion de probabilité. Donc avec n qubits, 2^n circuits de mitigation d'erreurs sont nécessaires. Le nombre de circuits augmente très rapidement et la méthode est très vite coûteuse. Les circuits de calibration créés sont assez simples, on applique la porte quantique X si le qubit a un 1 sinon on ne fait rien.

Une autre version moins coûteuse est de supposer que les qubits sont indépendants ce qui permet d'avoir plusieurs matrices d'assignement de taille (2,2) pour chaque qubit et ensuite permet de calculer leur produit tensoriel afin d'obtenir la matrice d'assignement finale. Il suffit donc de générer deux circuits, l'un avec des 0 sur tous les qubits et l'autre avec des 1 sur tous les qubits.

Une fois la matrice d'assignement construite, la dernière étape consiste à inverser la matrice M et à l'appliquer au résultat bruité du circuit bruité, cela produira une estimation mitigée des résultats $P_{mitigée}$ On applique la matrice M au vecteur colonne résultant de la distribution de probabilités de la mesure du circuit.

Le module **qiskit-experiments** offre la possibilité d'implémenter ces deux méthodes de readout error mitigation avec respectivement les classes **CorrelatedReadoutError** et **LocalReadoutError**, en voici les principales étapes pour les utiliser.

```
# Choisir la liste de qubits à calibrer pour la matrice d'assignement
qubits = [0,1,2,3,4]
# Instancier la classe LocalReadoutError ou CorrelatedReadoutError
exp = LocalReadoutError(qubits)
#exp = CorrelatedReadoutError(qubits)
#Executer les circuits de mitigation afin de calculer la matrice d'assignement.
result = exp.run(backend)
mitigator = result.analysis_results(0).value
# Construire et exécuter les circuits à mitiger
circuit = QuantumCircuit(num_qubits)
counts = backend.run(circuit, shots=shots, seed_simulator=42).result()
.get_counts()
# Appliquer l'inverse de la matrice d'assignement
mitigated_quasi_probs = mitigator.quasi_probabilities(counts)
```

Mthree

La technique dite « matrix free measurement mitigation », ou M3, est également une méthode de mitigation d'erreurs de lecture adaptée pour des programmes plus gros en termes de qubits. Ici la matrice d'assignement est construite uniquement sur les bitstrings renvoyés et non pour tous les bitsrings à savoir 2^n états possibles comme c'est le cas pour les méthodes citées au préalable. Sur l'exemple de la figure 3 la matrice d'assignement A est réduite à en considérant que les bitstrings en violet qui correspondent à la distribution de probabilité du circuit à mitiger. Au final, on a une matrice de taille (9,9) au lieu de (16,16). **Figure 3**

Par ailleurs M3 exploite les méthodes itératives matrix free tel que la méthode de Jacobi pour avoir le résultat mitigé. Elles permettent de résoudre des systèmes d'équations plus efficacement. Les packages mthree et qiskit-ibm-runtime permettent d'exploiter cette technique. Il faut spécifier le resilience_level = 1 et utiliser le sampler de qiskit-ibm-runtime pour appliquer mthree. Notez que le sampler calcule la probabilité d'obtenir chaque état de base. Les circuits sont exécutés plusieurs fois afin d'obtenir la distribution de probabilité.

```
# Création des circuits et s'assurer de mesurer les qubits
circuit = QuantumCircuit(num_qubits)
# Choisir le backend à utiliser (machine physique ou simulateur)
backend = service.get_backend('device')
# Dans les options bien specifier le résilience level à 1.
options = Options()
options.resilience_level = 1
# Créer une instance du sampler et l'exécuter sur le backend défini
sampler = Sampler(backend, options= options)
job = sampler.run(circuit)
# stocker les quasi probabilités des bitstrings
quasi_dists = job.result().quasi_dists
```

Les résultats des expériences montrent que la mitigation d'erreurs de mesure est effective, mais son utilité est limitée. Pour réaliser une mitigation des autres types d'erreurs lors de la lecture, il nous faut explorer d'autres techniques.

Twirled readout error extinction

« Twirled readout error extinction » ou T-rex utilise une technique appelée « Pauli Twirling » afin de de réduire les erreurs introduites par les mesures des qubits et pour calculer les expectations values quantiques. Pauli twirling est une forme de compilation aléatoire qui insère des paires de portes de Pauli (I, X, Y, Z) avant et après des portes d'intrication de sorte que le circuit reste inchangé. T-rex fonctionne en appliquant des portes de Pauli aléatoires P_q avant la mesure sur le circuit à mitiger schéma (a) de la figure 4 puis ce même opérateur P_q est appliqué sur un autre circuit initialisé à l'état $|0\rangle$ (b). Au final, l'expectation value est estimée grâce au rapport des valeurs moyennes des circuits (a) et (b). **Figure 4**

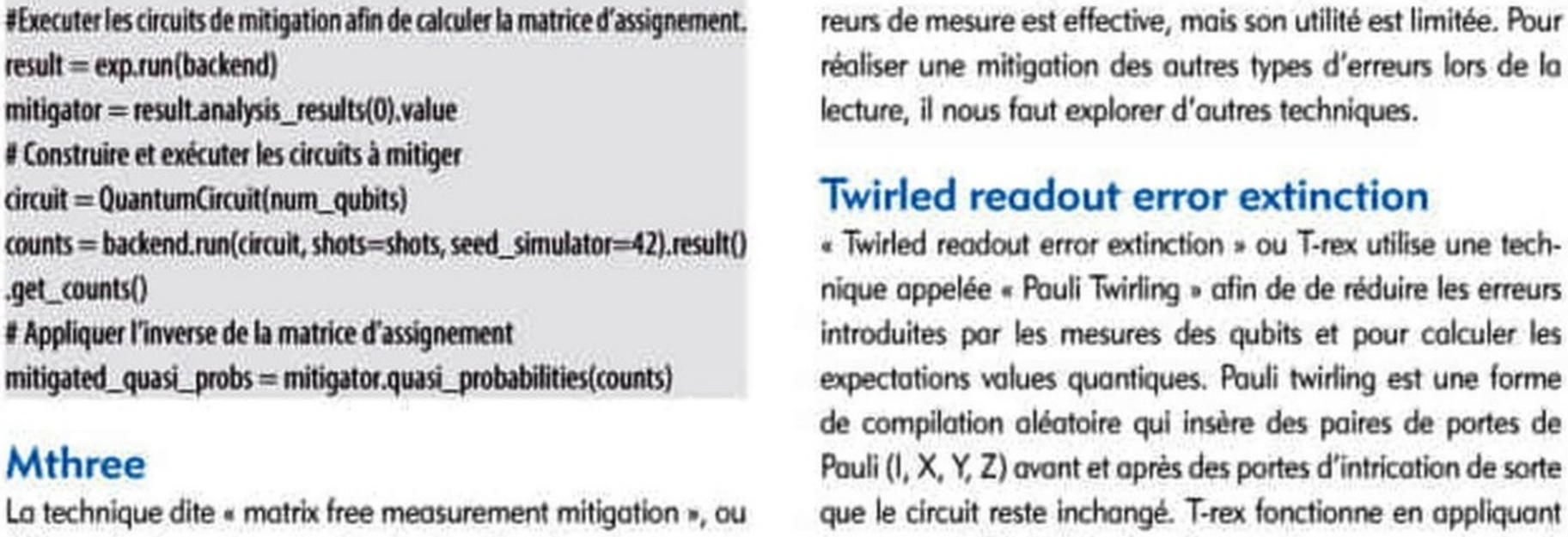


Figure 3 : *Reference - "Scalable Mitigation of Measurement Errors on Quantum Computers", Paul D. Nation, Hwajung Kang, Neereja Sundaresan, and Jay M. Gambetta, PRX Quantum 2, 040326 (2021)*

Avec « resilience level » à 1 et la classe « Estimator » de qiskit-ibm-runtime, il est possible d'effectuer l'algorithme T-rex. L'Estimator est une stratégie qui estime l'expectation value d'observables (valeur moyenne) qui est utilisée dans certains algorithmes. Ci-dessous les principales étapes pour implémenter l'algorithme en qiskit.

```
# Création des circuits
circuit = QuantumCircuit(num_qubits)
# Choisir le backend à utiliser (machine physique ou simulateur)
backend = service.get_backend('device')
# Dans les options bien specifier le résilience level à 1.
options = Options()
options.resilience_level = 1
# Créer une instance de l'estimator et l'exécuter sur le backend
estimator = Estimator(backend, options= options)
job = estimator.run(circuit)
# Obtenir les expectations values
exp_val = job.result().values
```

Zero noise extrapolation ou zne

Zero noise extrapolation est une technique d'error mitigation dans laquelle l'expectation value ou espérance mathématique est calculée pour plusieurs niveaux de bruit. À partir des résultats obtenus, l'espérance mathématique idéale est déduite par extrapolation.

Sur la figure 5, C1 et C2 sont les deux facteurs d'étirement utilisés pour amplifier le bruit. E1 et E2 sont les résultats des deux circuits avec des niveaux de bruit amplifiés respectivement par C1 et C2. E' est le résultat extrapolé. **Figure 5**

Le processus fonctionne en plusieurs étapes :

1. Augmenter intentionnellement le bruit du circuit en étendant le circuit pour plusieurs facteurs de noise C1, C2 par exemple.
2. Exécuter les circuits obtenus sur une machine quantique et calculer leurs expectation values E1, E2.
3. Calculer l'expectation value en extrapolant à la limite du zéro bruit grâce à une fonction définie au préalable afin d'avoir le résultat mitigé proche de la valeur exacte.

Deux méthodes sont généralement utilisées pour amplifier le bruit : ZNE digitale et ZNE analogue.

Dans la méthode ZNE digitale ou « gate folding », les portes quantiques sont répétées plusieurs fois ce qui aura pour effet d'augmenter la profondeur du circuit. Sur la figure 6, le circuit généré (droite) reste intrinséquement identique au circuit de base(gauche), car les portes insérées $U^{\dagger}U$ sont équivalentes à la porte identité mais plus le circuit a des portes quantiques, plus il est susceptible d'être bruité. **Figure 6:**

Avec la méthode ZNE analogue, la représentation pulse des portes est directement utilisée pour amplifier le bruit. Les circuits de gauche et droite(figure 7) sont équivalents par contre les durées des portes pulses U sont augmentées pour générer plus de bruit dans le circuit de droite. **Figure 7**

Enfin, l'extrapolation pour avoir le résultat mitigé est effectuée grâce à un fitting de fonction qui peut être de modèle linéaire, polynomiale ou encore exponentielle. En spécifiant resilience_level = 2, la librairie qiskit-ibm-runtime va automatiquement créer et exécuter les circuits zne. En dessous les étapes à suivre pour effectuer la méthode zne digitale.

```
# Choisir le backend à utiliser (machine physique ou simulateur)
backend = service.get_backend('device')
# Dans les options bien specifier le résilience level à 2
options = Options()
options.resilience_level = 2
# Créer une instance de l'estimator et l'exécuter sur le backend
estimator = Estimator(backend, options= options)
job = estimator.run(circuit)
# Obtenir les expectations values
exp_val = job.result().values
```

La méthode zne analogue nécessite de modifier la taille des pulses des portes quantiques ce qui peut être compliquée dans le cadre de cet article.

Notes : *Qiskit est une librairie python permettant de faire de l'informatique quantique. Afin de pouvoir exécuter les circuits sur des machines physiques, il faut impérativement créer un compte sur IBM Quantum Platform (https://quantum-computing.ibm.com/) et installer la librairie qiskit-ibm-runtime :*

```
pip install qiskit
pip install qiskit-ibm-runtime
```

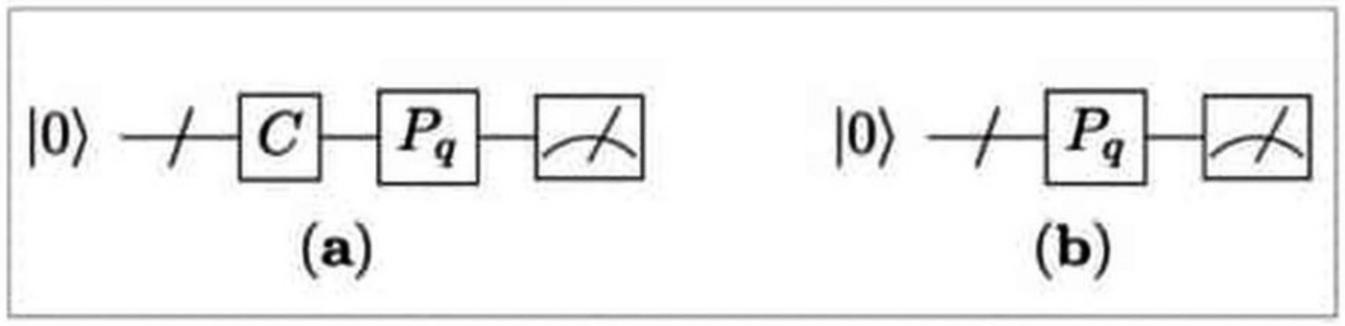


Figure 4 : *circuits T-rex, reference- E. van den Berg, Z. Minev, and K. Temme, Model-free readout-error mitigation for quantum expectation values arXiv:2012.09738*

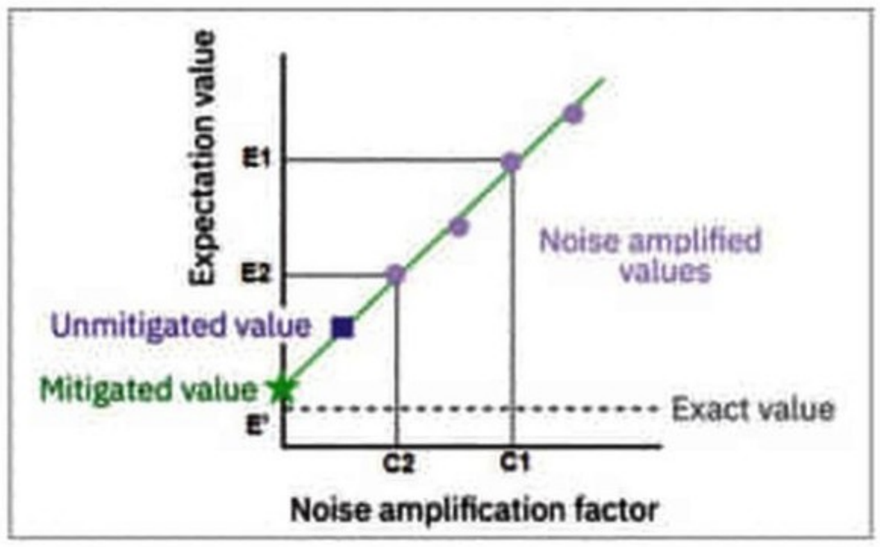


Figure 5 : *fonctionnement de ZNE*

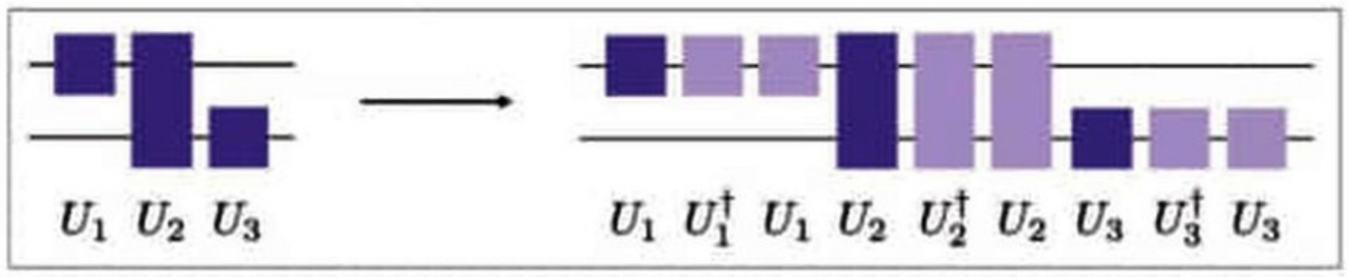


Figure 6 : *ZNE digitale*

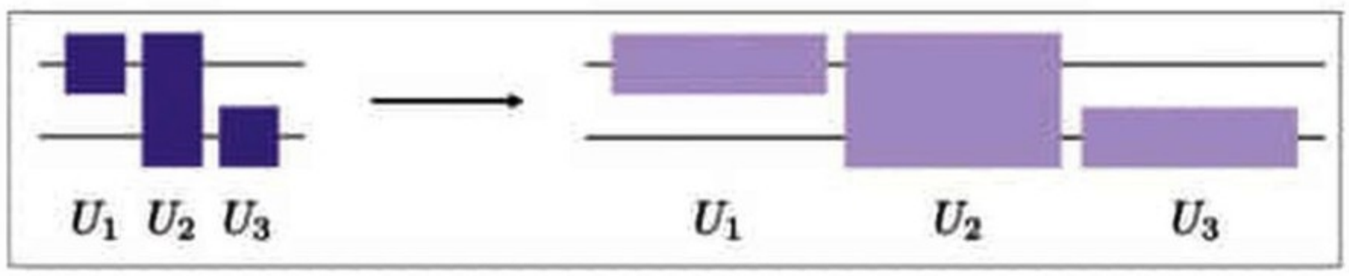


Figure 7 : *ZNE analogue*

Informatique quantique : ce que vous devez savoir pour vous lancer

Olivier Hess

Head Quantum Computing France & Global Quantum Computing Technology Advisor

Olivier a longtemps travaillé dans le domaine du HPC et du Quantum Computing chez IBM. En 2021 il a rejoint Eviden (an Atos Business) où il a en charge le développement des activités consulting d'informatique quantique, la promotion des solutions d'Eviden, la coordination des collaborations et partenariats en France. Doctorat – Chimie quantique, Université Paris VI, France, 1989

EVIDEN

Depuis la naissance de l'informatique, on fait appel à des machines de plus en plus puissantes pour résoudre des problèmes industriels toujours plus complexes. Cette course à la puissance, qui a longtemps suivi la loi de Moore, ralentit depuis quelques années car la miniaturisation des processeurs nous fait atteindre les limites de la physique classique.

Le développement de l'intelligence artificielle révolutionne certes la façon d'adresser des problèmes industriels, mais n'en diminue pas moins les besoins en puissance de calcul, au contraire. L'informatique quantique, elle, propose de tirer profit de phénomènes de la mécanique quantique (qui ont lieu à très petite échelle) pour accélérer la résolution de certains problèmes particulièrement complexes. L'objectif à terme étant la mise au point d'ordinateurs qui permettront de faire des calculs spécifiques beaucoup plus rapidement et/ou plus précisément que les ordinateurs classiques actuels.
Il s'agit d'un sujet difficile pour lequel nous vous proposons dans ce numéro une introduction pédagogique.

Comment fabriquer un processeur quantique ?

L'informatique quantique s'appuie sur les développements de la mécanique quantique dont les premiers développements datent des années 1900 environ. La mécanique quantique a introduit, en particulier, deux notions fondamentales connues sous le nom de « **superposition** » et « **intrication** » (notions explicitées dans l'encadré).
Le développement d'un processeur quantique passe par la maitrise d'une structure de base que l'on appelle un **qubit** ou bit quantique qui précisément pourra, dans des conditions particulières, exhiber ces propriétés de superposition et intrication. Le qubit est donc la structure qui va porter l'information quantique. Ce qubit peut être vu comme l'équivalent du bit classique de nos ordinateurs actuels.

L'ordinateur quantique et le qubit

En pratique, tout système physique ayant deux états quantiques distincts, contrôlables et évolutifs, est susceptible d'être un candidat pour créer un qubit. Il existe aujourd'hui différentes approches permettant de concevoir un qubit. On peut par exemple utiliser des atomes, des ions, des photons, des structures de silicium ou des matériaux supraconducteurs (dont les éléments de base sont, pour la technologie la plus répandue actuellement, appelés transmons). En pratique, le développement et la maîtrise des technologies de qubits sont extrêmement compliqués pour des raisons propres au monde quantique, et chacune de ces technologies possède avantages et inconvénients. Mais en tout état de cause, aujourd'hui il est encore impossible de concevoir un « qubit parfait ».
La deuxième difficulté dans le développement d'un ordinateur quantique est le passage à l'échelle. Avoir un qubit (même parfait) ne sert à rien ; il en faut un très grand nombre, des milliers, voire des millions. Et bien évidemment concevoir un tel système pose des difficultés scientifiques et industrielles très importantes, qu'il faudra des années pour surmonter.

La génération actuelle d'ordinateurs quantiques

La conséquence directe des deux difficultés précédentes est que l'on ne sait que concevoir des ordinateurs quantiques à « faible nombre de qubits imparfaits ». Des erreurs causées par divers phénomènes physiques sont inhérentes au fonctionnement de cette génération d'ordinateurs, erreurs qualifiées de « **bruit** » (ou noise en anglais). C'est ce que l'on appelle l'ère NISQ, pour « Noisy Intermediate Scale Quantum ».
Depuis environ 10 ans un très grand nombre d'acteurs de l'informatique et startups travaillent sur le développement de ces systèmes NISQ. De nombreux acteurs sont présents : en Amérique du Nord IBM, Intel, Google, Xanadu, DWave ; en Europe : IQM, AQT... En France, citons par exemple Pasqal, Quandela, Alice&Bob, Quobly ou C12. A nouveau, chacun de ces acteurs fait le choix d'une technologie de qubit particulière. Aujourd'hui tous les systèmes évoqués ci-dessus comportent au maximum quelques centaines de qubits ... loin encore des milliers ou des millions nécessaires !

L'ère des ordinateurs parfaits

Comme déjà évoqué, les systèmes actuels sont « imparfaits » (ère NISQ) et de forts développements seront encore nécessaires pour tendre vers l'ordinateur idéal, en particulier concernant la correction des erreurs et le passage à l'échelle. C'est ce que l'on appelle l'ère FTQC (« Fault Tolerant Quantum Computer ») ou parfois LSQ (« Large Scale Quantum »). Ce Graal n'est pas prévu avant 10 ou 15 ans probablement. C'est à ce moment-là seulement que les ordinateurs quantiques pourraient permettre des traitements extrêmement complexes inaccessibles aux calculateurs scientifiques classiques, même les plus puissants au monde.

Les familles d'ordinateur quantiques

Au-delà des différentes technologies de qubits évoquées ci-dessous, il existe deux grands modèles de programmation - digital et analogique - qui vont impacter fondamentalement la manière dont on va développer une application.
Le **digital** est aussi appelé ordinateur quantique universel à

portes. Il permet en théorie d'implémenter n'importe quel algorithme quantique. En pratique on va appliquer à chaque qubit, q0, q1, q2, ... des « portes » et ainsi créer un circuit quantique comme illustré ci-dessous. Un circuit quantique est donc un modèle de calcul quantique, similaire aux circuits logiques classiques, dans lequel un calcul est une séquence de portes quantiques, de mesures, d'initialisations de qubits à des valeurs connues et éventuellement d'autres opérations.
Figure 1

L'**analogique** se segmente en deux sous-catégories : recuit quantique et simulateur. Le recuit quantique ou Quantum Annealing (QA) permet de trouver l'état fondamental d'un modèle de physique, appelé le modèle d'Ising. Il résout donc spontanément un problème de minimisation (ici c'est l'énergie qui est minimisée). En formulant son problème comme un problème de minimisation, on peut donc utiliser le recuit quantique. Ainsi on peut par exemple résoudre des problèmes dits d'optimisation combinatoire comme nous allons l'illustrer plus loin. Les simulateurs quantiques, de leur côté, permettent de simuler des phénomènes quantiques, pour étudier par exemple des molécules complexes ou d'autres systèmes complexes (qualifiés de « à N corps »).

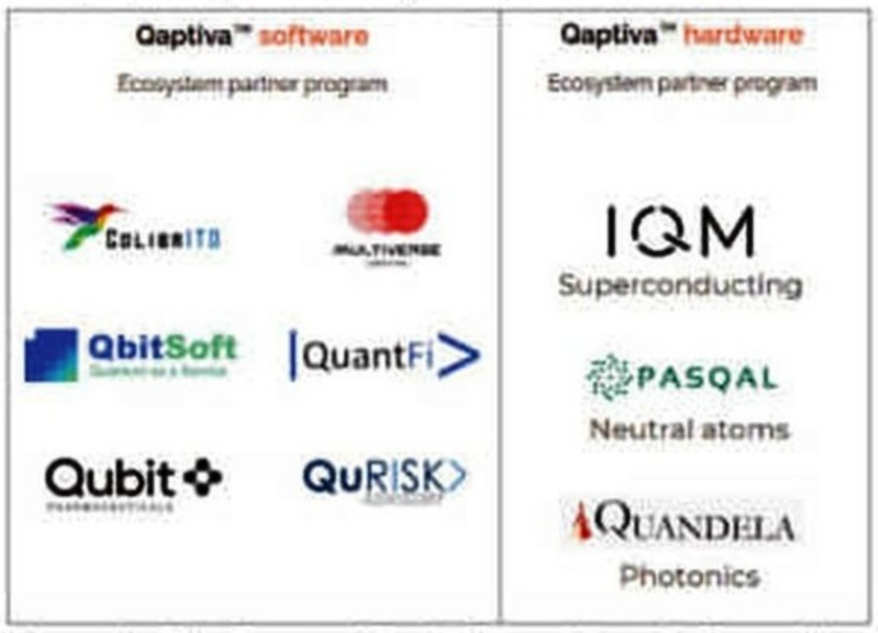
Le Quantique chez Eviden et la stratégie Qaptiva

Bull SAS, société française, est le premier constructeur européen de supercalculateurs. Elle participe à l'aventure du calcul quantique depuis 2016. Bull fait partie d'Atos, puis de la structure Eviden récemment annoncée.

La stratégie d'Eviden pour le calcul quantique est différente de celle des acteurs présentés précédemment. Compte tenu de la difficulté de développer des processeurs quantiques, Eviden a pris l'option, dès 2016, de développer un **émulateur quantique** ; l'idée étant que, en attendant la maturité du matériel, un point essentiel est le développement d'algorithmes et d'applications, ainsi que la couche logicielle nécessaire à l'utilisation effective des processeurs quantiques (cf. ci-après). Eviden produit des outils adaptés à l'apprentissage et l'exploration du calcul quantique connus sous le nom de Qaptiva et à destination des professionnels de l'industrie et de la recherche. Ces outils permettent le développement d'algorithmes quantiques, leur mise au point, et leur exécution sur l'émulateur Qaptiva ou sur des machines NISQ (et demain FTQC/LSQ).

La solution Qaptiva offre en particulier la capacité d'émuler la plupart des technologies de qubits et de les caractériser finement, et d'adapter un code quantique écrit génériquement aux spécificités d'un processeur particulier. En classique, vous pouvez par exemple développer un code en C++ sans vous soucier de savoir s'il tournera sur Windows ou Linux. De la même façon, avec Qaptiva, vous pouvez par exemple coder un circuit quantique à portes sans vous soucier de savoir sur quel processeur quantique à portes il tournera.

L'utilisateur n'est donc ainsi pas cantonné à une technologie particulière, ce qui lui permet de développer des solutions agnostiques au matériel quantique. Un axe fort de la stratégie Eviden est également la mise en place de partenariats stratégiques avec les acteurs du monde quantique. D'autres annonces sont prévues très prochainement.



Pour se familiariser avec le calcul quantique, Eviden a développé myQLM, une version allégée des outils logiciels de Qaptiva. myQLM est une suite logicielle de programmation quantique gratuite, accessible à qui le souhaite. Cet environnement, utilisant une interface en langage Python, peut être utilisé sur n'importe quel ordinateur personnel pour émuler des calculs quantiques, ou bien en lancer l'exécution sur des machines réelles.

Il est aussi possible d'échanger avec d'autres outils de programmation quantique (frameworks). Par exemple, on peut

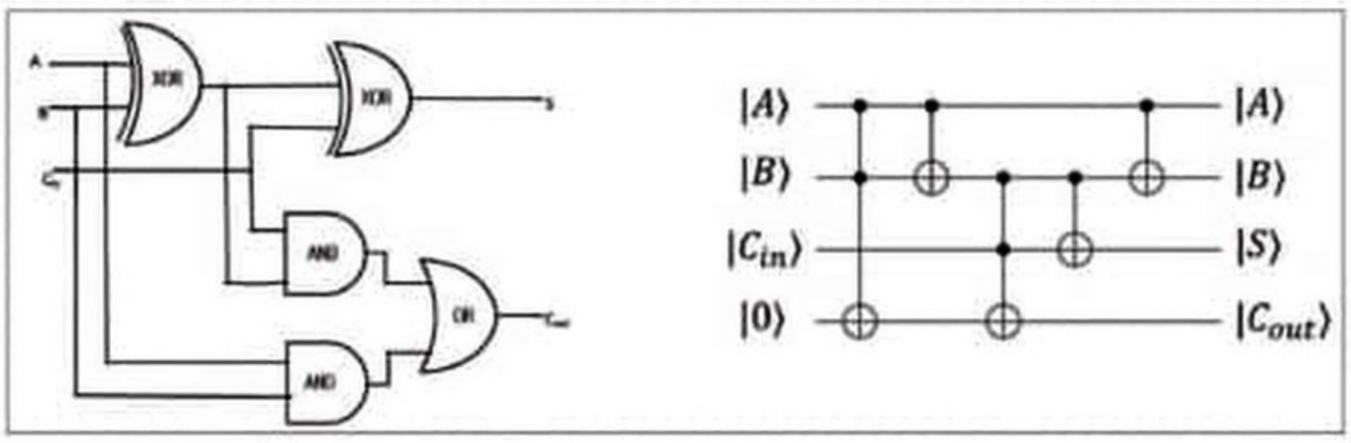
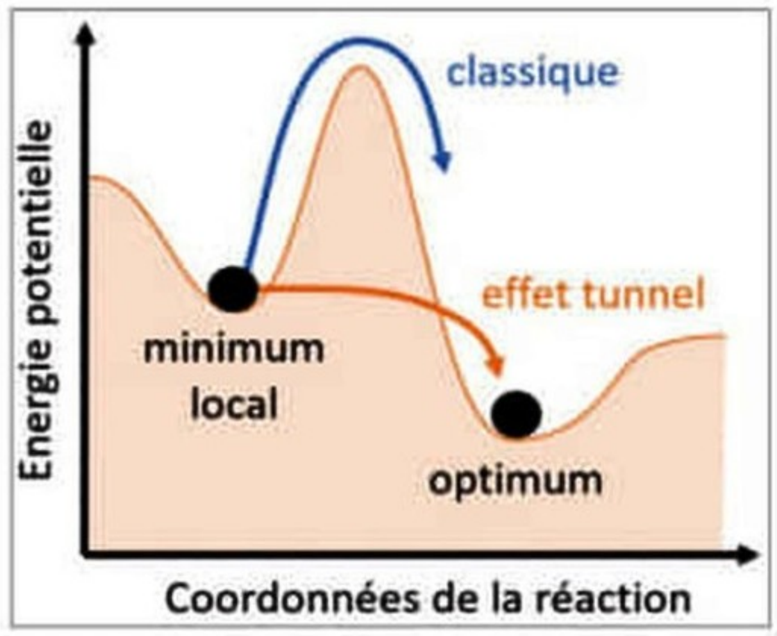


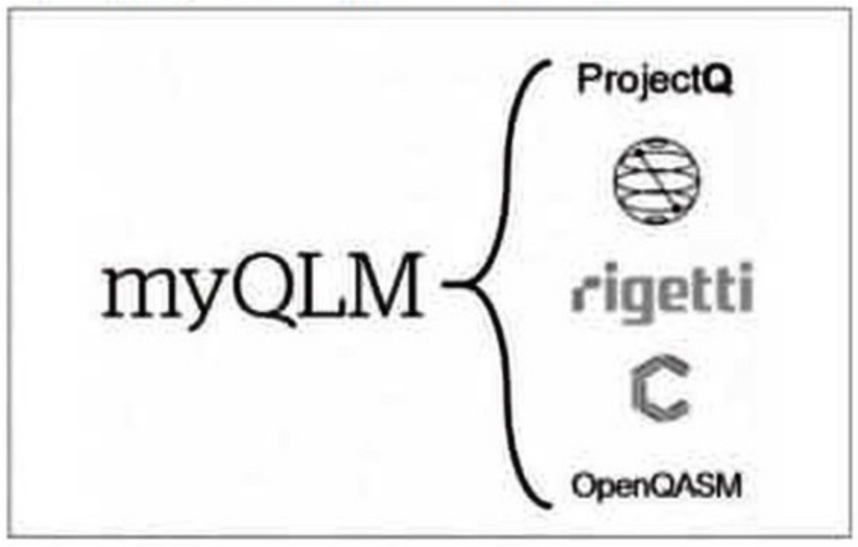
Figure 1
Exemples de circuits d'additionneurs : à gauche, circuit logique classique ; à droite, circuit quantique.

RECUIT QUANTIQUE

Le phénomène principal qu'on exploite dans le recuit quantique est l'effet tunnel. Il s'agit d'un phénomène de mécanique quantique qui permet de traverser une barrière d'énergie sans avoir à surmonter le coût nécessaire à passer l'énergie maximale de la barrière, aidant ainsi le système à échapper aux minimums locaux. Le recuit quantique tire parti de cette propriété pour trouver l'état d'énergie minimale des Hamiltoniens de type Ising.Le recuit quantique est une technique dans laquelle on fait évoluer le système quantique en question suffisamment lentement pour que le système, initialement à l'état fondamental, reste toujours à l'état fondamental pendant et à la fin de l'évolution (théorème adiabatique). L'état initial est facile à préparer ; l'état final encode la solution optimale au problème.



traduire un circuit quantique de Qaptiva vers le langage développé par IBM, Qiskit, et vice-versa.
https://myqlm.github.io/%3Amyqlm%3Ainteroperability.html



myQLM est donc l'environnement idéal pour se lancer sur le sujet de développement d'applications pour le quantique. Vous trouverez à l'adresse https://myqlm.github.io/ une documentation très complète et des notebooks Jupyter pour vous familiariser avec le sujet !

L'informatique quantique, pour quels usages ?

L'enjeu du calcul quantique n'est pas de remplacer l'informatique classique (celle que ne connaissons aujourd'hui) mais d'adresser des problèmes extrêmement compliqués, inaccessibles même aux ordinateurs HPC les plus puissants de la planète. En s'attaquant à la résolution de problèmes complexes, difficiles voire impossibles pour un ordinateur classique, l'ordinateur quantique ouvre un monde de possibilités qui impacteront différents aspects de la vie moderne, comme par exemple la conception de nouveaux médicaments ou de nouveaux matériaux pour la transition énergétique.

Mais comme évoqué précédemment la programmation quantique est fondamentalement différente de la programmation classique et il n'existe aucune portabilité applicative. En d'autres mots, il est nécessaire de repenser complètement les algorithmes classiques en une formulation quantique. C'est d'ailleurs tout un pan de la recherche algorithmique actuelle. Cette difficulté inhérente au quantique explique qu'il n'existe aujourd'hui que peu d'algorithmes quantiques comparés aux algorithmes classiques. Le site ci-après en fait la synthèse https://quantumalgorithmzoo.org/.

Citons parmi ceux-ci l'algorithme de David Deutsch, inventé en 1985, qui a montré le premier un avantage quantique théorique (mais sur un problème qui n'a pas vraiment d'utilité). Un autre algorithme quantique avantageux est celui de Peter Shor, inventé en 1994, qui permet théoriquement de factoriser des nombres entiers exponentiellement plus rapidement que le meilleur algorithme classique connu. L'algorithme de Shor permet une réduction de complexité telle que ce problème de factorisation, considéré classiquement comme insoluble en un temps pratique, devient soluble. Cet algorithme permettrait alors de craquer des clés RSA, ce qui menacerait la cybersécurité, mais pour l'instant les processeurs quantiques existants n'ont pas les capacités en taille et en précision pour y arriver.

Il existe de nombreux autres problèmes difficiles, voire considérés comme pratiquement insolubles par l'informatique classique, qui sont éligibles à une résolution quantique. De ce fait, beaucoup d'industries s'y intéressent. Citons ici quelques domaines directement concernés :

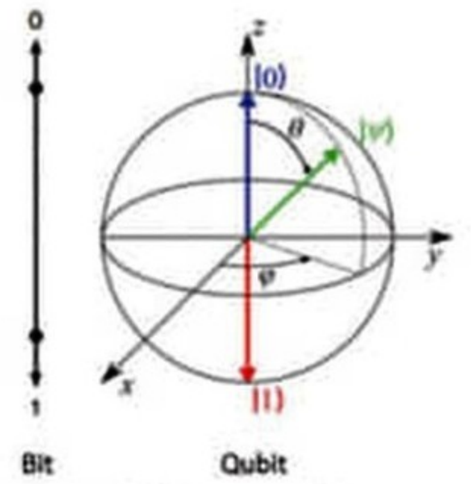
- Modélisation moléculaire : examiner la structure exacte d'une molécule pour déterminer ses propriétés et comprendre ses potentielles interactions avec d'autres molécules
- Finance : en effectuant des calculs massifs et complexes, l'ordinateur quantique pourrait ainsi réaliser des prévisions financières et permettre de mieux comprendre certains phénomènes économiques.
- Logistique : modélisation afin d'aider à optimiser la logistique et la planification des flux de travail associés à la gestion des chaînes d'approvisionnement.
- Santé : accélérer la compréhension des maladies et d'améliorer la précision des traitements

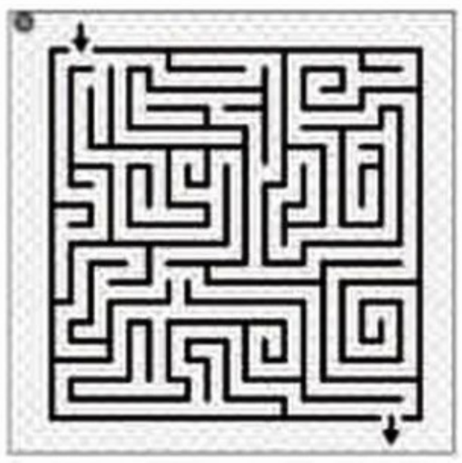
Pour aller plus loin :

- Pour tout savoir sur le quantique le livre blanc de référence de Olivier Ezratty ; un must dans votre bibliothèque https://www.oezratty.net/wordpress/2023/understanding-quantum-technologies-2023/
- Nous conseillons également le livre de Michael Nielsen et Isaac Chuang, 2010 (10e édition) (https://www.amazon.fr/Quantum-Computation-Information-10th-Anniversary/dp/1107002176). Il couvre les outils mathématiques utilisés en informatique quantique, les bases de la physique quantique, la complexité des problèmes, la mesure quantique, les algorithmes quantiques, l'impact du bruit, les corrections d'erreurs, etc. Il couvre également la distribution de clés quantiques et la cryptographie.
- Cet article tout récent, https://arxiv.org/abs/2310.03011v1, donne une étude des applications et des complexités de bout en bout des algorithmes quantiques à l'heure actuelle.

SUPERPOSITION ET INTRICATION

Comme évoqué plus haut deux notions sont fondamentales pour le développement de qubits, nécessaires à la conception d'un ordinaire quantique : la superposition et l'intrication.Les bits quantiques, ou qubits, sont les équivalents des bits du calcul classique. Un bit ne peut prendre que deux valeurs 0 ou 1. Un qubit est un état quantique, qui est une combinaison linéaire des deux états de base : $|0\rangle$ et $|1\rangle$. C'est la superposition. Il est habituel de le représenter sur la « sphère de Bloch ». L'intrication quantique, ou enchevêtrement quantique (*entanglement* en anglais), est un phénomène dans lequel deux particules forment un système lié et présentent des états quantiques dépendant l'un de l'autre quelle que soit la distance qui les sépare. Ce phénomène très contre-intuitif a été mis en évidence par Alain Aspect – Prix Nobel de physique en 2022... et membre du Conseil Scientifique d'Atos/Eviden Quantum !





Pour illustrer ces deux points et l'impact sur un ordinateur quantique, prenons l'image d'un labyrinthe comme illustré dans la figure ci-contre. En classique, pour trouver la sortie d'un *labyrinthe « infiniment complexe »* il faudra explorer séquentiellement tous les chemins ; on imagine que ce processus peut être infiniment long. En quantique on a la possibilité d'explorer tous les chemins en même temps ; c'est la superposition. Lorsque la sortie est trouvée par l'une des explorations parallèles l'information est transmise instantanément à tous les chemins et le travail s'arrête ; c'est l'intrication.En théorie le traitement quantique peut donc être infiniment plus rapide que le traitement classique !

Cas pratique : implémentation, compilation et réduction du bruit avec la solution Qaptiva d'Eviden

Qaptiva, la solution de calcul quantique d'Eviden (an Atos business), a l'avantage de permettre de coder selon différents modèles de programmation. Nous proposons ici de résoudre un problème d'optimisation (Max-Cut) avec deux de ces modèles : annealing et à portes. Vous pourrez les implémenter vous-mêmes facilement avec myQLM, une version allégée et gratuite de Qaptiva (voir article introductif p.29).
Nous verrons ensuite comment préparer un circuit quantique pour le lancer sur un processeur quantique (QPU) spécifique, comment émuler le bruit de ce QPU et le réduire afin d'améliorer son résultat.

Anne-Lise Guilmin

Quantum Computing Consultant France

Anne-Lise a fait du calcul scientifique classique dans le domaine de l'énergie. En 2022, elle a rejoint Eviden (an Atos Business) où elle est en charge d'accompagner les industriels et académiques dans leur adoption du calcul quantique. Elle participe aussi au projet BACQ qui définit des métriques pour le calcul quantique.

Doctorat – Géomatériaux, Université Paris-Est (ENPC), France, 2012

Robert Wang

Quantum Computing Consultant France

Robert a eu différents rôles de consulting notamment auprès de différentes entreprises automobile et aéronautique. En 2022, il a rejoint Eviden (an Atos Business) où il est en charge d'accompagner les industriels et académiques dans leur adoption du calcul quantique. Il participe aussi au projet BACQ qui définit des métriques pour le calcul quantique.

Formation : Doctorat – Biologie Structurale, Université de Paris-Sud, France, 1990

EVIDEN

Ces auteurs ont contribué à parts égales.

RÉSOLUTION D'UN PROBLÈME D'OPTIMISATION PAR 2 MÉTHODES AVEC MYQLM

Présentation du problème Max-Cut (coupe maximum)

Max-Cut est un problème d'optimisation combinatoire qui a été utilisé par exemple dans l'étude du chargement intelligent des véhicules électriques et dans la conception de circuits intégrés [1].

Étant donné un graphe composé de sommets et d'arêtes, l'objectif de Max-Cut est de trouver deux ensembles de sommets tel que le nombre d'arêtes entre les deux ensembles soit maximal. La séparation du graphe initial en 2 sous-graphes selon ces 2 ensembles de sommets coupera donc un maximum d'arêtes, d'où le nom Max-Cut (coupe maximum).

Considérons le graphe suivant à 5 sommets et 6 arêtes :

Problème — *Coupe maximale : 5 arêtes coupées* — *Exemple de coupe non maximale : 4 arêtes coupées*

La solution optimale correspond au minimum d'une fonction de coût qu'on construit en fonction du graphe.

Il se trouve que le problème Max-Cut appartient à une sous-classe de problèmes d'optimisation combinatoire qui se prête particulièrement bien à une résolution quantique : les problèmes d'optimisation binaire quadratique sans contrainte (dits QUBO pour Quadratic Unconstrained Binary Optimization).

Pourquoi les problèmes QUBO se prêtent bien à la résolution quantique ?

Proximité avec le modèle d'Ising

Les problèmes QUBO consistent à rechercher x, un vecteur de n valeurs binaires, qui minimise une fonction f_Q de la forme suivante :

$$f_Q(x) = \sum_{i=1}^{n}\sum_{j=i}^{n} Q_{ij}x_i x_j$$

On parle souvent de f_Q comme d'une fonction de coût puisqu'on cherche à la minimiser.

Ce sont des problèmes particulièrement difficiles : NP-difficiles selon la théorie de la complexité. En pratique, cela signifie que le temps pour les résoudre est exponentiel en la taille du graphe.

Or cette forme de problème se retrouve dans le **modèle d'Ising**, un modèle de physique statistique en mécanique quantique. Son **hamiltonien**, c'est-à-dire l'opérateur mathématique qui décrit l'évolution dans le temps du système quantique, s'écrit en effet sous une forme très proche :

$$H(\sigma) = -\sum_{i=1}^{n}\sum_{j=1}^{n} J_{ij}\sigma_i\sigma_j - h\sum_{i=1}^{n}\sigma_i$$

Où σ ne peut prendre que les valeurs 1 et -1 (spin magnétique).

En prenant $x = (\sigma+1)/2$ dans notre fonction de coût f_Q, on peut mettre f_Q sous la forme d'un hamiltonien d'un modèle d'Ising à une constante près. L'ajout d'une constante ne modifie en rien la solution optimale au problème puisqu'il s'agit d'un problème de minimisation.

Comment une machine quantique peut-elle minimiser un hamiltonien ?

Dans la nature, les systèmes physiques évoluent spontanément vers leur état de plus basse énergie : les objets descendent les pentes, les corps chauds se refroidissent, etc. Ce comportement se retrouve aussi dans les systèmes quantiques.

Un système physique qui peut être décrit par le modèle d'Ising peut être construit et paramétré pour qu'il corresponde à notre problème QUBO. Le système physique va spontanément évoluer vers son énergie fondamentale, c'est-à-dire qu'il va minimiser son hamiltonien (qu'on a choisi égal à la fonction de coût de notre problème QUBO). Voir encadré sur

le recuit quantique p.30. La « lecture » de l'état final du système physique nous permettra alors de connaître la solution optimale.

Cette méthode de programmation analogique est appelée recuit quantique (ou QA pour Quantum Annealing). C'est le premier modèle de programmation quantique qui a été utilisé dès 2007 par l'entreprise D-Wave Systems. Elle ne permet de résoudre que des problèmes mathématiques qui peuvent être exprimés comme la minimisation d'un hamiltonien d'Ising.

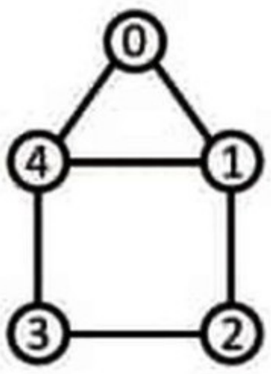
Il existe également une approche heuristique classique inspirée de l'approche QA, appelée recuit quantique simulé (ou SQA pour Simulated Quantum Annealing). Maintenant, voyons comment programmer la résolution d'un problème Max-Cut avec cette approche SQA en utilisant myQLM. Comme évoqué dans l'introduction, myQLM est la version allégée, gratuite et accessible à tous de Qaptiva™ (la plateforme d'émulation quantique d'Eviden). Son interface est en langage Python.

Résoudre le problème de Max-Cut par SQA avec myQLM

Commençons par construire un graphe qui sera la donnée d'entrée de notre problème Max-Cut. Dans cet exemple, on utilise pour cela la librairie Python networkx.

Etape 1 : Création du graphe

```
import networkx as nx
graph = nx.Graph()
graph.add_nodes_from([0, 1, 2, 3, 4])
for i, j in [(0, 1), (0,4), (1, 2), (1, 4) , (2, 3) , (3, 4)]:
  graph.add_edge(i, j)
```

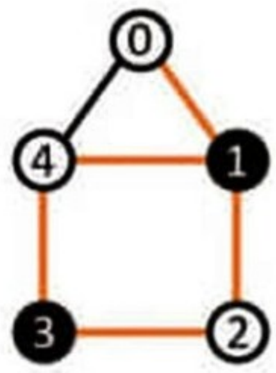


Pour ce graphe, on peut mathématiquement construire une fonction de coût dont le minimum est la solution optimale du problème. Cette fonction de coût est elle-même une donnée d'entrée pour la construction de notre modèle analogique. Ces 2 étapes sont regroupées dans le solveur MaxCutGenerator de myQLM.

Il existe d'ailleurs d'autres solveurs tout aussi pratiques pour les problèmes d'optimisation combinatoire suivants : graph colouring, graph partition, K-Clique, VertexCover, ainsi qu'un solveur générique de problème d'optimisation combinatoire [2].

Etape 2 : Utiliser le solveur MaxCutGenerator, en indiquant job_type = « annealing »

```
from qat.generators import MaxCutGenerator
from qat.qpus import SQAQPU
max_cut_application = MaxCutGenerator(job_type="annealing") | SQAQPU()
combinatorial_result = max_cut_application.execute(graph)
```



C'est bien la bonne solution, puisqu'on coupe 5 arêtes comme on peut le lire dans l'objet *combinatorial_result.cost*. On peut lire les deux sous-ensembles de nœuds dans l'objet *combinatorial_result.subsets*.

Nous allons maintenant résoudre le même problème par le modèle de programmation digitale (circuit à portes).

Résoudre le problème de Max-Cut par QAOA avec myQLM

Inspiré du recuit quantique, QAOA (algorithme d'optimisation approximative quantique) est un algorithme quantique variationnel, c'est-à-dire qu'il boucle sur une architecture hybride quantique-classique jusqu'à la convergence du calcul ou une condition d'arrêt. L'ordinateur quantique utilisé ici est du type universel à portes quantiques. L'évolution continue du recuit quantique qu'on a utilisé précédemment est ici remplacée par des itérations de calculs. Le processeur quantique prend en entrée des paramètres et évalue une énergie. A partir de celle-ci l'optimiseur classique réactualise les paramètres d'entrée du processeur quantique.

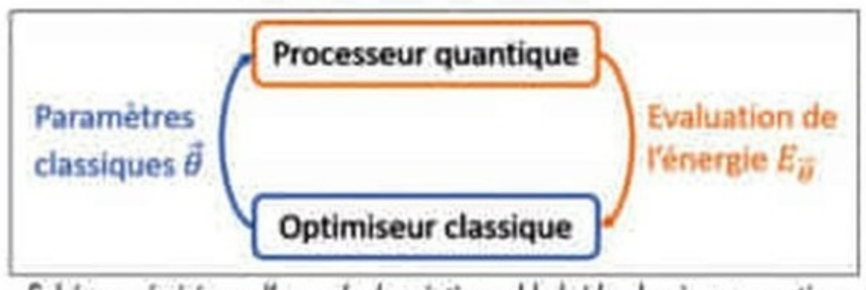


Schéma générique d'un calcul variationnel hybride classique-quantique

Qaptiva fournit plusieurs optimiseurs, vous pouvez aussi utiliser votre propre optimiseur. Plus le nombre d'itérations est grand, plus la solution est précise, mais plus le nombre des paramètres est grand, et plus le temps nécessaire pour les optimiser est long. C'est un compromis à trouver en pratique.

Etape 1 : Création du graphe ; étape identique à celle de SQA plus haut

Etape 2 : Utiliser le même solveur MaxCutGenerator, mais en indiquant job_type = « qaoa ».

Dans notre exemple, nous prenons la méthode classique COBYLA pour optimiser les paramètres gamma et beta, et l'émulateur de QPU par défaut de Qaptiva pour le calcul quantique. C'est un émulateur idéal pour émuler l'évolution d'un système quantique parfait (non bruité).

```
from qat.generators import MaxCutGenerator
from qat.plugins import ScipyMinimizePlugin
from qat.qpus import get_default_qpu
scipy_args = dict(method="COBYLA", tol=1e-5, options={"maxiter": 200})
max_cut_application = MaxCutGenerator(job_type="qaoa") | (ScipyMinimizePlugin
(**scipy_args) | get_default_qpu())
combinatorial_result = max_cut_application.execute(graph)
```

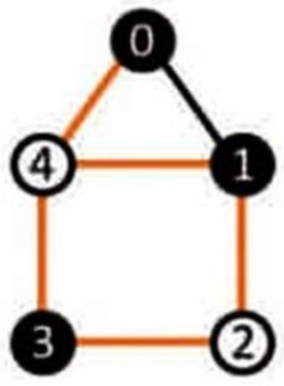

Eviden Quantum Research Lab

Thomas Ayral

Quantum Computing Research Engineer

Thomas est physicien théorique dans le domaine de la matière condensée. En 2017, il a rejoint Eviden où il est en charge de la recherche des applications du calcul quantique. Doctorat – Physique, Ecole Polytechnique, France, 2015.

EVIDEN



C'est aussi la bonne solution, puisqu'on coupe encore une fois 5 arêtes. Notons qu'il existe plusieurs solutions puisque ce graphe est symétrique et que la numérotation des sous-ensembles (représentée ici par la couleur noire ou blanche des nœuds) n'importe pas.

Nous avons donc résolu le même problème par deux approches différentes de façon très simple avec myQLM. L'étape suivante serait de les faire tourner sur un processeur quantique (QPU). Jusqu'à présent, remarquez que nous ne nous sommes pas soucié des caractéristiques d'un tel QPU. Nous avons construit des solutions génériques.

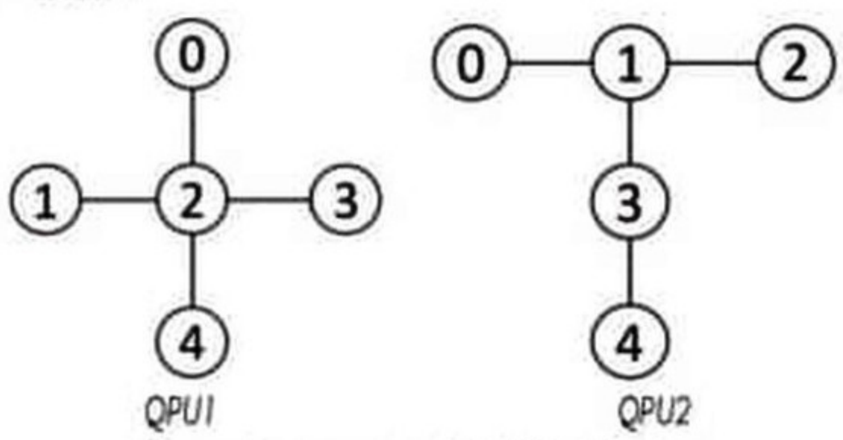
Dans la section suivante, nous allons montrer comment adapter un circuit à portes à un QPU. Le circuit que nous venons d'écrire comportant beaucoup de portes, nous illustrerons cette étape avec un circuit beaucoup plus simple. La méthode présentée reste inchangée. A partir d'ici, certaines fonctionnalités utilisées ne sont pas disponibles dans myQLM.

ADAPTATION POUR UN PROCESSEUR QUANTIQUE DEPUIS QAPTIVA

Pourquoi compiler un circuit ?

Aujourd'hui, la plupart des matériels quantiques ont des contraintes de connectivité et de contrôle.

Pour la contrainte de **connectivité**, cela signifie que les portes à deux qubits ne peuvent être exécutées que sur quelques paires de qubits. Les paires de qubits pouvant interagir définissent la connectivité d'un matériel. Par exemple, pour le QPU1 avec la connectivité ci-dessous à gauche, le qubit 0 est connecté avec le qubit 2 mais pas directement avec les autres qubits. Si on lance notre programme sur la machine et si le fournisseur de machine ne fait pas de transformation de circuit pour sa machine, alors il suffit d'une porte à 2 qubits appliquée aux qubits 0 et 1 par exemple pour faire échouer l'exécution.



2 exemples de connectivité de QPUs à 5 qubits

Qaptiva permet de réécrire un circuit donné pour l'adapter à une connectivité de qubits spécifique, grâce à des échanges optimisés de qubits.

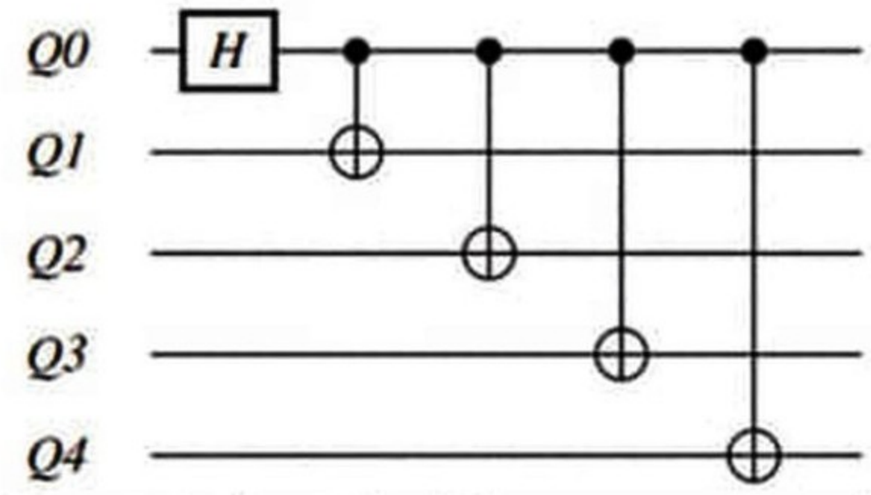
Pour les contraintes de **contrôle**, cela signifie que seules certaines opérations natives à la machine peuvent être effectuées. Est-ce limitant ? Pas nécessairement. En logique classique, on peut écrire tout circuit logique uniquement avec des portes NAND par exemple. Une machine disposant nativement de la porte NAND peut donc implémenter n'importe quel circuit. C'est une machine dite universelle. De même, une machine disposant nativement uniquement des portes NOT et OR pourra aussi implémenter n'importe quel circuit ; il faudra juste réécrire le circuit pour qu'elle puisse l'exécuter. De la même manière, il existe différents ensembles de portes universels pour les machines quantiques. Et Qaptiva permet de transformer n'importe quel circuit en un circuit qui n'utilise que les portes natives d'une machine universelle donnée. Cette transformation est optimisée au sens où on minimise le nombre de portes les plus coûteuses (en temps) du circuit final (CNOT typiquement). Il est aussi possible de prendre d'autres critères d'optimisation.

Ce processus de transformation et optimisation est une compilation, indispensable pour qu'un programme quantique soit exécutable et de façon efficace sur une machine quantique réelle. **Voir Tableau ci-contre**

Un exemple de compilation de circuit

Adaptation d'un circuit simple à 2 topologies différentes

Prenons le bout de circuit suivant qui est beaucoup plus simple que le circuit QAOA utilisé précédemment. Il s'agit de l'état GHZ (Abréviation de Greenberger – Horne – Zeilinger) à 5 qubits. L'état GHZ est une généralisation à n qubits de l'état de Bell : il produit un état qui présente la même probabilité de mesurer soit tous les qubits dans l'état $|0\rangle$, soit tous les qubits dans l'état $|1\rangle$. On le retrouve souvent en début de circuit pour préparer des intrications maximales.



Dans cette implémentation générique, on trouve par exemple la deuxième porte qui est appliquée aux qubits Q0 et Q1 (qui ne sont pas connectés dans le QPU1), et la troisième porte qui est appliquée aux qubits Q0 et Q2 (qui ne sont pas connectés dans le QPU2). Il faut donc nécessairement adapter notre circuit pour le faire tourner sur QPU1 ou QPU2.

Pour le QPU1, on comprend intuitivement que le qubit 2 qui est connecté à tous les autres qubits devrait porter l'information de Q0 de notre circuit générique et qu'il suffit d'échanger Q0 et Q2 dans le circuit d'origine. Voyons comment obtenir ce résultat avec Qaptiva grâce au plugin Nnizer. On suppose ici qu'on a déjà encodé le circuit GHZ générique dans l'objet *circuit*.

Etape 1 : Création de la topologie de la connectivité des qubits

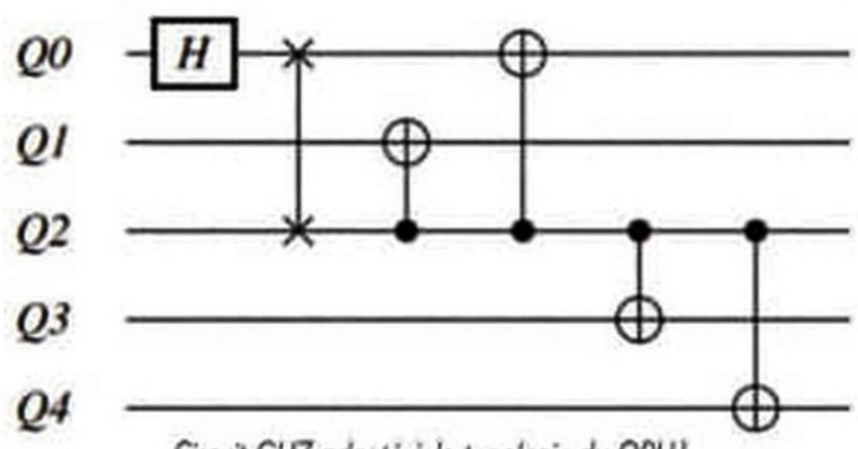
```
from qat.core import Topology
my_topology = Topology()
for i, j in [(0, 2), (1, 2), (3, 2), (4, 2)]:
  my_topology.add_edge(i, j)
```

Etape 2 : Définition du matériel avec la topologie et le nombre de qubits

```
from qat.core import HardwareSpecs
my_hardware = HardwareSpecs(nbqbits=5, topology=my_topology)
```

Etape 3 : Adaptation du circuit à cette topologie avec le plugin Nnizer

```
from qlmaas.plugins import Nnizer
from qat.core import Batch
nnizer_atos = Nnizer(method="atos")
nnized_batch_atos = nnizer_atos.compile(Batch(jobs=[job]), my_hardware)
nnized_ansatz_circuit_atos = nnized_batch_atos.jobs[0].circuit
%qatdisplay nnized_ansatz_circuit_atos --svg
```

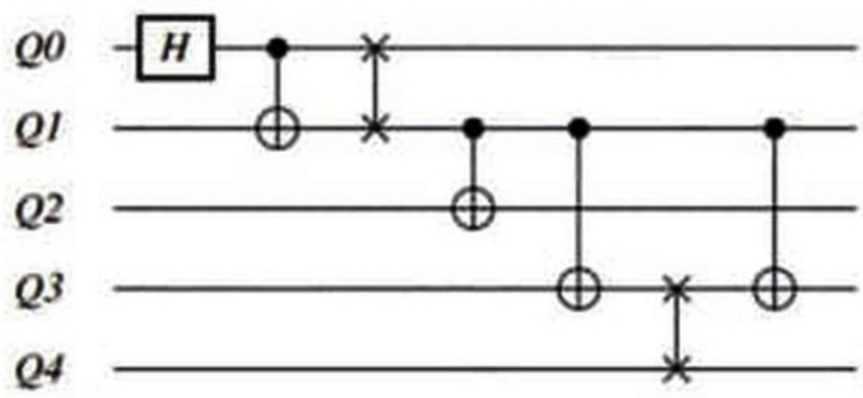


Circuit GHZ adapté à la topologie du QPU1

Pour le QPU2, quel est le nombre minimal de portes à utiliser pour adapter le circuit ? Pour répondre à cette question avec Qaptiva™, on peut reprendre le même code en changeant la topologie du QPU

```
for i, j in [(0, 1), (1, 2), (1, 3), (3, 4)]:
```

et on obtient ce nouveau circuit à 7 portes :



Circuit GHZ adapté à la topologie du QPU2

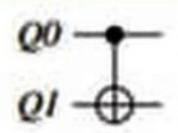
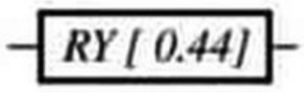
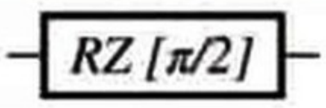
Adaptation d'un circuit simple à un ensemble de portes natives donné

Supposons maintenant que les portes natives de la machine quantique sont CNOT, ID, RZ, X et SX (racine de X), alors les portes H et SWAP du circuit ne sont pas exécutables en tant que telles par le matériel. Avec Qaptiva nous pouvons faire cette compilation nécessaire.

Dans notre cas, l'équivalence des portes H et SWAP se fait selon les égalités suivantes :

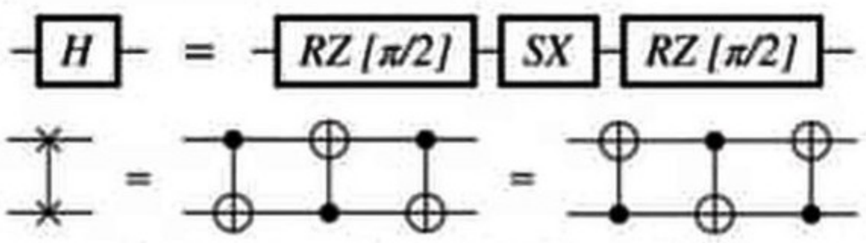
$$H = e^{i\pi/4}\, R_Z\left(\frac{\pi}{2}\right) \cdot SX \cdot R_Z\left(\frac{\pi}{2}\right)$$

Tableau

Portes quantiques utilisées dans nos exemples		
image	description	matrice
	Porte H (Hadamard) : transforme un état $\|0\rangle$ en superposition équiprobable de $\|0\rangle$ et $\|1\rangle$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$
	Porte CNOT : porte NOT contrôlée. Dans cet exemple, on applique NOT à Q1 uniquement dans les cas où Q0 vaut $\|1\rangle$	$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$
	Porte SWAP : échange la valeur de 2 qubits	$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$
	Porte R : rotation générique d'angles θ et ϕ. Dans l'image à gauche, $\theta = \frac{\pi}{2}$	$\begin{bmatrix}\cos\left(\frac{\theta}{2}\right) & -ie^{-i\phi}\sin\left(\frac{\theta}{2}\right)\\ -ie^{i\phi}\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right)\end{bmatrix}$
	Porte RY : rotation d'angle θ selon l'axe Y dans la représentation de la sphère de Bloch	$\begin{bmatrix}\cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right)\\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right)\end{bmatrix}$
	Porte RZ : rotation d'angle φ selon l'axe Z dans la représentation de la sphère de Bloch. Dans l'image à gauche, $\varphi = \frac{\pi}{2}$	$\begin{bmatrix}e^{-i\frac{\varphi}{2}} & 0\\ 0 & e^{i\frac{\varphi}{2}}\end{bmatrix}$
	Porte SX : calcule la racine de NOT. Ainsi la porte SX appliquée deux fois consécutivement est équivalente à la porte NOT.	$\frac{1}{2}\begin{bmatrix}1+i & 1-i\\ 1-i & 1+i\end{bmatrix}$
	Porte CSIGN : porte Z contrôlée. Une porte Z inverse la phase.	$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\end{bmatrix}$

$$SWAP(i,j) = CNOT(i,j) \cdot CNOT(j,i) \cdot CNOT(i,j)$$

Vous pouvez retrouver ces égalités par un calcul matriciel à partir des éléments de l'encadré précédent.



Représentation graphique des équivalences utilisées

Le plugin GateRewriter de Qaptiva permet d'appliquer ces substitutions dans le circuit. Le circuit peut ensuite être optimisé en utilisant le plugin PatternManager. Il existe encore bien d'autres plugins dans Qaptiva pour manipuler les circuits facilement [3].

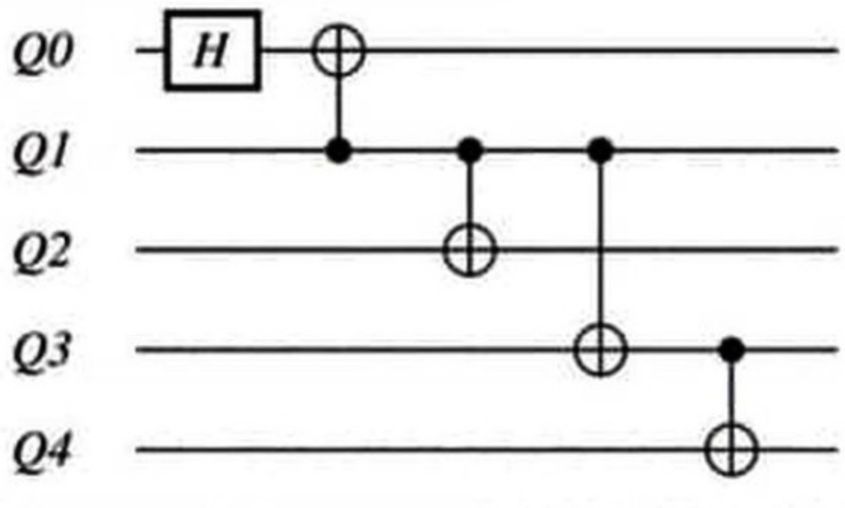
Pour aller encore plus loin, nous pouvons utiliser Qaptiva pour émuler un calcul quantique bruité. Ceci permet d'anticiper le niveau d'erreurs induit par les caractéristiques des

processeurs disponibles actuellement. Cette fonctionnalité n'est pas disponible dans myQLM, la version gratuite allégée de Qaptiva.

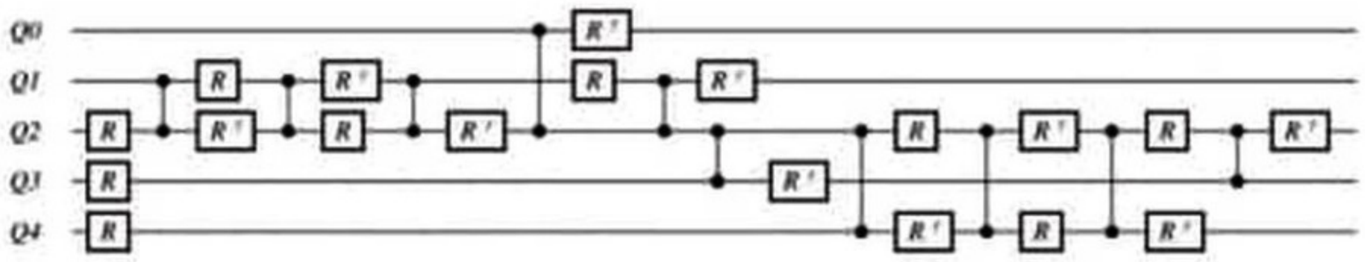
RÉDUCTION DU BRUIT AVEC QAPTIVA

Simulation de bruit d'une machine quantique IQM avec Qaptiva

Les qubits sont des systèmes quantiques fragiles. Les sources de bruit sont nombreuses : décohérence (perte d'intrication et superposition), interaction entre qubits voisins, rayons cosmiques, perturbations du champ magnétique terrestre, etc.
Une façon de lutter contre le bruit est de mettre en œuvre une correction d'erreur quantique (ou QEC pour Quantum Error Correction). Ceci est hors de portée des processeurs NISQ d'aujourd'hui. En revanche, des techniques de post-traitement peuvent être utilisées pour atténuer les effets du bruit (Quantum Error Mitigation).
Prenons le circuit suivant très simple. Cela nous permet de focaliser sur la simulation de bruit.



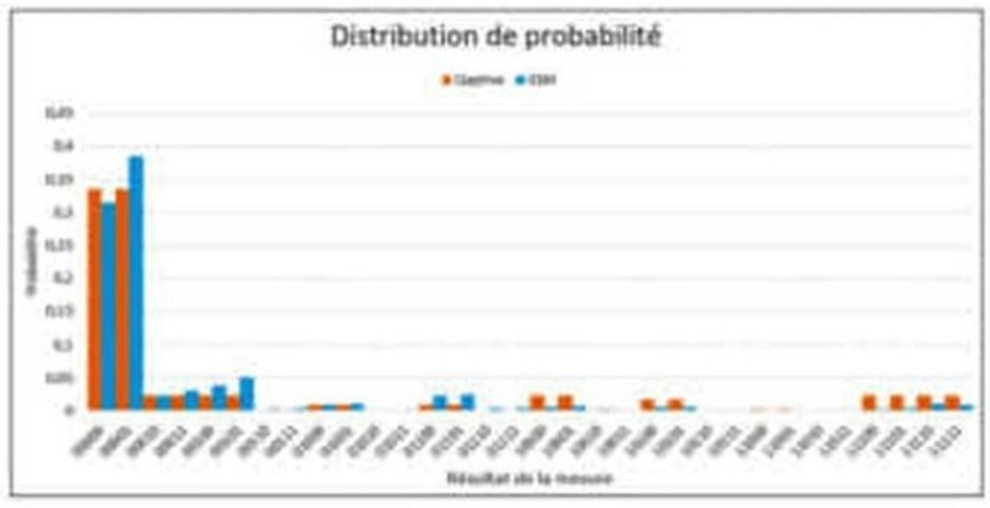
Ce circuit produit l'état quantique $(|00000\rangle+|00001\rangle)/\sqrt{2}$ selon la convention MSB sur l'ordre des qubits : le bit le plus significatif, Q_n, est à gauche $|Q_n...Q_1Q_0\rangle$. Ce circuit pourrait se réduire à sa seule porte H car les qubits sont initialisés à $|0\rangle$ et les portes CNOT ne seront donc jamais activées. Néanmoins la présence de ces portes CNOT va apporter du bruit et cela va générer l'apparition de résultats parasites, et ce sont justement ces résultats parasites que nous voulons retrouver dans cet exemple d'émulation bruitée.
Dans le cadre de notre partenariat avec la société IQM productrice finlandaise de processeurs quantiques (https://www.meetiqm.com/), nous avons accès à une de leurs machines quantiques. Cette machine a la connectivité du QPU1 et les portes natives R et CSIGN (voir encadré sur les portes). La procédure de compilation vue précédemment nous permet d'obtenir ce circuit :



Où la porte R désigne ici $R\left(\frac{\pi}{2}, \frac{3\pi}{2}\right)$ et R† est l'inverse de R.

Pour simuler le bruit avec Qaptiva, il faut disposer des spécifications techniques de la machine quantique, tels que le temps de décohérence (T1), le temps de déphasage (T2), l'erreur de mesure pour chaque qubit, le temps d'exécution et l'erreur d'exécution de chaque porte. S'il s'agit de la porte CNOT ou CSIGN, il faut avoir ces valeurs pour chaque couple de qubits.
Pour donner une idée sans perdre la généralité, nous ignorons les erreurs de mesure, construisons un modèle simplifié de matériel où chaque porte supposée parfaite est suivie d'un bruit dépolarisant avec un taux d'erreur moyen eps1 =0,01pour les portes à un qubit, et eps2 = 0,05 pour les erreurs CSIGN.

```
from qat.hardware import make_depolarizing_hardware_model
hw_model = make_depolarizing_hardware_model(eps1=0.01, eps2=0.05)
from qlmaas.qpus import NoisyQProc
noisy_qpu = NoisyQProc(hardware_model=hw_model)
job_2 = circuit_1.to_job()
result_2 = noisy_qpu.submit(job_2)
```



Le résultat de notre émulation bruitée (couleur orange) ressemble à celui obtenu directement d'une machine quantique d'IQM (couleur bleu). On y retrouve les 2 résultats attendus (00000 et 00001) avec une grande probabilité, mais aussi des résultats parasites issus des erreurs liées au bruit. Comme nous avons appliqué un bruit moyenné de la même façon sur chaque qubit, et chaque type de porte, les résultats parasites apparaissent tous avec la même fréquence dans notre émulation bruitée (orange) alors qu'ils varient en fréquence sur la machine quantique d'IQM (couleur bleue).

Atténuation du bruit avec Qaptiva

Qaptiva permet d'atténuer l'effet des bruits avec le principe ZNE (Zero-Noise Extrapolation ou extrapolation sans bruit) [4]. L'idée est de répéter le même calcul avec différentes intensités de bruit, afin de pouvoir extrapoler le résultat du calcul qu'on obtiendrait si la machine quantique était sans bruit.
Le bruit est artificiellement augmenté en multipliant le nombre de portes CNOT : puisque $CNOT^2=I$, on peut remplacer chaque porte CNOT du circuit par $2n_{pairs} + 1$ portes CNOT sans changer la fonction logique du circuit. Les portes CNOT étant considérablement plus défaillantes que les portes à un qubit, cela revient à augmenter globalement le bruit qui sera capté lors de l'exécution du circuit.
Il est possible de créer une ligne ou une courbe qui passe au plus près des points des mesures. L'extrapolation de la courbe permet de déduire la valeur « sans bruit » recherchée. Le paragraphe 8.2 de l'article [5] (dont est tiré le schéma suivant) explique également le principe et donne des exemples.

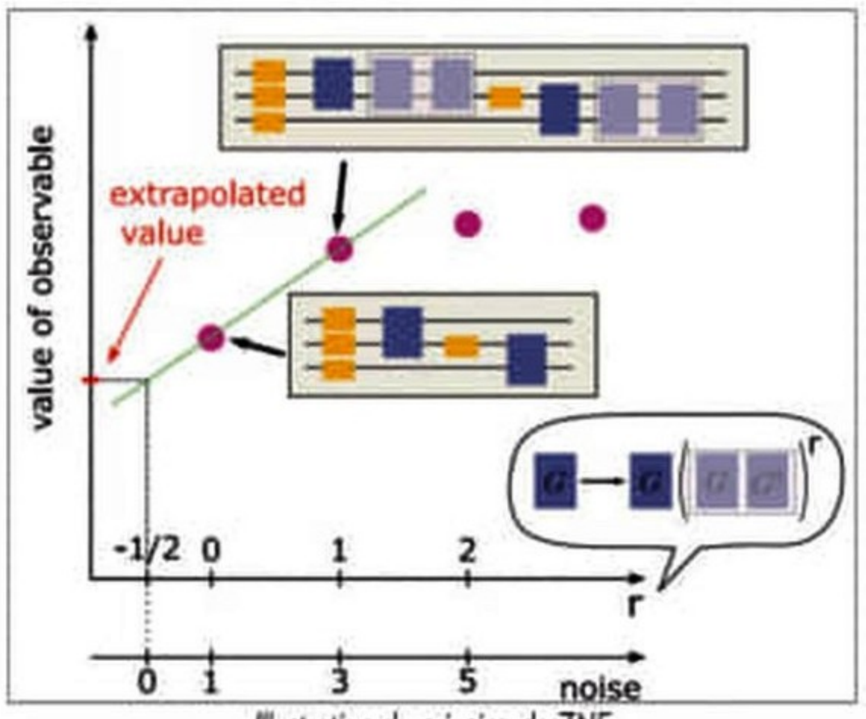
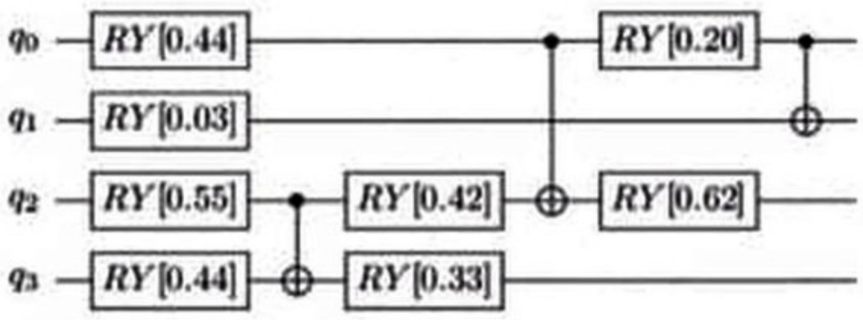


Illustration du principe de ZNE

Nous allons maintenant l'illustrer sur un petit circuit à 4 qubits pour lequel nous avions fait ce type d'étude.



Les distributions de probabilité, sans bruit et avec bruit (en utilisant la même simulation de bruit que le paragraphe précédent) sont :

ZNE nécessite le choix d'un observable pour mesurer sa valeur moyenne. Sans perte de généralité, nous allons utiliser l'observable ZZZZ. On commence par une émulation parfaite.

```
from qat.core import Observable, Term
nbqbits = 4
pauli_terms = [Term(1, op, [0, 1, 2, 3]) for op in ["ZZZZ"]]
obs = Observable(nbqbits, pauli_terms=pauli_terms)
job = bd_circ.to_job(observable=obs)
perfect_qpu = LinAlg()
perfect_res = perfect_qpu.submit(job)
E_noisefree = perfect_res.value
print("E without noise=", E_noisefree)
```

E without noise= 0.46451515841140556

Maintenant, nous définissons un QPU bruité (dépolarisant) et évaluons l'énergie avec celui-ci.

```
eps1 = 0.016
eps2 = 0.085
hw_model = make_depolarizing_hardware_model(eps1=eps1, eps2=
eps2, depol_type="randomizing", correl_type="multi_qubit")
noisy_qpu = NoisyQProc(hardware_model=hw_model, sim_method=
"deterministic-vectorized")
noisy_res = noisy_qpu.submit(job)
E_noisy = noisy_res.value
print("E noisy=", E_noisy, "(error: %s %%)" % (abs(E_noisefree - E_noisy)
/ abs(E_noisefree) * 100))
```

E noisy= 0.3238031255524545 (error: 30.29223703703704 %)

Le plugin ZNE fournira une meilleure estimation de E que le QPU bruité. Il évaluera l'énergie associée aux circuits dont le bruit est artificiellement augmenté (sans affecter la logique du circuit) grâce à l'insertion de paires de CNOT.
Il faut définir le nombre de points de régression qu'il utilisera, ainsi que le modèle de régression, qui peut être linéaire ou exponentielle. Une régression linéaire peut se contenter de deux points.
Ici nous produisons 100 points et faisons une régression linéaire :

```
ZNE_stack = ZeroNoiseExtrapolator(n_ins=1) | noisy_qpu
ZNE_res = ZNE_stack.submit(job)
E_mitigated_linear = ZNE_res.value
print(
  "E mitigated with linear ansatz:",
  E_mitigated_linear,
  "(error: %s %%)" % (abs(E_noisefree - E_mitigated_linear) / abs(E_noisefree) * 100),
)
fit_parameters = eval(ZNE_res.meta_data["ZNE_fit_parameters"])
a = fit_parameters["a"]
b = fit_parameters["b"]
linear_fit = [a * x + b for x in range(-1, 100)]
```

E mitigated with linear ansatz: 0.4070339307507018 (error: 12.374456811545972 %)

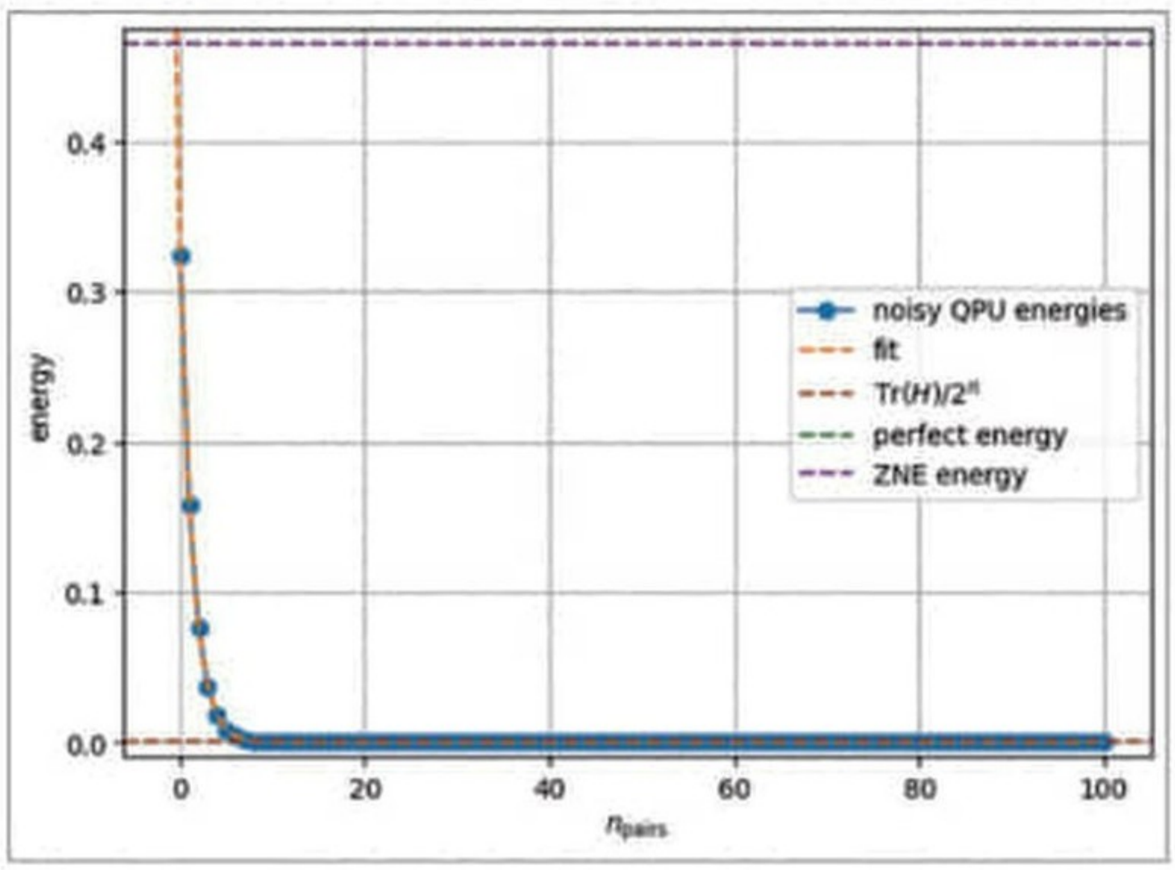
C'est plus précis ! Maintenant faisons une régression exponentielle (toujours sur 100 points).

```
ZNE_stack = ZeroNoiseExtrapolator(n_ins=1, extrap_method="exponential") | noisy_qpu
ZNE_res = ZNE_stack.submit(job)
E_mitigated_exponential = ZNE_res.value
print(
  "E mitigated with exponential ansatz:",
  E_mitigated_exponential,
  "(error: %s %%)" % (abs(E_noisefree - E_mitigated_exponential) / abs(E_noisefree) * 100),
)
fit_parameters = eval(ZNE_res.meta_data["ZNE_fit_parameters"])
a = fit_parameters["a"]
b = fit_parameters["b"]
exp_fit = [np.exp(a * x + b) + obs.constant_coeff for x in range(-1, 100)]
# be careful with the sign before the exponential, it depends on the job!
```

E mitigated with exponential ansatz: **0.464515158411405** (error: 1.195034224956 1952e-13 %)

L'extrapolation exponentielle ajuste mieux que l'extrapolation linéaire dans notre cas : la valeur extrapolée est confondue à la valeur sans bruit à un facteur 10^{-15} près.

```
plt.plot(energies, marker="o", label="noisy QPU energies")
plt.plot(range(-1, 100), exp_fit, ls="dashed", label="fit")
plt.axhline(obs.constant_coeff, ls="dashed", color="sienna", label="$\mathrm{Tr}(H)/2^n$")
plt.axhline(E_noisefree, ls="dashed", color="green", label="perfect energy")
plt.axhline(E_mitigated_exponential, ls="dashed", color="purple", label="ZNE energy")
plt.grid()
plt.legend()
plt.ylim(ymin=obs.constant_coeff - 0.01, ymax=perfect_res.value + 0.01)
plt.xlabel("$n_{\mathrm{pairs}}$")
plt.ylabel("energy")
```



Extrapolation exponentielle de l'énergie calculée pour 100 niveaux de bruits différents

	Sans bruit	Bruité	ZNE avec extrapolation linéaire	ZNE avec extrapolation exponentielle
Erreur	0%	30%	12%	10^{-13} %

Ainsi grâce à ZNE, nous sommes passés d'une erreur de 30% sur le QPU bruité à une erreur quasi nulle (10^{-13} %).

Remerciements

Nous tenons à remercier Xavier Geoffret et Jami Rönkkö de la société IQM, qui nous ont gentiment fourni le résultat de l'exécution d'un circuit sur une de leurs machines quantiques à base de qubits supraconducteurs.

Références

[1] Barahona et al., An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design, https://doi.org/10.1287/opre.36.3.493

[2] https://myqlm.github.io/04_api_reference/module_qat/module_generators.html

[3] https://qlm.bull.com/qlm-doc/04_api_reference/module_qat/module_plugins.html#circuit-rewriting-plugins

[4] https://github.com/myQLM/myqlm-notebooks/blob/master/tutorials/digital_quantum_simulations_spin_fermion/qat_fermion_zero_noise_extrapolation_plugin.ipynb

[5] Thomas Ayral et al., Quantum computing with and for many-body physics, https://arxiv.org/abs/2303.04850

Eviden Quantum Computing Consulting

Calcul quantique : n'importe où, n'importe quand

L'informatique quantique n'est pas seulement un moyen plus rapide de faire ce que nous pouvons déjà faire avec les ordinateurs classiques, elle représente un changement radical dans notre façon de penser les données, les algorithmes, le matériel et les logiciels. C'est un domaine passionnant et en pleine croissance, avec des acteurs du monde entier repoussant les limites de la technologie. Dans cet article, nous rassemblons les éléments constitutifs d'un ordinateur quantique, de la théorie au matériel, et terminons avec un programme quantique que vous pouvez exécuter aujourd'hui, maintenant, sur un véritable ordinateur quantique de Quandela via le cloud.

Jason Mueller

Jason a fait son doctorat à l'Université de Bristol, où il a construit des sources laser brillantes pour l'imagerie quantique. Il travaille actuellement dans l'engagement client chez Quandela et il enseigne l'acrobatie pendant son temps libre.

Théorie

Les ordinateurs quantiques ont besoin de trois ingrédients de base qui les distinguent fondamentalement des ordinateurs classiques que nous utilisons au quotidien : un qubit physique, et la capacité de superposer ces qubits et de les intriquer.

Un qubit, *quantum bit*, est le composant de calcul de base d'un ordinateur quantique, comme le bit binaire 0 et 1 l'est pour un ordinateur classique. Il est utilisé pour conserver et traiter des informations, mais présente certaines curiosités de la mécanique quantique, dont les bits classiques ne possèdent pas. Pour accéder à ces propriétés particulières, les qubits sont généralement fabriqués à partir des éléments fondamentaux de la nature, ces systèmes quantiques dont le comportement peut être décrit par la mécanique quantique (c'est-à-dire pouvez-vous les superposer ou les intriquer ?). Des exemples populaires incluent les supraconducteurs, les atomes, les photons. Chez Quandela, nous utilisons des photons uniques comme qubits, qui sont envoyés dans des circuits spéciaux qui effectuent des calculs, comme nous l'expliquerons plus tard.

Les deux caractéristiques qui nous intéressent le plus sont la superposition et l'intrication. Les bits classiques, ceux auxquels nous sommes le plus habitués en tant que programmeurs aujourd'hui, existent exclusivement dans les états 0 ou 1 ; ils ne peuvent exister en superposition. Toutes les données stockées et traitées par les ordinateurs modernes ne sont en réalité que de longues chaînes de 0 et de 1, activées et désactivées par de petits transistors. Les qubits ne sont pas si exclusifs : la mécanique quantique permet aux qubits d'exister à la fois comme 0 et 1 avec une certaine probabilité pour chacun (ou *amplitude* de probabilité, pour être précis), et nous écrivons les états de la particule comme une somme de $|0\rangle$ et $|1\rangle$. Une façon d'y parvenir est d'envoyer un seul photon sur un miroir qui a 50% de chances de réfléchir le photon et 50% de chances de le transmettre. Si le photon était réfléchi, nous pourrions appeler cela l'état $|0\rangle$, et s'il était transmis, nous pourrions appeler cela l'état $|1\rangle$. La partie importante de la superposition est que nous ne pouvons pas savoir si le photon a été réfléchi ou transmis à moins de le mesurer après le miroir (voir **fig. 3**, gauche), à quel point on dit que l'état du photon s'effondre. Nous pouvons écrire cet état de superposition $|s\rangle$ mathématiquement comme $|s\rangle = \alpha|0\rangle + \beta|1\rangle$, avec $|\alpha|^2 + |\beta|^2 = 1$.

La deuxième caractéristique est l'intrication, qui est un peu plus subtile. Imaginez que vous ayez maintenant deux qubits (deux photons, par exemple), que l'on dit intriqués. L'aspect essentiel est que si vous mesurez l'état de l'une de vos particules comme $|0\rangle$ ou $|1\rangle$, alors vous pouvez immédiatement décrire l'état de l'autre particule, même si vos particules sont éloignées – nous ne pouvons pas écrire l'état de ces deux particules séparément. Vous pouvez également le voir mathématiquement : un exemple de deux particules intriquées est $(|00\rangle + |11\rangle)/\sqrt{2}$, où si l'on mesure la première particule dans l'état 0 (resp. 1), alors on sait que la seconde particule se trouve dans l'état 0 (resp. 1). Cependant, ne vous y trompez pas : partager des particules intriquées avec un ami ne signifie pas que l'on peut communiquer plus vite que la lumière, limite fondamentale de la relativité, car le changement d'état des particules n'est pas directement observable ! Mais c'est un moyen de sécuriser les communications avec votre ami, en comparant astucieusement les résultats de vos mesures de particules respectives.

Il s'avère que les photons sont de bons candidats pour les qubits, car il existe plusieurs façons de les superposer ou de les intriquer, et la technologie pour le faire est maîtrisée : semi-conducteurs, puce de verre et silicium, matériaux supraconducteurs.

Matériel

Découvrez, MosaiQ. La **fig. 1** est un exemple réel d'un ordinateur quantique, auquel vous pouvez accéder sur le cloud presque 24h/24 et 7j/7. Chacun des tiroirs contient les composants essentiels du calcul quantique, de la génération de photons-qubits à la puce spéciale qui traite les informations quantiques jusqu'aux détecteurs à photon unique, ainsi que tout le matériel de support (câblage électronique, connexions optiques, etc.) – une approche modulaire élégante.

La puissance des ordinateurs quantiques de Quandela provient du générateur à photons uniques, mais pour la discussion de cet article, nous sommes plus intéressés par la logique de la puce afin que vous puissiez écrire des programmes quantiques.

Encore une fois, les circuits numériques classiques et les cir-

cuits quantiques se ressemblent : il y a des entrées à gauche, des opérations qui se produisent au milieu et une façon de mesurer le résultat. Bien que similaire aux ordinateurs modernes, il est important de comprendre à quel point les ordinateurs quantiques calculent différemment. Une fois les photons envoyés dans la puce et manipulés, tous les résultats possibles du circuit existent dans une superposition. Lorsque les photons sont ensuite mesurés, la superposition s'effondre suivant une distribution de probabilité en un résultat possible. L'objectif est de concevoir vos algorithmes de manière que les états soient plus susceptibles de se réduire à la solution correcte lorsqu'ils sont mesurés. Ensuite, en exécutant votre algorithme plusieurs fois, vous construisez des probabilités de tous les résultats possibles, qui est le résultat final du calcul quantique. Le workflow d'écriture d'un programme quantique est illustré à la **fig. 2**.

Vous partez d'une idée, votre algorithme ou expérience, qui est écrite sous forme de code. Dans notre cas, nous utiliserons la librairie open source basée sur Python Perceval, développé par Quandela spécifiquement pour écrire et exécuter des algorithmes quantiques. Ce code représente ce qu'on appelle un circuit quantique optique linéaire. Il est linéaire, car les opérations se déroulent séquentiellement de gauche à droite, et optique, car les photons sont utilisés comme qubits. Ce circuit est ensuite représenté sur la puce physique, ce qui est beaucoup plus compliqué.

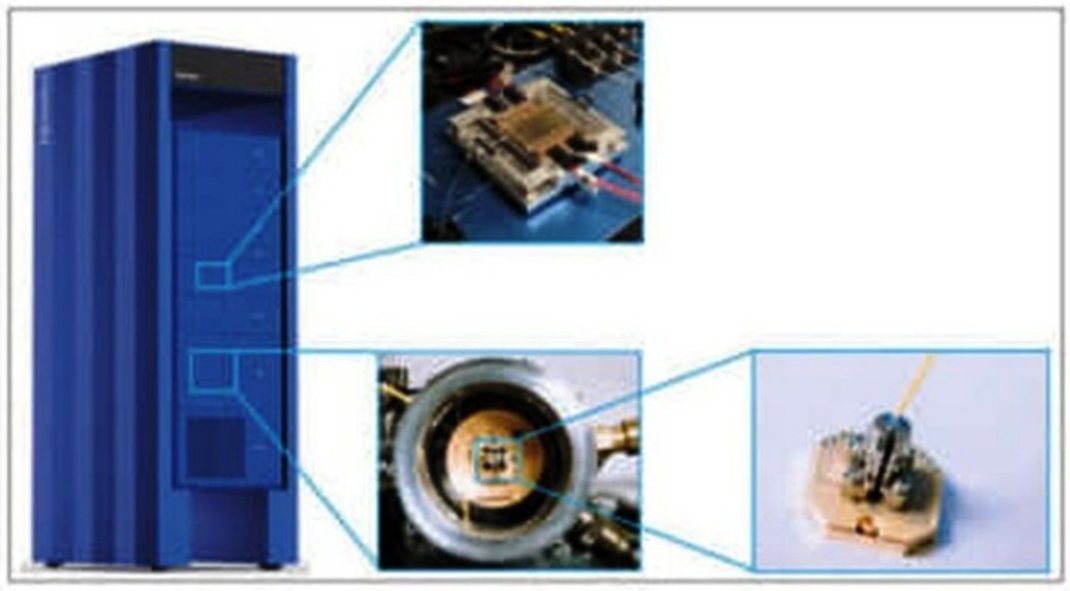


Fig. 1 : *L'ordinateur quantique photonique de Quandela, MosaiQ. Il contient l'une des sources de photons uniques les plus fiables au monde (en bas) et une puce photonique spéciale qui effectue les calculs (en haut).*

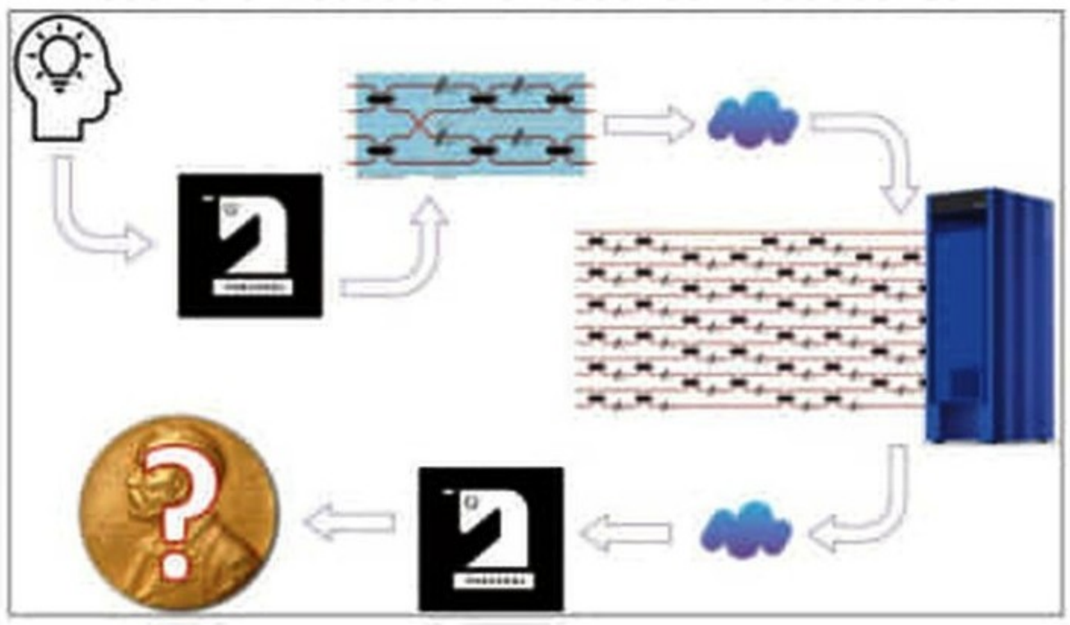


Fig. 2 : *Le workflow de programmation quantique. Une idée est écrite en Python, visualisée comme un circuit, encodée sur le processeur quantique de Quandela, et elle vous est renvoyée. Qui sait où cela vous mènera ?*

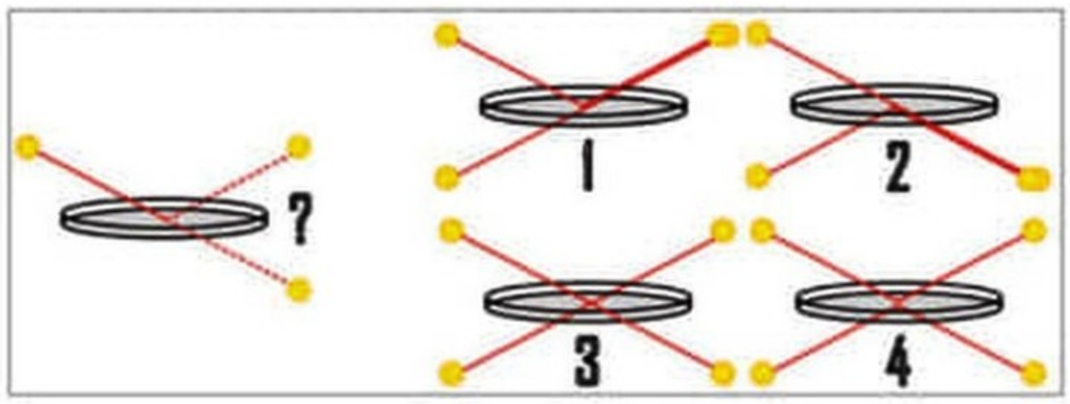


Fig. 3 : *L'effet d'un beamsplitter, et les quatre cas de l'effet HOM, avec deux photons incidents sur un miroir semi-transparent.*

Au lieu de transistors, de condensateurs et de résistances pour acheminer les électrons comme sur une puce électronique typique, les puces photoniques utilisent des *beamsplitters* (comme le miroir semi-transparent dont nous avons parlé précédemment) et des déphaseurs, pour intriquer les qubits et les mettre en superposition. Et au lieu de fils métalliques pour transporter les électrons, les photons sont envoyés à travers des fibres optiques qui sont gravées sur des puces de verre. Ces composants ressemblent en effet beaucoup à notre miroir susmentionné lorsque nous comparons la **fig. 2** et la **fig. 3**, et sont suffisants pour implémenter n'importe quelle porte logique quantique lorsqu'ils sont disposés selon le modèle régulier illustré à la **fig. 3**. Mettons cette technologie à profit maintenant !

Applications

Nous allons reproduire une expérience de physique simple, mais fondamentale, appelée l'effet Hong-Ou-Mandel (HOM). Dans cette expérience, nous envoyons deux photons uniques sur un miroir semi-transparent à 50 %, comme le montre la **fig. 3**. Il y a quatre résultats : deux où un photon est transmis et l'autre réfléchi, un où les deux photons transmettent et un où les deux réfléchissent.

Cependant, si les photons sont identiques et arrivent au miroir en même temps, la mécanique quantique nous dit que les deux cas de transmission et de réflexion s'annulent (les cas 3 et 4), donc nous nous attendons seulement à voir les cas où les deux photons se regroupent à une sortie (les cas 1 et 2). Cet effet peut donc être utilisé pour tester que les photons sont identiques, une propriété dont nous avons besoin pour faire du calcul quantique photonique.

Pour commencer, nous allons installer le package Python de Quandela pour l'informatique quantique (optique), Perceval :

```
pip install perceval-quandela
```

Nous importons ensuite les méthodes nécessaires. Nous devons importer le composant du *beamsplitter* (BS ; le miroir), les états quantiques (BasicState), un moyen de gérer les circuits, les processeurs locaux et cloud, les algorithmes qui analysent les sorties du circuit (Analyzer et Sampler) et une méthode pour afficher nos résultats (pdisplay) :

```
from perceval.components.unitary_components import BS
from perceval.utils.statevector import BasicState
from perceval.components import Circuit, Processor
from perceval.runtime import RemoteProcessor
from perceval.algorithm import Analyzer, Sampler
from perceval.rendering import pdisplay
```

Construire notre circuit est simple. Il y a deux entrées connectées à un seul miroir :

```
circuit = Circuit(2) // BS.H()
pdisplay(circuit)
```

L'entrée dans ce circuit est un photon en haut et un photon en bas, mathématiquement noté $|11\rangle$. Dans Perceval, nous

appelons cela un BasicState, qui gère les listes comme des entrées, dans ce cas [1,1]. Un Processor gère le circuit, la source (facultatif ; ici, nous laisserons cet argument vide pour supposer une source de photons parfaite) et le backend. Perceval a plusieurs backends, avec des temps d'exécution plus rapides ou plus lents selon votre application – reportez-vous à la documentation en ligne. Dans ce cas, nous utiliserons quelque chose appelé simulation forte, SLOS. Enfin, nous utilisons l'Analyzer, qui utilise le Processor pour calculer les probabilités de divers états d'entrée et de sortie.

```
input_state = BasicState([1,1])
processor = Processor('SLOS', circuit)
analyzer = Analyzer(processor, [input_state], '*')
pdisplay(analyzer)
```

	|1,1>	|2,0>	|0,2>
|1,1>	0	1/2	1/2

Les résultats qu'on obtient sur Perceval sont conformes à ce qu'on a décrit en **fig. 3**. Étant donné un état d'entrée $|11\rangle$, nous ne mesurons en effet que les résultats où les deux photons finissent en haut $|20\rangle$ ou les deux photons finissent en bas $|02\rangle$, comme prédit par l'effet HOM. N'oubliez pas que jusqu'à présent, cela n'a été qu'une simulation sur votre ordinateur classique local. Nous pouvons faire mieux; nous pouvons exécuter cette expérience à distance sur l'ordinateur quantique de Quandela via le cloud.

Pour commencer, créez un compte gratuit sur cloud.quandela.com. Au moment de la rédaction de cet article, les nouveaux utilisateurs bénéficient tous d'une heure gratuite de temps de calcul sur l'ordinateur quantique, et de dix heures gratuites de temps de calcul sur un supercalculateur pour les simulations. Une fois connecté, vous pouvez créer un token à utiliser dans votre script Python.

Le cloud computing ressemble beaucoup à l'expérience que nous venons de réaliser localement, pour faciliter votre workflow. Nous copions le token, puis définissons le RemoteProcessor en tant que simulateur ou processeur quantique :

```
token_qcloud = 'YOUR_TOKEN'
remote_qpu = RemoteProcessor('qpu:ascella', token_qcloud)
```

Ensuite, comme précédemment, nous passons le Circuit et l'état d'entrée au RemoteProcessor :

```
remote_qpu.set_circuit(circuit)
remote_qpu.with_input(input_state)
```

Cette fois, cependant, plutôt que d'utiliser un analyseur pour une simulation forte, nous échantillonnons à partir de la sortie de l'ordinateur quantique. En effet, les ordinateurs quantiques ne mesurent pas directement les probabilités, comme nous l'avons calculé localement. Au lieu de cela, ils prennent un nombre donné d'entrées comme vecteurs de 0 et de 1 et produisent des vecteurs de sortie de 0 et de 1 – c'est alors notre travail d'interpréter ces sorties comme, par exemple, des distributions de probabilité. Nous indiquons donc au RemoteProcessor le nombre d'échantillons que nous souhaitons, nommons notre job afin que nous puissions en garder une trace sur le cloud, puis l'exécutons :

```
sampler = Sampler(remote_qpu)
nsample = 1000000
remote_job = sampler.sample_count
remote_job.name = 'HOM'
remote_job.execute_async(nsample)
```

Après quelques secondes, vous pouvez récupérer vos résultats sous forme de dictionnaire

```
results = remote_job.get_results()
print(results['results'])
```

```
{
  |1,0>: 544286
  |0,1>: 576270
  |1,1>: 848
}
```

et traduisez les échantillons en probabilités :

```
nsample_qpu = sum(results['results'].values())
for keys,values in results['results'].items():
  print(str(keys) + ': ' + str(round(values/nsample_qpu,3)))
```

```
|1,0>: 0.485
|0,1>: 0.514
|1,1>: 0.001
```

Hélas, le calcul est terminé ! Le lecteur attentif aura peut-être remarqué une particularité, cependant. Alors qu'avant nous avions des états de sortie comme $|20\rangle$, maintenant nous n'avons que $|10\rangle$. Où est passé le deuxième photon ? Deux choses auraient pu arriver. Tout d'abord, l'un des photons a été perdu ou absorbé, et donc non détecté. Deuxièmement, plus important encore, nos détecteurs de photons ne peuvent actuellement pas faire la distinction entre un et deux photons ; il sait seulement qu'un certain nombre de photons sont arrivés et produit 1.

Félicitations, vous avez fait les premiers pas dans l'informatique quantique et exécutez l'une des expériences les plus importantes de la physique moderne sur un véritable ordinateur quantique !

Remarques finales

Maintenant que nous avons mené une première expérience, que pouvons-nous vraiment faire avec ces machines ? Parmi les autres exemples d'applications que vous trouverez dans ce numéro de Programmez!, des universitaires et des acteurs de l'industrie travaillent depuis des décennies sur ce problème pour développer des algorithmes utiles à court et long terme (certains sont à votre disposition dans la documentation Perceval). Les ordinateurs quantiques résolvent déjà des problèmes simples, mais intéressants en chimie, finance, science des matériaux, logistique. Par exemple, Quandela et l'ONERA travaillent ensemble pour modéliser la réaction de combustion dans un moteur thermique. Ils développent de nouvelles méthodes pour résoudre des équations différentielles sur un ordinateur quantique Quandela – l'impact à long-terme serait d'avoir des moteurs plus efficaces pour le transport aérien.

Contactez l'auteur ou l'équipe Quandela pour passer à l'étape suivante et découvrir les nombreuses applications du calcul quantique !

Solveurs quantiques en différences finies pour la simulation physique

Aujourd'hui, pour modéliser précisément les phénomènes physiques, les simulations nécessitent de plus en plus de temps d'exécution. De plus, les systèmes présentent souvent des géométries localement complexes (discontinuité, zones non régulières) qui augmentent encore le temps de calcul nécessaire pour trouver une solution acceptable ainsi que l'allocation de mémoire. C'est sur ces deux points que les algorithmes quantiques peuvent avoir un avantage. IBM a donné un accès gratuit à ses processeurs quantiques réels à la communauté scientifique pour développer cette nouvelle technologie et expérimenter des algorithmes avec la bibliothèque Qiskit de Python.

Anthony Chagneau

32 ans et originaire de Fréjus, est Ingénieur Docteur sur les projets d'algorithmie quantique chez Expleo Group à Vitrolles. Après des études universitaires en mathématiques appliquées à l'Université de Toulon, il effectue un doctorat en simulation mécanique, notamment sur les méthodes de zoom structural appliquées à des régions comportant des hétérogénéités à comportement non linéaire à l'Université de Montpellier. Par la suite, il s'est intéressé à l'informatique quantique comme solution d'optimisation et de simulation de phénomènes physiques.

La plus grande différence entre un superordinateur et un ordinateur quantique est la manière dont l'information est stockée. Dans un superordinateur, l'information est codée sous la forme de 0 et de 1, appelée bits. La valeur de chaque bit ne peut-être que 0 ou 1. Dans l'ordinateur quantique, les bits quantiques, ou qubits peuvent être à la fois 0 et 1. En outre, ces qubits sont dépendants les uns des autres, de sorte qu'une action sur l'un d'entre eux affecte instantanément tous les autres. Ces différences apparaissent lors de l'utilisation des propriétés de la mécanique quantique, telles que la superposition et l'intrication, pour résoudre certains problèmes de calcul.

Aujourd'hui déjà, certains algorithmes quantiques ont montré leur supériorité sur leurs homologues classiques. L'un d'entre eux est l'algorithme de Shor. Cet algorithme permet de factoriser un entier en temps polynomial alors que les meilleurs algorithmes classiques effectuent la factorisation en temps exponentiel.

Actuellement, les chercheurs tentent de mettre en œuvre l'algorithmie quantique pour résoudre les équations dérivées partielles (EDP) en thermique, en dynamique des fluides, en mécanique des milieux continus, en acoustique, en électrostatique et en thermodynamique.

L'objectif ici n'est pas de définir de nouveaux algorithmes quantiques, mais d'intégrer certains schémas numériques existants dans des algorithmes qui utilisent les principes de la mécanique quantique afin de définir des schémas numériques quantiques.

Matériel et méthodes

Dans notre étude, nous considérons plusieurs cas à résoudre, chacun provenant d'applications physiques différentes (convection, diffusivité thermique, propagation des ondes, simulation de vagues scélérates, rupture de barrage ...). Ces phénomènes ont été choisis pour couvrir un large spectre de domaines physiques (thermique, hydrodynamique et acoustique) résolus par des méthodes numériques bien connues.

L'idée ici est de simuler les phénomènes physiques par deux approches (classique et quantique) basées sur la même méthode. Nous comparerons la convergence et la stabilité des deux approches et la valeur ajoutée des schémas quantiques, le cas échéant.

Les phénomènes physiques mentionnés ci-dessus peuvent être modélisés par des équations aux dérivées partielles (EDP). Les solutions à trouver sont représentées par u(t,x,y) et décrivent le comportement du phénomène physique simulé; ces EDP comportent deux types d'opérateurs. Les opérateurs différentiels temporels décrivent une variation dans le temps et les opérateurs différentiels spatiaux décrivent une variation dans l'espace.

Dans la plupart des phénomènes physiques, il est impossible de trouver des solutions analytiques; il est alors nécessaire d'utiliser des calculs numériques sur ordinateur pour estimer ces solutions. Dans tous les cas, le problème à résoudre n'est plus continu, mais discret. On n'obtient alors qu'une solution numérique approximative au lieu d'une solution exacte. Pour discrétiser l'espace-temps, nous introduisons deux pas d'espace, un pour chaque direction x et y, et un pas de temps qui sera la plus petite échelle dans notre développement. Pour passer d'un problème continu à un problème discret, le domaine physique est divisé en plusieurs points où la solution est calculée. L'ensemble de ces points est appelé maillage.

Un problème dépendant du temps décrit un phénomène physique qui est modélisé à différents moments discrets t. En simulation numérique, un problème dépendant du temps peut être formulé implicitement ou explicitement. Les méthodes d'Euler implicite et explicite ont été choisies parce qu'elles sont faciles à mettre en œuvre et que, comme pour les méthodes d'approximation numérique, la comparaison de tous les différents schémas numériques existants n'est pas l'objectif de ce document.

Le schéma explicite est intrinsèquement plus facile à mettre en œuvre que le schéma implicite, puisque le flux est explicitement connu, mais il est souvent moins précis que le premier. De plus, la méthode explicite peut être numériquement instable si le pas de temps ne vérifie pas la condition de stabilité de Courant-Friedrichs-Lewy (CFL), contrairement au schéma implicite qui est inconditionnellement stable.

Solveurs quantiques

Dans cette section, nous utiliserons ces deux méthodes pour construire les schémas numériques quantiques. Le formalisme est différent du formalisme classique, car, en plus d'un facteur aléatoire dans les résultats, toutes les valeurs de la

solution seront incluses entre 0 et 1, c'est-à-dire que la solution sera mathématiquement normalisée avec une norme par rapport à l'espace de Hilbert, sans tenir compte du signe. Pour dimensionner la solution et trouver le signe de ces valeurs, nous aurons besoin d'une tomographie d'état quantique. Ce processus permet de reconstruire un état quantique à partir de mesures. La mesure quantique donne un ensemble de données représentant l'état quantique du système.

Schéma quantique implicite

Comme pour la version classique, le schéma quantique implicite reste toujours un système linéaire. Dans l'algorithme quantique, pour obtenir des informations sur la solution de « certain » systèmes linéaires, on utilise l'algorithme HHL.

En informatique quantique, le système linéaire peut-être reformulé par le fait de trouver un état quantique tel que si on lui applique une porte ou un ensemble de portes, que nous notons A, l'état se retrouve dans un second état. Pour cet algorithme, nous regrouperons les qubits en trois ensembles que nous appellerons *registres*. Un registre quantique est donc un système comprenant plusieurs qubits. Nous avons un registre de nb qubits, qui contiendra l'information du second membre au début de l'algorithme puis l'information sur l'état solution à la fin. Un second registre de nl qubits sera utilisé pour stocker les informations relatives à la matrice A du système linéaire. Un registre auxiliaire de na qubits est nécessaire pour les calculs est sera utilisé pour diriger la solution. L'algorithme HHL peut être décomposé en cinq étapes :

1. Placer le premier registre dans l'état correspondant au second membre.
2. Placer le second registre dans l'état engendré par l'estimation de la phase quantique.
3. Mettre le registre auxiliaire dans un état de superposition, en appliquant des rotations contrôlées, conditionnées par les valeurs propres.
4. Mettre le deuxième registre dans son état initial en décompilant.
5. Mesurer les états et ne conserver que les états où le registre auxiliaire est dans le bon état quantique.

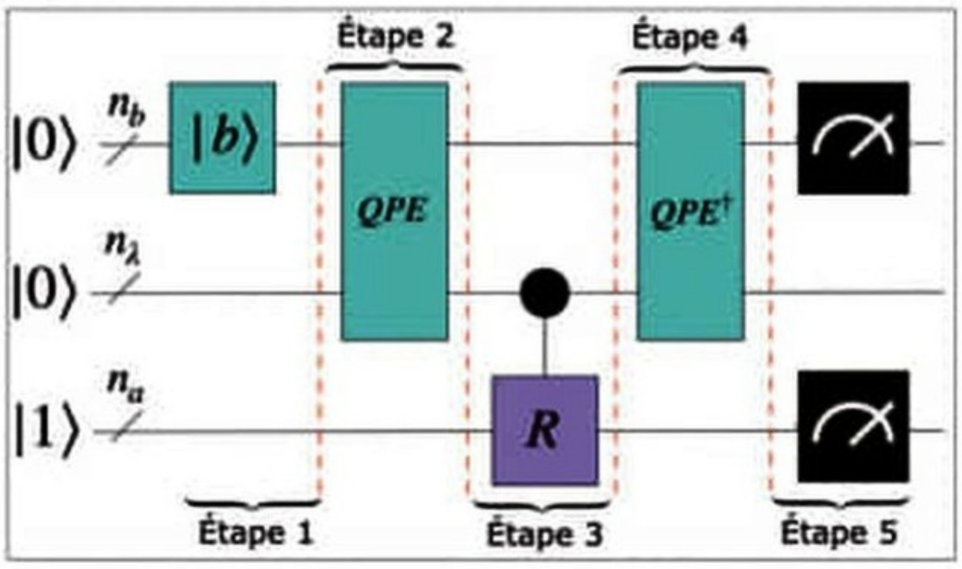


Figure 1 : *Circuit quantique de l'algorithme HHL*

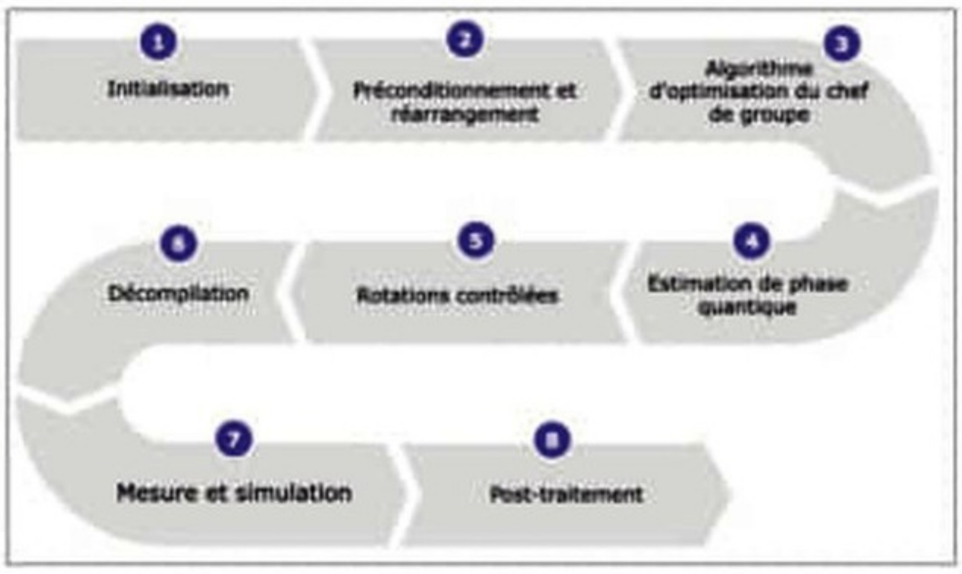


Figure 2 : *Diagramme de l'algorithme HHL étendu*

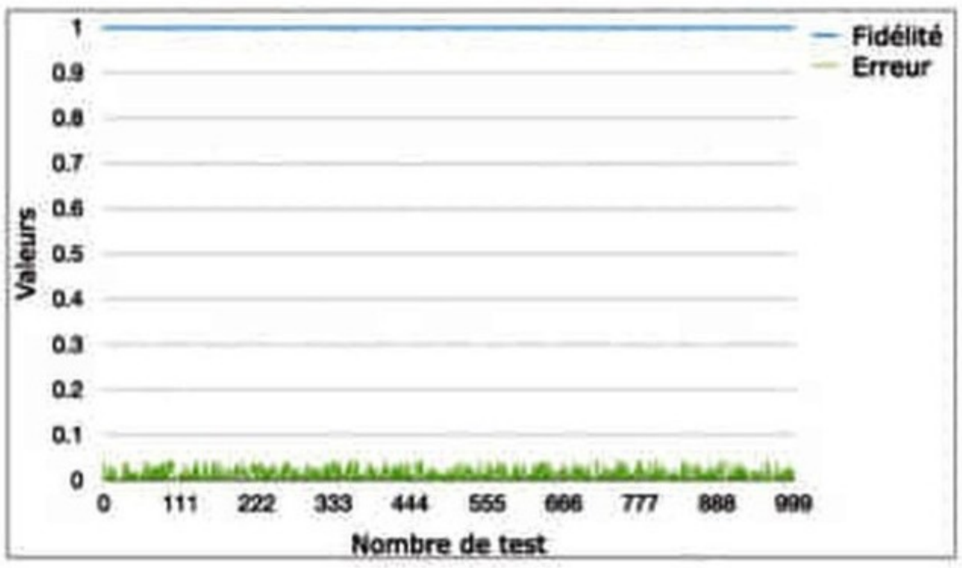


Figure 3 : *Fidélité et distance de l'algorithme HHL étendues.*

Figure 1

L'algorithme HHL est l'un des principaux algorithmes de l'informatique quantique. Il fait toujours l'objet de recherches actives. En plus de fournir des informations sur la solution de l'un des problèmes les plus courants en sciences appliquées, il fait appel des outils importants de l'informatique quantique, tels que la simulation de l'Hamiltonien.

Cependant, l'algorithme HHL d'origine n'est en effet pas utilisable en l'état pour les systèmes que nous voulons résoudre, car il ne fournit des informations que sur la solution de quelques systèmes linéaires. En 2015, Aaronson a énuméré différentes exigences pour que l'algorithme HHL soit opérationnel.

Pour utiliser l'algorithme HHL, le système linéaire doit satisfaire les conditions suivantes :

- la taille du système linéaire doit nécessairement être une puissance de 2
- la matrice du système linéaire doit vérifier les propriétés suivantes :

1. Être Hermicienne
2. Être définie positive
3. Avoir ses valeurs propres doivent être comprises entre 0 et 1

Pour entrer dans les conditions d'utilisation, on utilise un préconditionnèrent et un réarrangement du système linéaire. Maintenant que l'algorithme HHL étendu est utilisable pour les systèmes physiques que nous allons résoudre, le reste de l'algorithme est similaire à celui d'origine. **Figure 2**

L'algorithme HHL étendu a été testé sur 1000 matrices réelles générées aléatoirement à raison de 2x2. Pour comparer deux états trouvés avec l'algorithme HHL étendu, nous pouvons définir deux quantités :

- La fidélité notée F.

Cette fidélité est une mesure qui décrit la proximité entre deux états. Elle exprime la probabilité qu'un état réussisse un test permettant de l'identifier avec un autre état. Plus la fidélité est proche de 1, plus les deux états sont indissociables.

- La distance, notée D.

Cette distance est la généralisation quantique de la distance statistique et est définie par les bornes supérieures et inférieures des inégalités de Fuchs-van de Graaf.

La borne supérieure de la distance est un équivalent de la norme L2 pour les vecteurs. **Figure 3**

L'erreur est inférieure à 5 % et la fidélité très proche de 1. L'algorithme étendu peut alors résoudre n'importe quel système linéaire 2x2.

Toute matrice A peut-être décomposée sous la forme A= R-M-N, ou˚ R est la matrice diagonale de A, -M est la matrice triangulaire inférieure de A avec une diagonale nulle et -N est la matrice triangulaire supérieure de A avec une diagonale nulle. L'algorithme de Jacobi quantique convergera vers l'état solution du système classique. Le processus du schéma implicite quantique est décrit dans la **Figure 4**.

D'une part, l'inconvénient de cette méthode, comme la méthode de Jacobi classique, est qu'elle prend beaucoup de temps, mais d'autre part, l'avantage est que cette méthode donne un schéma inconditionnellement stable qui convergera vers la solution correcte, quelles que soient les données d'entrée.

Figure 4 : *Processus du schéma quantique implicite*

Figure 5 : *Processus du schéma quantique explicite*

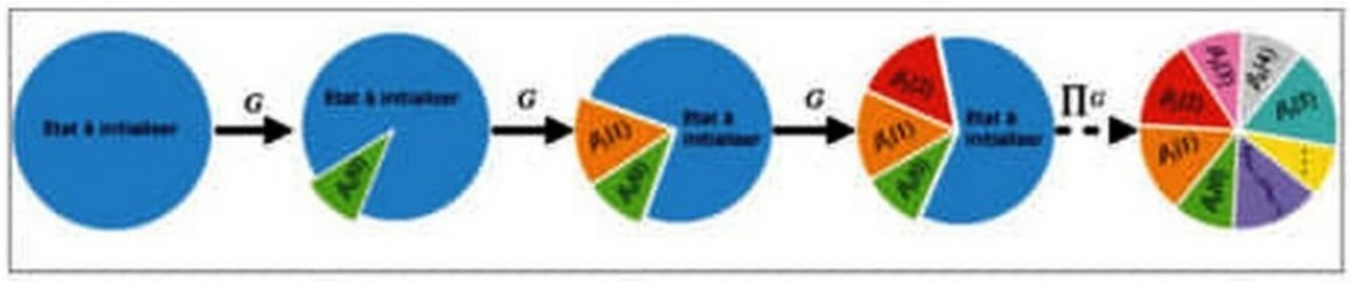


Figure 6 : *Cadre du schéma quantique explicite*

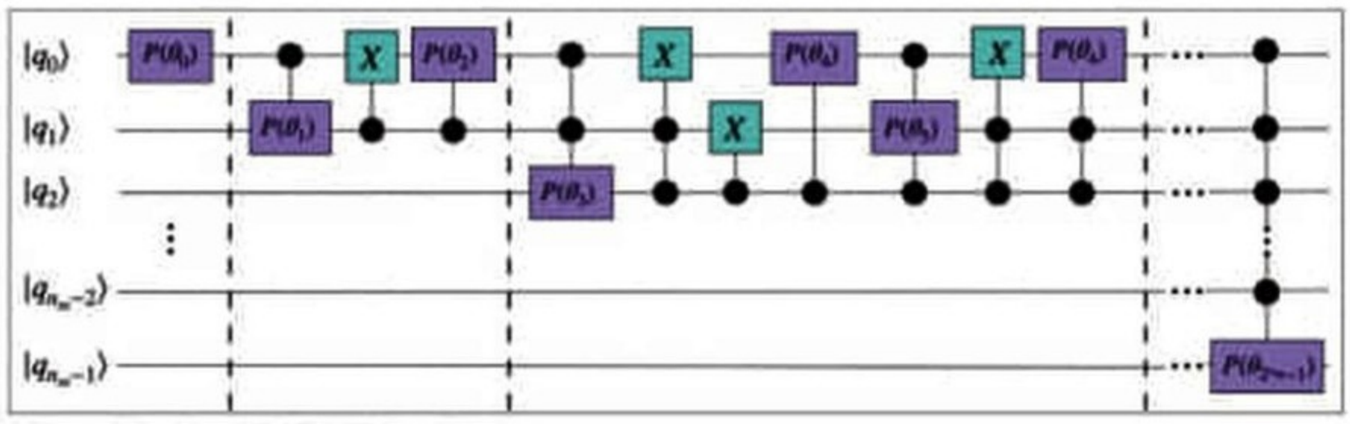
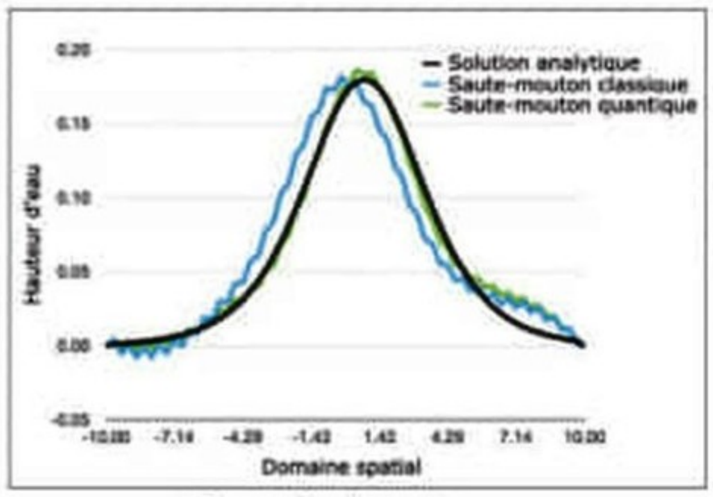


Figure 7 : *Circuit de l'algorithme du gâteau.*



Propagation des ondes

(b) Erreur L2 des schémas classiques et quantiques

Figures 8 : *Résultats pour la simulation d'une vague scélérate*

Schéma quantique explicite

Nous allons maintenant adapter le schéma explicite numérique à l'algorithme quantique. Le schéma explicite quantique suit la définition du schéma explicite classique avec des états quantiques au lieu des vecteurs. Cette fois, nous avons directement la solution au pas de temps n+1 en fonction de la solution au pas précédent. En d'autres termes, l'état du système quantique au temps n+1 est égal à l'état au temps n auquel une porte a été appliquée. Ainsi, pour définir le schéma explicite quantique, il suffit de construire la porte en question. Le processus du schéma explicite quantique est décrit dans la **Figure 5**.

L'objectif du circuit explicite du schéma quantique est que les différentes portes puissent décrire le schéma numérique classique pour un point particulier du maillage. Pour ce faire, nous partirons du premier état de la base quantique et nous initialiserons progressivement chaque état de la base par une valeur correspondant au schéma numérique. **Figure 6**

Il suffit maintenant de définir la porte à utiliser. La porte en question ne doit influencer qu'un seul état de base afin de contrôler la distribution et de respecter le schéma numérique choisi. C'est pourquoi nous choisissons la porte de changement de phase multi-contrôlée P, accompagnée ou non d'une ou plusieurs portes correspondant à l'instruction logique Not, et notée X, multi-contrôlée, afin de cibler le qubit de travail. Le circuit relatif à ce processus est décrit par la **Figure 7**.

Les angles des portes de changement de phase *P*, correspondent à la définition du schéma numérique. Maintenant que les deux types de schémas quantiques sont construits, nous allons les tester pour la simulation de nos phénomènes physiques.

Résultats numériques

Tous les programmes quantiques ont été mis en œuvre sur le simulateur quantique d'IBM appelé *ibmq_qasm_simulator*. Tous les résultats quantiques ont été vérifiés en effectuant les calculs sur le véritable processeur quantique d'IBM à Oslo, appelé *IBM_Oslo*. La **Figure 8** montre que le schéma quantique est plus proche de la solution analytique que le schéma classique. Si l'on considère l'ensemble de la simulation, le schéma classique commence à créer des oscillations avant le schéma quantique, ce qui déplace la solution classique. Nous avons constaté que le schéma quantique donne un temps de résolution plus faible que le schéma classique lorsque le nombre d'éléments est très élevé. Les schémas quantiques

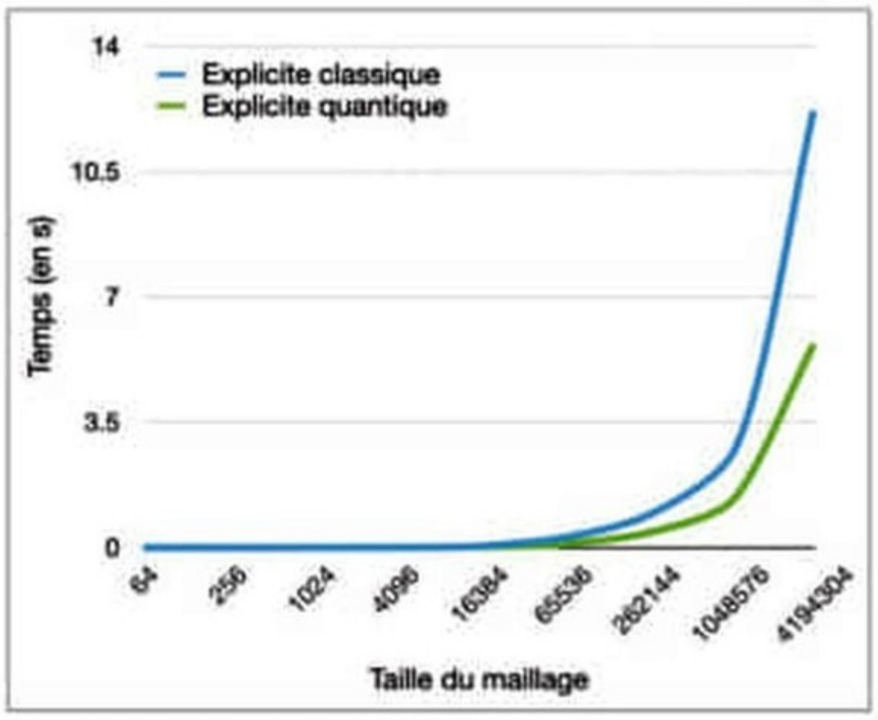


Figure 9 : *Temps de simulation*

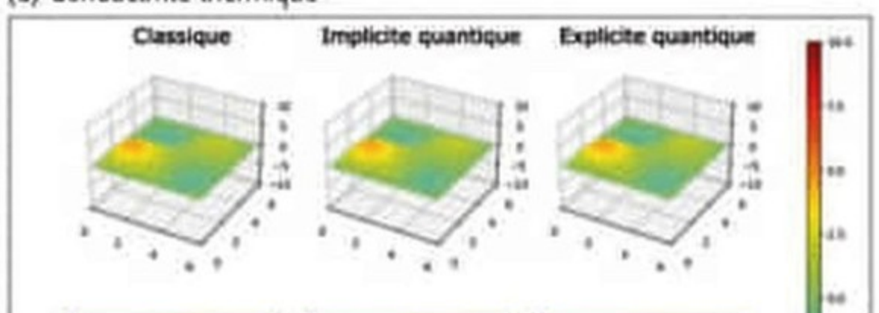
ne sont intéressants que pour un grand nombre de données, c'est-`a-dire lorsque le phénomène est complexe et/ou lorsque le maillage est très fin. **Figure 9**

Les schémas numériques pour les phénomènes physiques seront toujours construits avec la méthode des différences finies pour une mise en œuvre plus facile. Tous ces schémas sont centrés de sorte qu'ils divergent plus rapidement que les schémas non centrés. Pour la diffusivité thermique, la propagation des ondes et la dynamique des gaz, les figures suivantes montrent que les schémas numériques classiques et quantiques donnent les mêmes solutions. **Figure 10**

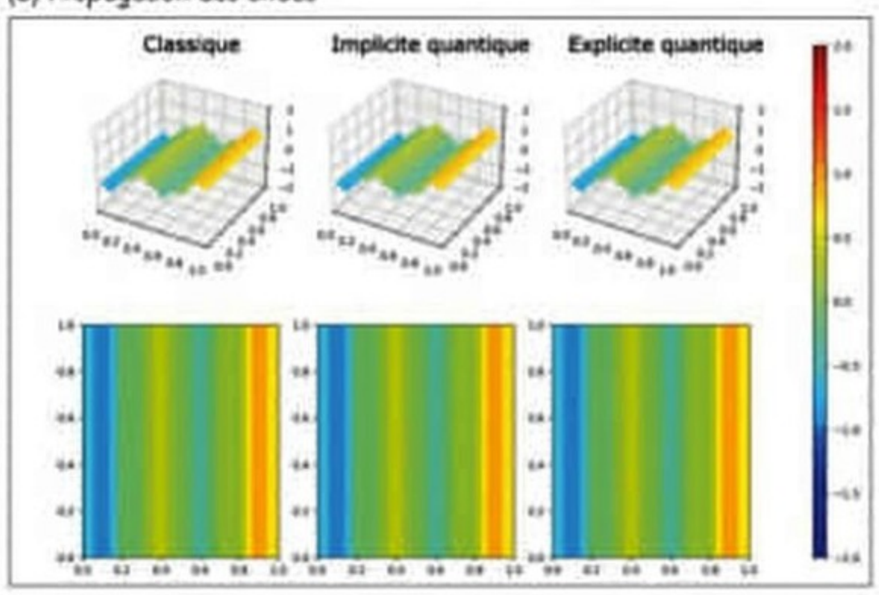
En raison de l'énorme temps de calcul de la résolution avec le schéma implicite quantique, seul le schéma explicite quantique est conservé pour les phénomènes de vague scélérate en eau peu profonde et de rupture de barrage. Pour la simulation de la vague scélérate en eau peu profonde, la figure suivante montre que les deux approches donnent des résultats totalement différents. Le schéma classique est rapidement instable pendant la simulation. En effet, une erreur de calcul s'accumule au cours de la simulation et fait diverger la solution, alors que le schéma quantique reste stable tout au long de la simulation. **Figure 11**

On peut voir sur la **figure 12** que pour le phénomène de rupture de barrage, les deux schémas ne donnent pas les mêmes résultats pour la hauteur d'eau ainsi que pour sa vitesse. Contrairement au phénomène de vague scélérate en eau peu profonde, le schéma quantique n'est pas stable, mais semble être TVD (Total Variation Diminishing), c'est-`a-dire que les oscillations sont contrôlées et que le schéma ne diverge pas. Par conséquent, pour le même schéma numérique, l'approche quantique donne au moins un résultat similaire à l'approche classique, et parfois de meilleures solutions dans certains cas.

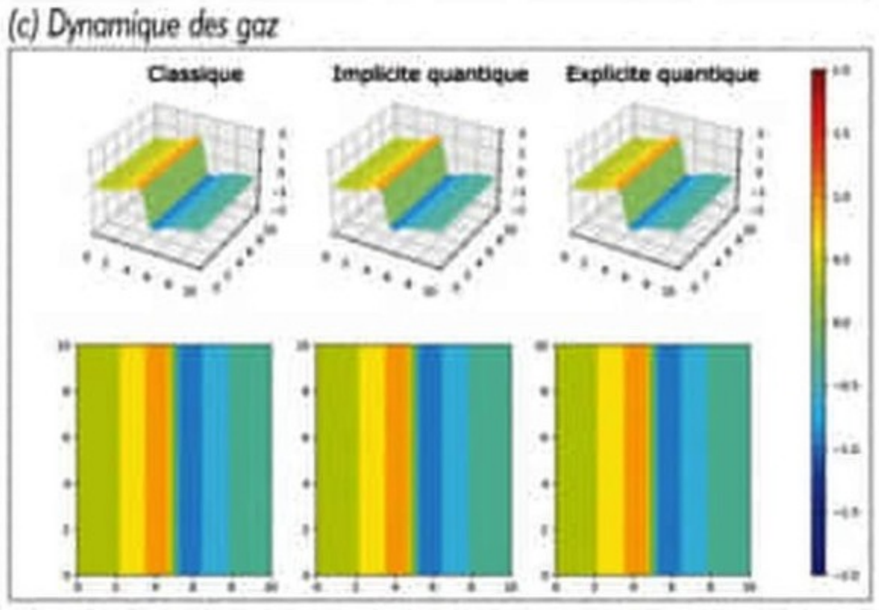
(a) Conductivité thermique



(b) Propagation des ondes



(c) Dynamique des gaz



Figures 10 : *Comparaison de la méthode classique et de la méthode quantique pour trois phénomènes physiques*

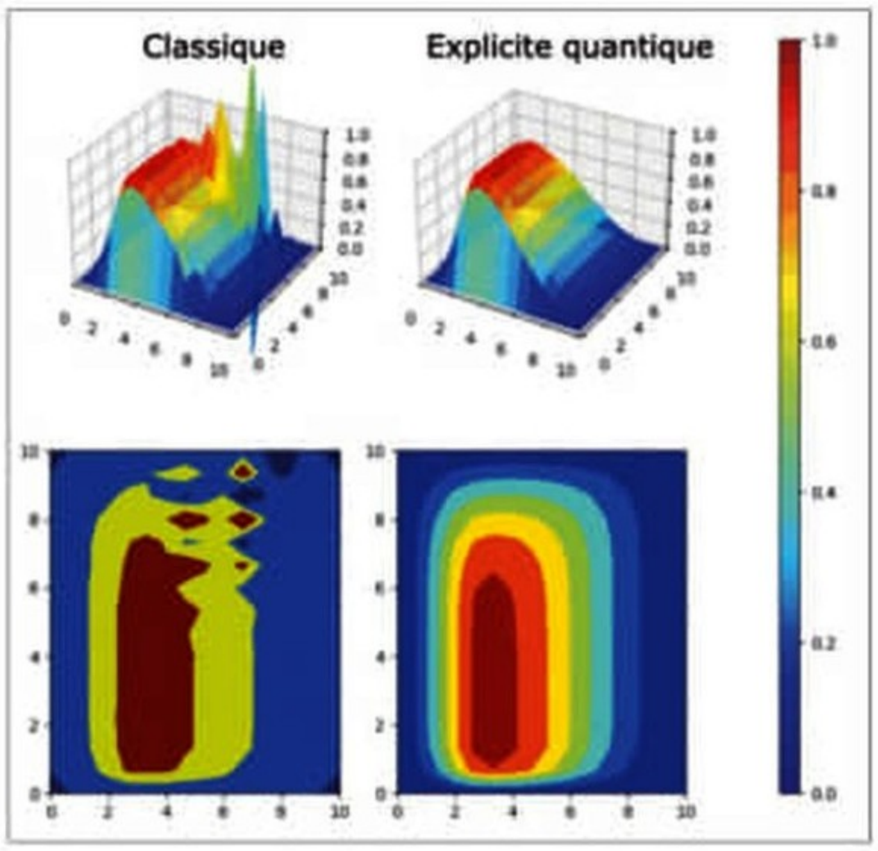
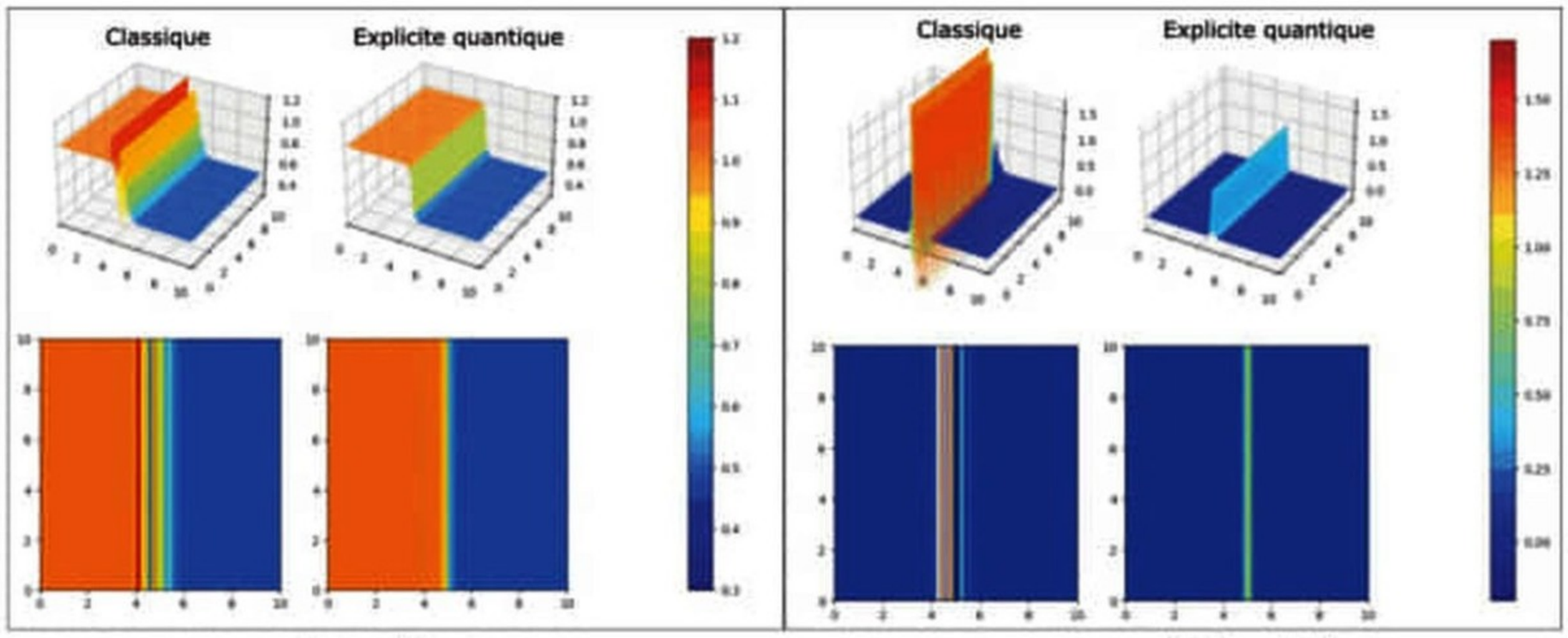


Figure 11 : *Comparaison des méthodes classique et quantique pour la simulation d'une vague scélérate*



Hauteur de l'eau *(b) Vitesse de l'eau*

Figures 12 : *Comparaison de la méthode classique et de la méthode quantique pour la simulation de la rupture d'un barrage*

Exolegend #2

hackathon_

by EXOTEC

Du 16 au 18 février 2024

Inscrivez-vous!

BTWIN VILLAGE

Le quantique pour les problèmes de la chaîne logistique

Dans les chaînes logistiques, on trouve de nombreux problèmes difficiles à résoudre :

- Gérer la flotte de camions pour les livraisons ;
- Gérer les tournées des commerciaux ;
- Organiser la production pour produire mieux et moins cher.

Les logiciels utilisés pour résoudre ces problèmes sont complexes et obtenir une bonne solution (un plan de production ou une tournée de livraison) nécessite des puissances de calcul importantes. Avec l'informatique quantique, on pourrait envisager de résoudre ces problèmes dans des temps de calcul beaucoup plus courts et ainsi imaginer gérer différemment les ressources de l'entreprise : calculer plus vite et fournir de meilleures solutions. Nous allons voir comment et pourquoi.

1 Les machines quantiques

Les débuts

Richard Feynman a jeté les bases de l'ordinateur quantique dans les années 80, et l'ensemble de ses contributions est aisément accessible en ligne. Les années 90 ont vu émerger des avancées plus axées sur les algorithmes que sur la physique, conduisant ainsi aux premiers algorithmes quantiques dédiés à un problème particulier.

De nouveaux algorithmes

En complément de cette série d'algorithmes développés dans les années 90, il convient d'ajouter des algorithmes itératifs plus récents qui, conceptuellement, ressemblent à des variantes quantiques de méta-heuristiques couramment utilisées pour résoudre des problèmes complexes, comme ceux rencontrés dans la gestion de chaînes logistiques, par exemple. Pour simplifier, on pourrait presque parler d'une version quantique du recuit simulé, bien que cela soit une simplification excessive. L'intérêt croissant porté par la communauté optimisation au quantique réside essentiellement dans deux éléments concomitants :

- Le **premier élément** clé est la mise en service des premières machines quantiques, qui permettent d'exécuter des codes informatiques liés à ces algorithmes quantiques. Parmi ces machines, on peut mentionner les systèmes de D-Wave (avec 5000 qbits dédiés), les dispositifs d'IBM (dotés de 100 qbits universels) sans oublier les simulateurs, dont celui d'Eviden/Atos
- Le **second aspect** concerne la proposition de nouveaux algorithmes itératifs de type méta-heuristiques quantiques. Ces algorithmes itératifs "quantiques" peuvent permettre de résoudre efficacement ce type de problèmes.

Intérêt du quantique pour l'Optimisation de la chaîne logistique

L'intérêt de l'informatique quantique est de résoudre dans des temps de calcul beaucoup plus courts des problèmes qui nécessitent actuellement des heures de temps de calcul.

Les responsables logistiques rêvent depuis longtemps de résoudre leurs problèmes de distribution en quelques secondes ce qui leur permettrait d'être réactif aux aléas (accidents, trafic perturbé, commande annulée).

Imaginons un commercial qui doit visiter un certain nombre de clients et revenir à son point de départ, en parcourant la plus courte distance possible

Sur la **Figure 1**, le problème se compose de 7 clients : le client 1 de coordonnées (2;0), le client 2 de coordonnées (2;2), le client 3 de coordonnées (0,3), le client 4 de coordonnées (-1;1), le client 5 de coordonnées (-3;-1), le client 6 de coordonnées (0;-2) et le client 7 de coordonnées (3;-4).

Deux boucles « for » imbriquées suffisent à calculer les distances **distance[i][j]**, qui donnent une matrice symétrique dans notre cas. Il est possible de calculer la distance kilométrique avec un logiciel de cartographie, ce qui peut se faire sans difficulté avec les web services ad hoc. **Figure 2**

Une solution est une suite de villes qui commence à 0 (le dépôt) et qui se termine au dépôt (le nœud numéro 0).

Une_Solution = [0]*(w+1)

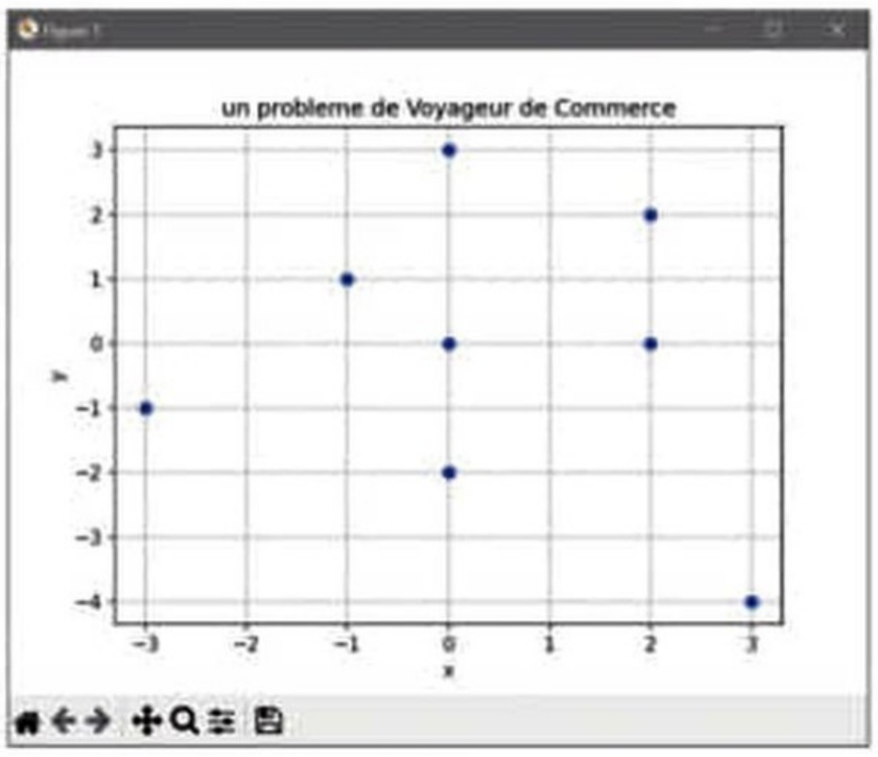


Fig 1. *Des points et des coordonnées*

```python
from geopy.geocoders import Nominatim
from geopy import distance

def get_coordinates(city):
    geolocator = Nominatim(user_agent="myapp")
    location = geolocator.geocode(city)
    return location.latitude, location.longitude

def calculate_distance(city1, city2):
    coords1 = get_coordinates(city1)
    coords2 = get_coordinates(city2)
    return distance.distance(coords1, coords2).kilometers

# Les deux villes
depart = "Paris, France"
arrivee = "Clermont-Ferrand, France"

# La distance
d = calculate_distance(depart, arrivee)
print("La distance entre "+ depart+" / "+arrivee+" est de "+str(d)+" km")
```

Fig 2. *Des vraies distances avec geopy*

Gérard Fleury

est Maître de Conférences associé au LIMOS. Il est spécialiste de statistique et d'optimisation. Il est co-auteur de deux livres sur le quantique parus en 2022 et 2023.

Email : gerard.fleury@isima.fr

Philippe Lacomme

est Professeur à l'ISIMA où il enseigne l'optimisation et l'informatique quantique.Email : philippe.lacomme@isima.fr

Caroline Prodhon

est Professeur à l'Université de Technologie de Troyes. Elle est co-responsable du Groupe de Travail GT2L.Email : caroline.prodhon@utt.fr

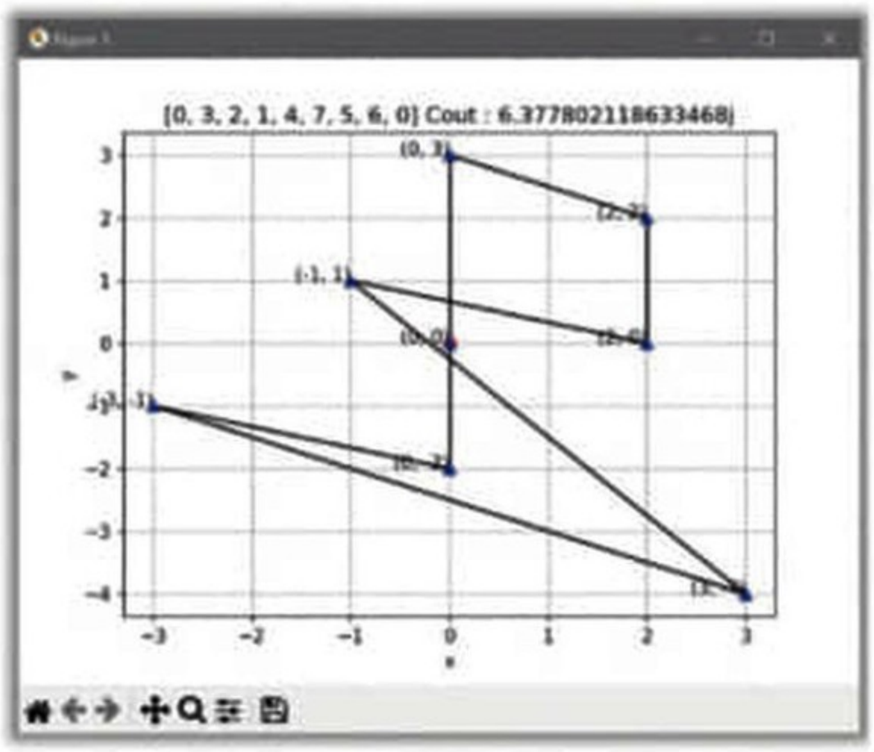


Fig 3.
La solution 0, 3, 2, 1, 4, 7, 5, 6, 0

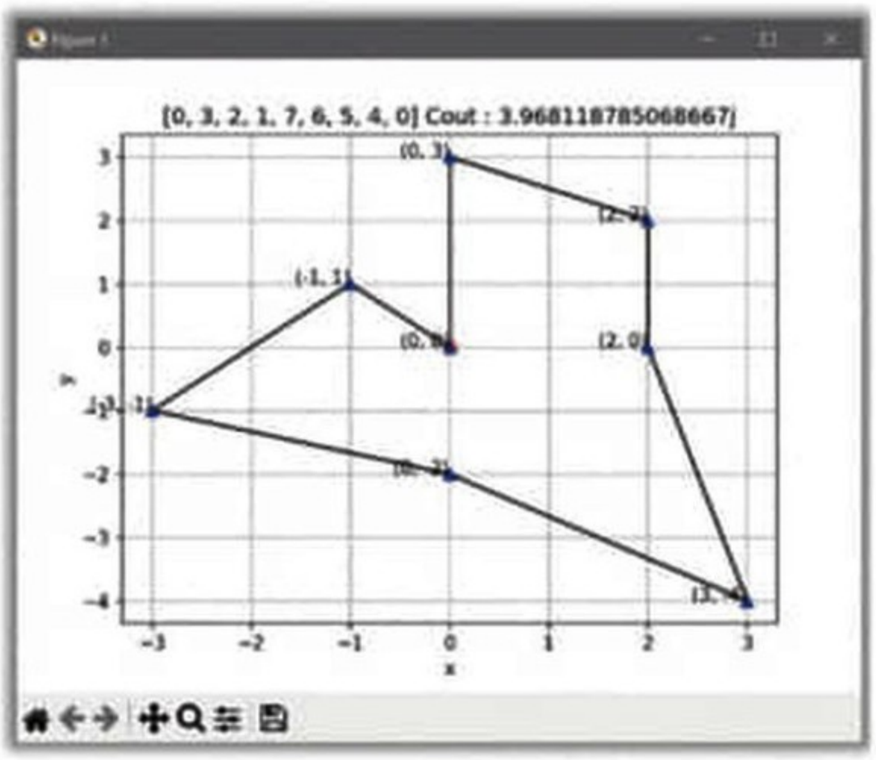


Fig 4.
La solution 0, 3, 2, 1, 7, 6, 5, 4, 0

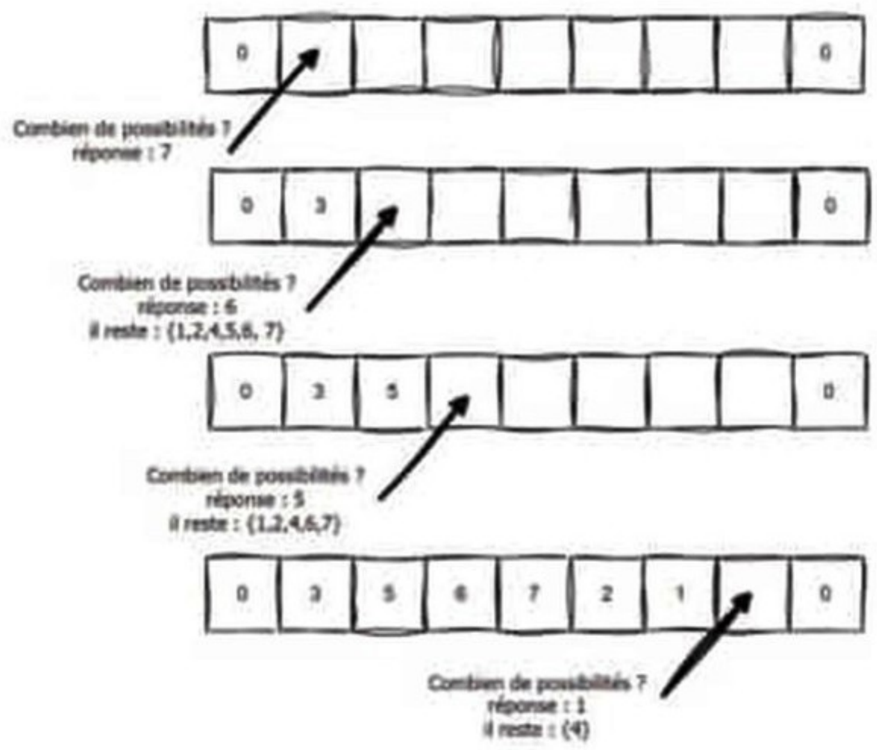


Fig 5.
Le nombre de solutions

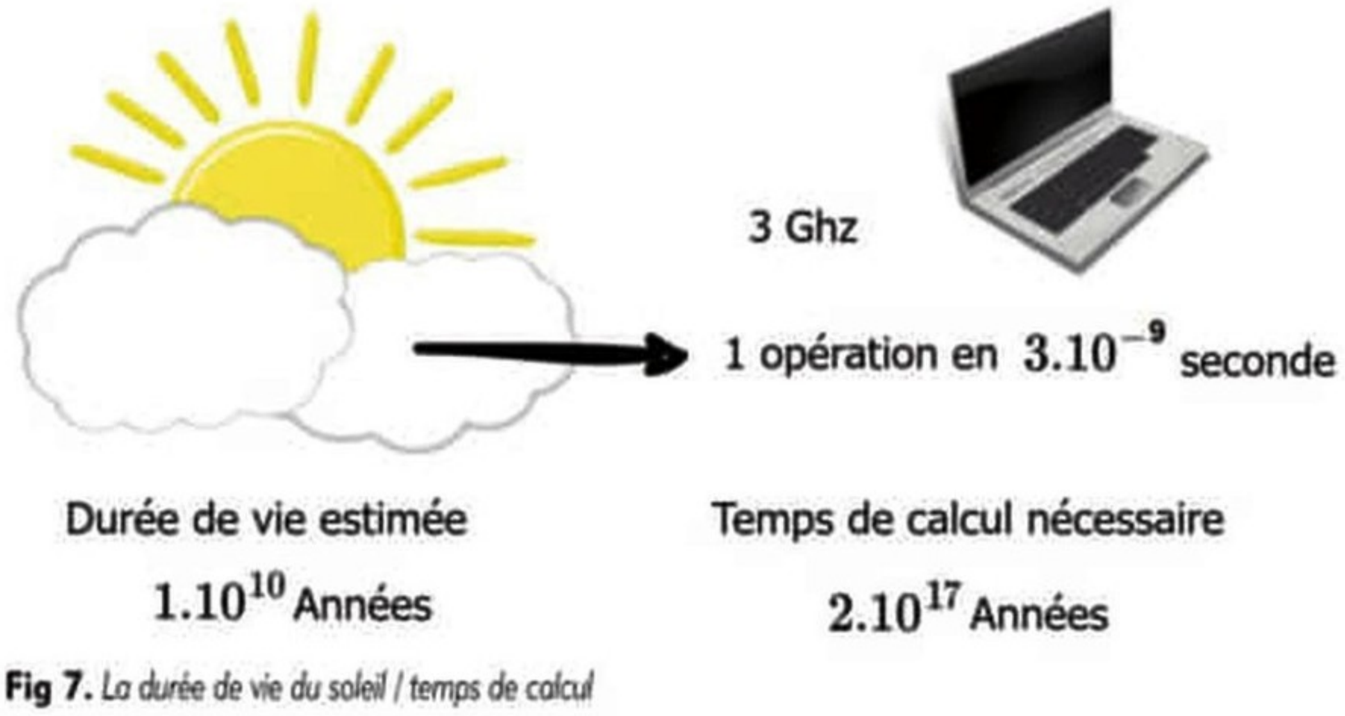


Fig 7. *La durée de vie du soleil / temps de calcul*

Une_Solution=[0, 3, 2, 1, 4, 7 ,5, 6, 0]

```
def cout_solution (liste, distance):
    cout = 0
    for i in range (0, w-1):
      cout = cout + distance[liste[i]][liste[i+1]]
    return cout

Cout = cout_solution (Une_Solution, distance)
print(Cout)
```

Quel est son coût ? Simplement la somme des distances.
Les **figures 3 et 4** montrent que des solutions différentes correspondent bien sûr à des ordres de visites différents.

Combien de solutions sont possibles ?

À mesure que le nombre de villes augmente, le nombre de circuits possibles explose puisque, pour n villes, le nombre de circuits est de l'ordre de n! Si l'on reprend le problème précédant à 7 clients, la première case du tableau **Solution** contient obligatoirement 0.
La deuxième case du tableau peut contenir n'importe quel élément parmi les 7 restants, la troisième case n'importe quel élément parmi les 6 restants et ainsi de suite (voir **figure 5**).
C'est la fameuse croissance exponentielle du nombre de solutions en fonction du nombre de clients, comme on peut le voir sur la **figure 6**. Pour 30 villes le nombre de solutions est 30! ce qui représente plus de 10^31 potentialités !

Le soleil ne brille plus !

Comme nous l'avons montré dans les pages précédentes, pour évaluer une solution, il faut n tours de boucle et dans la boucle For on trouve 8 opérations élémentaires :

- une affectation
- la lecture de la variable **coût**
- la lecture de la donnée distance[liste[i]][liste[i+1]] qui à elle seule nécessite 6 accès mémoire.

Pour 30 clients, le résultat est sans appel : 2.10^{17} années
Remarquons que ce temps de calcul est $2.10^{7} = 2\,000\,000$ fois plus grand que la durée de vie du soleil (estimée à 10^{10} années) comme illustré en **figure 7**.
Le soleil aura cessé de briller bien avant que notre PC ne termine de lister l'ensemble des solutions !

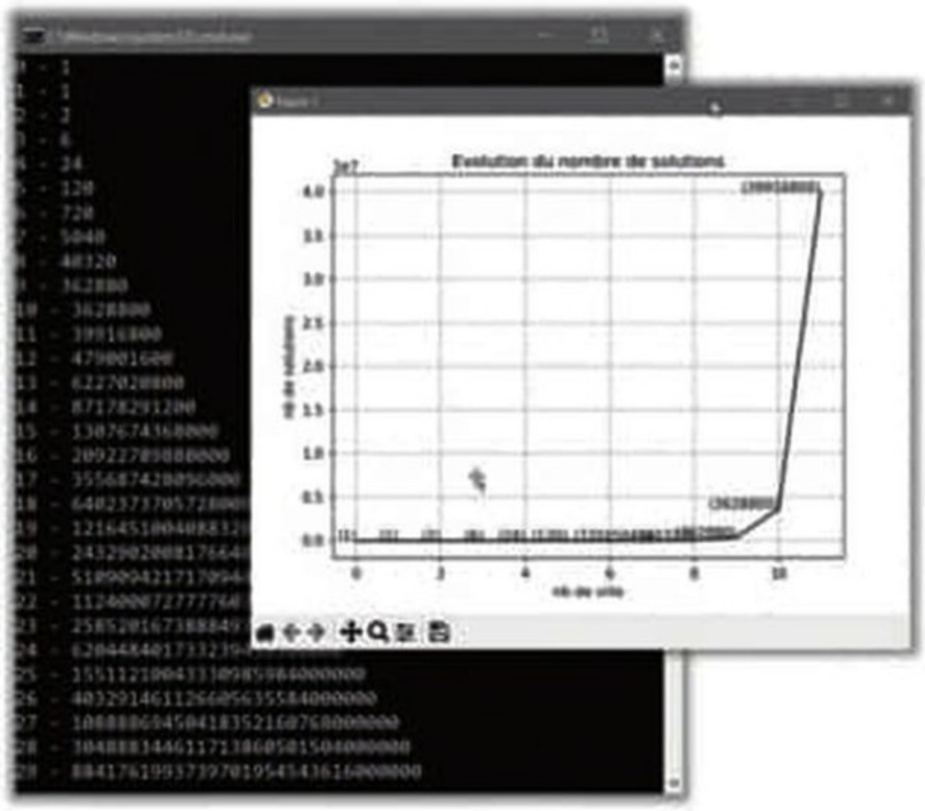


Fig 6. *Évolution du nombre de solutions*

Que promettent les machines quantiques ?

Une résolution efficace en temps de calcul.

De quoi avons-nous besoin pour tester ces machines ?

Il faut disposer d'un outil pour spécifier un circuit quantique qui sera l'équivalent quantique d'un programme informatique classique. Ce circuit est alors compilé (en informatique quantique on parle de Transpilation) pour être exécuté soit sur un simulateur soit sur une vraie machine (**Figure 8**).
Pour cela, de nombreuses solutions existent. IBM par exemple, propose un simulateur et plusieurs machines quantiques sur lesquelles il est possible d'exécuter des circuits gratuitement. L'accès à ces machines peut se faire soit via un code Python et l'utilisation de la librairie Qiskit ou directement à la souris via un navigateur Internet.
https://quantum-computing.ibm.com/

EVIDEN/ATOS a développé une solution packagée dans la QLM qui est un simulateur performant (https://myqlm.github.io/)
On peut aussi citer :

- La solution de Microsoft en Q# : https://azure.microsoft.com/fr-fr/resources/development-kit/quantum-computing/
- La solution Google Cirq : https://quantumai.google/cirq
- La solution Riggetti : https://qcs.rigetti.com/sdk-downloads

2 Exécuter son premier programme

Revenons à notre problème consistant à visiter un ensemble de clients en partant d'un dépôt. Comme nous l'avons vu, la liste des solutions possibles est exponentielle. Cependant, l'informatique quantique offre la possibilité de « créer » cette liste « instantanément ». Il faut alors trouver dans cette liste, la séquence de coût minimum. Et bonne nouvelle ! Il existe un algorithme quantique pour nous y aider…
Un autre exemple très répandu en informatique est la recherche d'un élément dans une base de données ou dans un tableau. On est dans un cas très proche du cas précédent. La valeur recherchée peut être dans n'importe qu'elle case du tableau et si le tableau comporte 1 000 000 de cases, potentiellement il faut parcourir toutes les cases pour trouver la valeur recherchée. La "magie quantique" à nouveau nous aide.

Trouver une donnée sans parcourir toutes les cases d'un tableau

Soit un tableau composé de 4 cases numérotées de 0 à 3 et contenant les valeurs suivantes :
T[0]=0, T[1]=7, T[2]=3 et T[3] = 5

Avec une représentation binaire des valeurs, ceci donne :
T[00]=000, T[01]=111
T[10]=011 et T[11] = 100

L'objectif est de créer un algorithme quantique permettant de retrouver la position d'une valeur dans le tableau. Connaissant *e* , il s'agit de trouver *i* tel que *T[i]* = e.
Mettre les éléments dans le tableau et chercher où se trouve la valeur 3 correspond au circuit de la **figure 9**. Ce circuit correspond à une seule itération. Le circuit permet de trouver la valeur 3 et donne une solution sous la forme d'une distribution de probabilité.
Cette distribution montre que la valeur 3 se trouve dans la case 10 avec une probabilité de 100% (**figure 10**).
Vous pouvez tester ce circuit sur le site de Qiskit (solution IBM).
En utilisant ce type de circuit pour notre problème, il est alors possible d'extraire de la liste des solutions possibles, celle qui nous intéresse pour minimiser la distance parcourue par notre commercial.

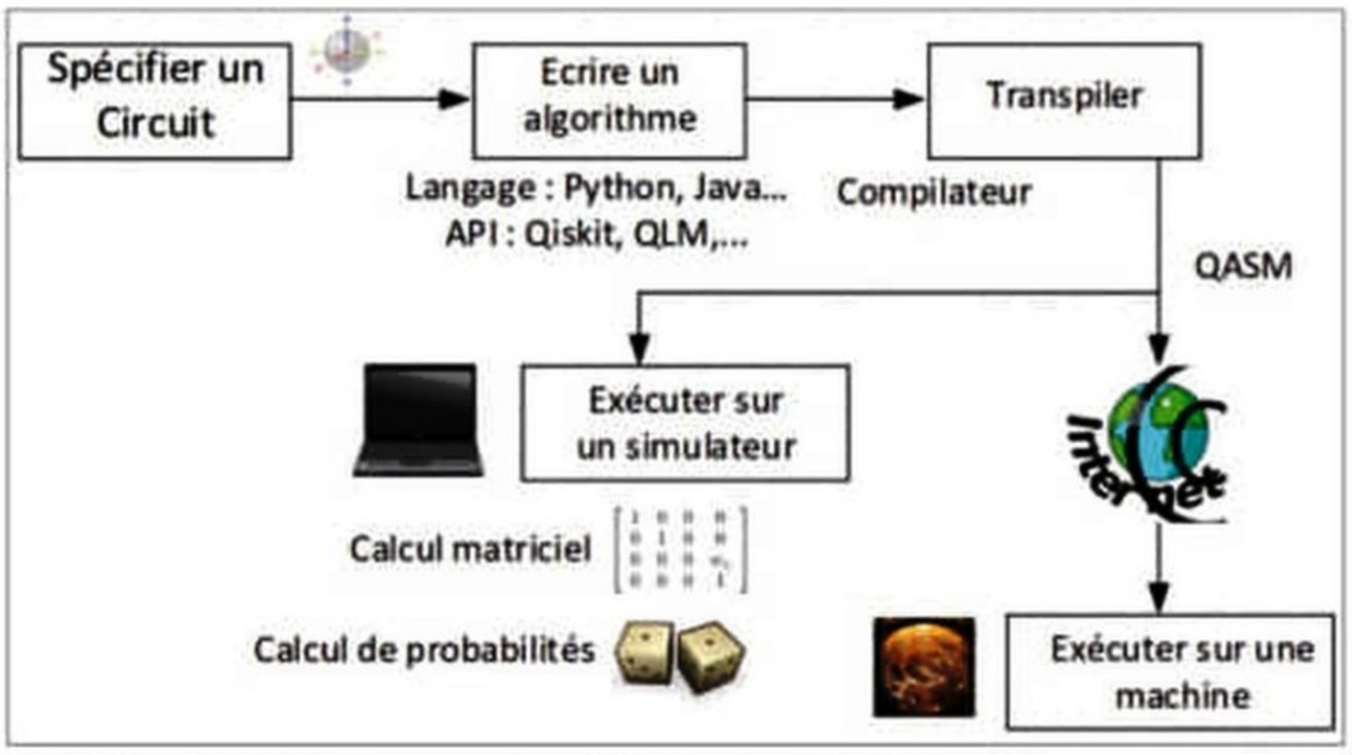


Fig 8. *Principes d'écriture d'un programme quantique (Bourreau et al., 2022)*

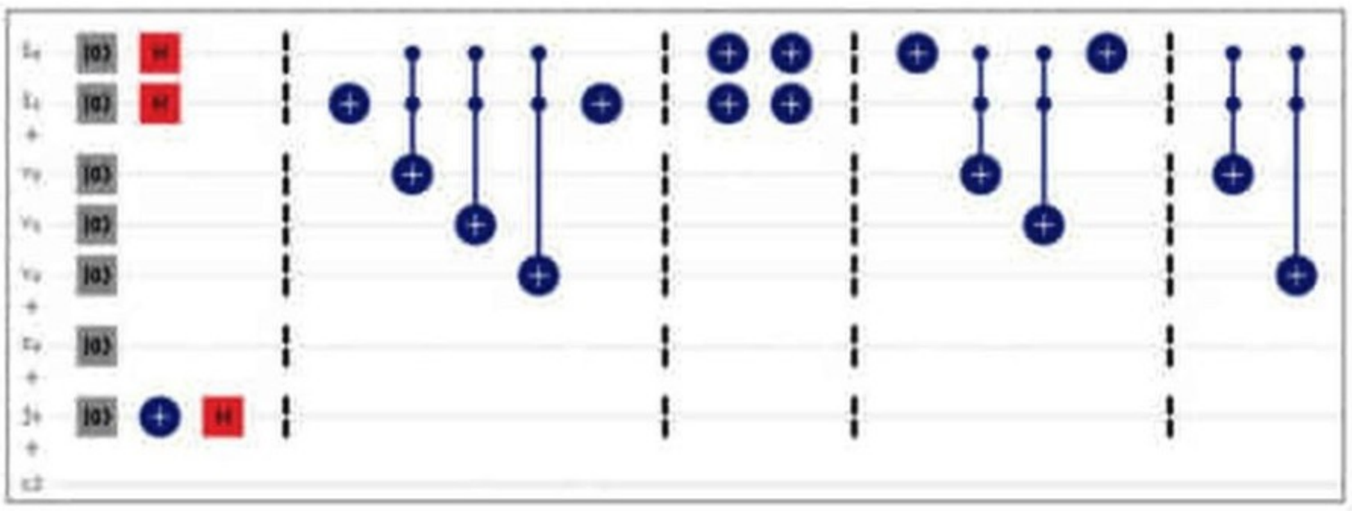
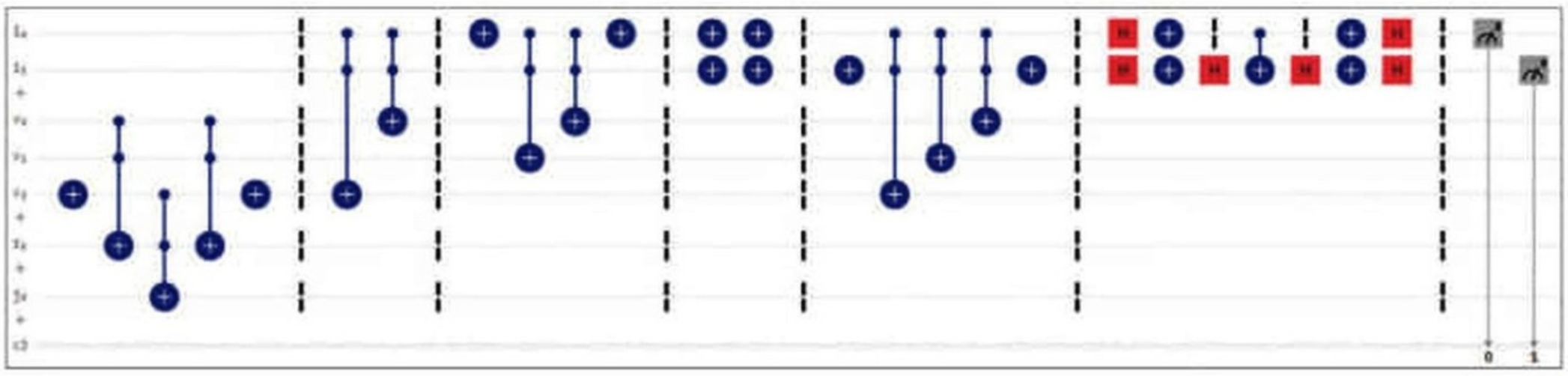


Fig. 9. *Un circuit quantique pour chercher un élément dans un tableau.*



3 Comment se former ?

Le plan quantique national

Les aides et soutiens financiers au quantique sont nombreux et variés et prennent forme par des programmes nationaux et par la création d'une plateforme nationale de calcul. Au niveau international, la France a signé en septembre 2021 un accord de collaboration avec les Pays-Bas au vu des complémentarités fortes entre les deux écosystèmes. L'objectif est de créer des synergies européennes, qui commencent au niveau scientifique, mais peuvent se poursuivre au niveau industriel. Un exemple récent est la fusion de Pasqal et Qu&Co.

Par ailleurs, une coopération bilatérale a été décidée avec les États-Unis en décembre 2022. Elle doit porter sur les applications du calcul quantique, la mutualisation des infrastructures, les réseaux de communication, l'hybridation du calcul classique et quantique et le développement de standards.

Et l'enseignement du quantique ?

Les unités d'enseignement ne sont pas oubliées dans les plans gouvernementaux puisque le gouvernement recommande que les formations en master en informatique proposent 6 ECTS sur l'informatique quantique.

On devrait donc voir apparaître dans les années à venir des cours dans les écoles et les formations universitaires. Actuellement il existe quelques formations scientifiques qui forment au quantique :

L'ISIMA de Clermont-Ferrand qui propose un cours de 20h aux étudiants de 3e année (Bac+5). Ce cours est réalisé par Philippe Lacomme

L'université de Technologie de Troyes qui a ouvert un cours de 6 crédits d'introduction à l'information quantique et aux technologies associées pour ses élèves ingénieurs.

4 Conclusion

Un peu de mathématiques, un peu de physique et un peu d'informatique : le quantique est à la croisée des chemins.

La lecture de cet article peut être complétée en écoutant les cours proposés sur la chaîne YouTube du GT2L. https://www.youtube.com/@lachainegt21248/videos

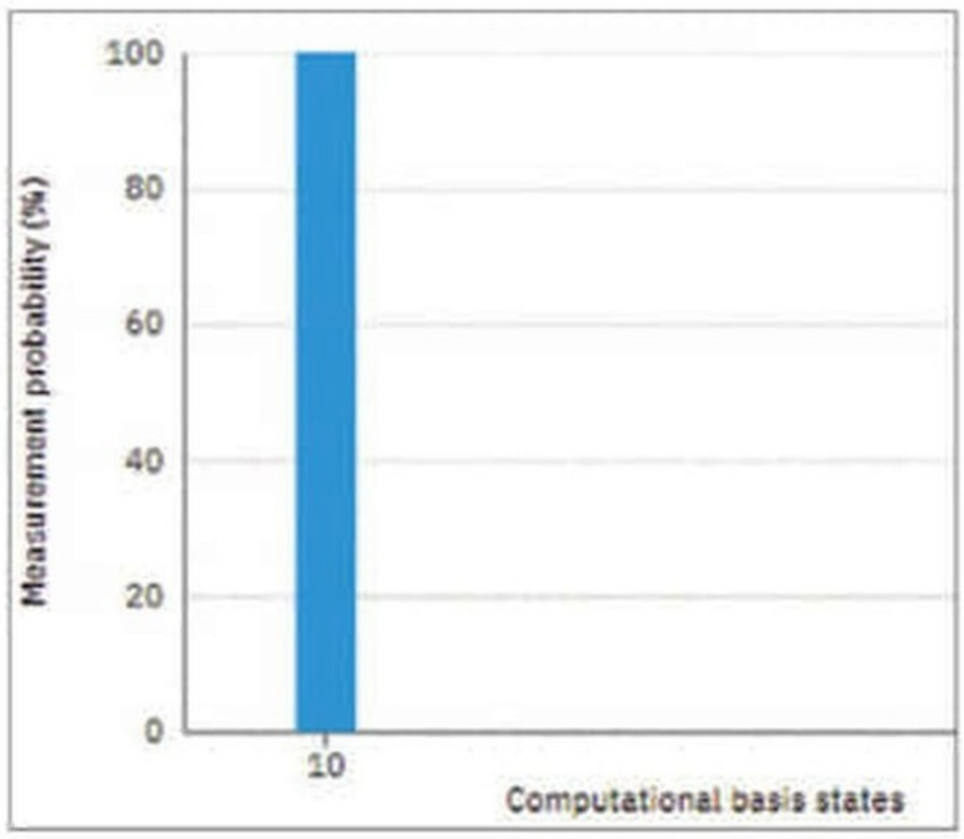
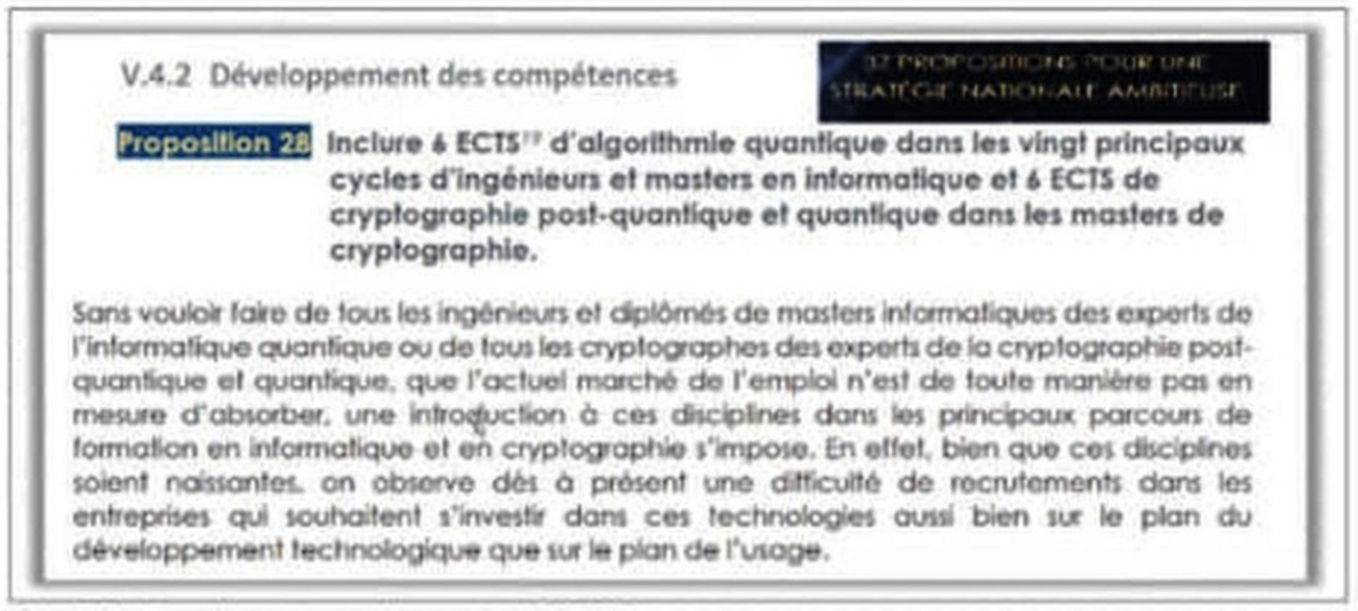


Fig. 10. *Le résultat d'un circuit quantique.*


V.4.2 Développement des compétences

32 PROPOSITIONS POUR UNE STRATÉGIE NATIONALE AMBITIEUSE

Proposition 28 **Inclure 6 ECTS[17] d'algorithmie quantique dans les vingt principaux cycles d'ingénieurs et masters en informatique et 6 ECTS de cryptographie post-quantique et quantique dans les masters de cryptographie.**

Sans vouloir faire de tous les ingénieurs et diplômés de masters informatiques des experts de l'informatique quantique ou de tous les cryptographes des experts de la cryptographie post-quantique et quantique, que l'actuel marché de l'emploi n'est de toute manière pas en mesure d'absorber, une introduction à ces disciplines dans les principaux parcours de formation en informatique et en cryptographie s'impose. En effet, bien que ces disciplines soient naissantes, on observe dès à présent une difficulté de recrutements dans les entreprises qui souhaitent s'investir dans ces technologies aussi bien sur le plan du développement technologique que sur le plan de l'usage.

Fig. 11. *Extrait du rapport parlementaire de 2019*

Références

Eric Bourreau, Gérard Fleury, Philippe Lacomme

Introduction à l'informatique quantique

Eyrolles - Avril 2022 -
ISBN10 : 2416006533

Réseaux de Tenseurs : du quantique à l'inspiration quantique

Alors que les perspectives de l'informatique quantique sont extrêmement prometteuses, la mise en œuvre d'ordinateurs quantiques non bruités reste un défi majeur. Cette barrière technologique a incité une partie de la communauté scientifique à se tourner vers des techniques dites d'"inspiration quantique" (ou "quantum-inspired"). Le grand avantage de ces approches est qu'elles ont le potentiel de transformer radicalement certains paradigmes de programmation sans nécessiter de machines quantiques pour fonctionner. Dans le cadre de l'évolution vers des techniques d'inspiration quantique, les réseaux de tenseurs émergent comme des structures mathématiques sophistiquées et puissantes, ayant leurs origines dans les défis de la physique et de la chimie quantique.

Michel KUREK

Directeur général de Multiverse Computing, start-up européenne leader dans le développement d'applications logicielles quantiques et inspirées par le quantique. M. Kurek a, par le passé, exercé des activités de trading et a été responsable d'équipe en charge du trading algorithmique au sein respectivement des groupes Crédit Agricole et Société Générale. M. Kurek est diplômé d'ENSIMAG, ESSEC, École polytechnique.

Dans cet article, nous introduirons les concepts, les notations et les applications de cette technologie "quantum-inspired". Nous explorerons comment la physique quantique et l'informatique se combinent, pour ne pas dire s'intriquent, pour offrir de nouvelles approches pour résoudre des problèmes industriels et opérationnels.

Un peu d'histoire

Pour mieux comprendre le rôle central des réseaux de tenseurs dans les techniques d'inspiration quantique, il est instructif de remonter à leurs origines au milieu des années 90. À l'époque, comme aujourd'hui, les chercheurs en physique et chimie quantique étaient confrontés à des défis computationnels majeurs. Si les ordinateurs se sont considérablement améliorés, les systèmes étudiés sont également devenus plus complexes et plus grands. Par conséquent, la simulation de systèmes quantiques comme des molécules ou des matériaux sophistiqués nécessite toujours des ressources computationnelles considérables, maintenant ainsi un haut niveau d'exigence dans ce domaine. C'est dans ce contexte que les tenseurs, généralisations mathématiques des scalaires, vecteurs et matrices, sont apparus et se sont imposés comme une solution compacte et efficace pour manipuler les structures de données multidimensionnelles complexes. En exploitant cette efficacité, un réseau de tenseurs est une architecture sophistiquée qui agence plusieurs de ces tenseurs en une configuration dédiée, facilitant ainsi des opérations et des calculs en haute dimension.

L'attrait des réseaux de tenseurs repose sur leur grande polyvalence. Initialement imaginés pour des applications en physique, ils ont montré leur utilité dans une multitude d'autres domaines. De la compression de données à l'apprentissage automatique, en passant par l'analyse de graphes et même l'essor de l'informatique quantique, les réseaux de tenseurs démontrent à chaque étape leur valeur intrinsèque.

Notations et représentations diagrammatiques : des outils pour appréhender les complexités des réseaux de tenseurs

Le formalisme des réseaux de tenseurs s'appuie sur une convention de notation et des représentations diagrammatiques spécifiques, les rendant particulièrement ingénieux pour aborder intuitivement les opérations tensorielles complexes que nous discuterons par la suite.

Notation d'Einstein : simplifier l'écriture des opérations tensorielles

La première convention clé à comprendre est la notation d'Einstein, également appelée « notation d'indice ». Introduite par Albert Einstein en 1916, elle trouve son application principale en physique théorique et en mathématiques appliquées pour simplifier significativement l'écriture des équations impliquant des tenseurs ou des vecteurs (équation tensorielle). Dans cette notation, le symbole de sommation ($\sum$) est souvent omis, la règle étant que chaque indice apparaissant deux fois dans un terme est automatiquement sommé sur son ensemble de définitions. Par exemple, la multiplication de deux matrices $\boldsymbol{A}$ et $\boldsymbol{B}$ peut être représentée ainsi : $A_{ij}B_{jk} \equiv C_{ik}$

Ce qui équivaut à : $C_{ik} = \sum_{j=1}^{n} A_{ij}\, B_{jk}$

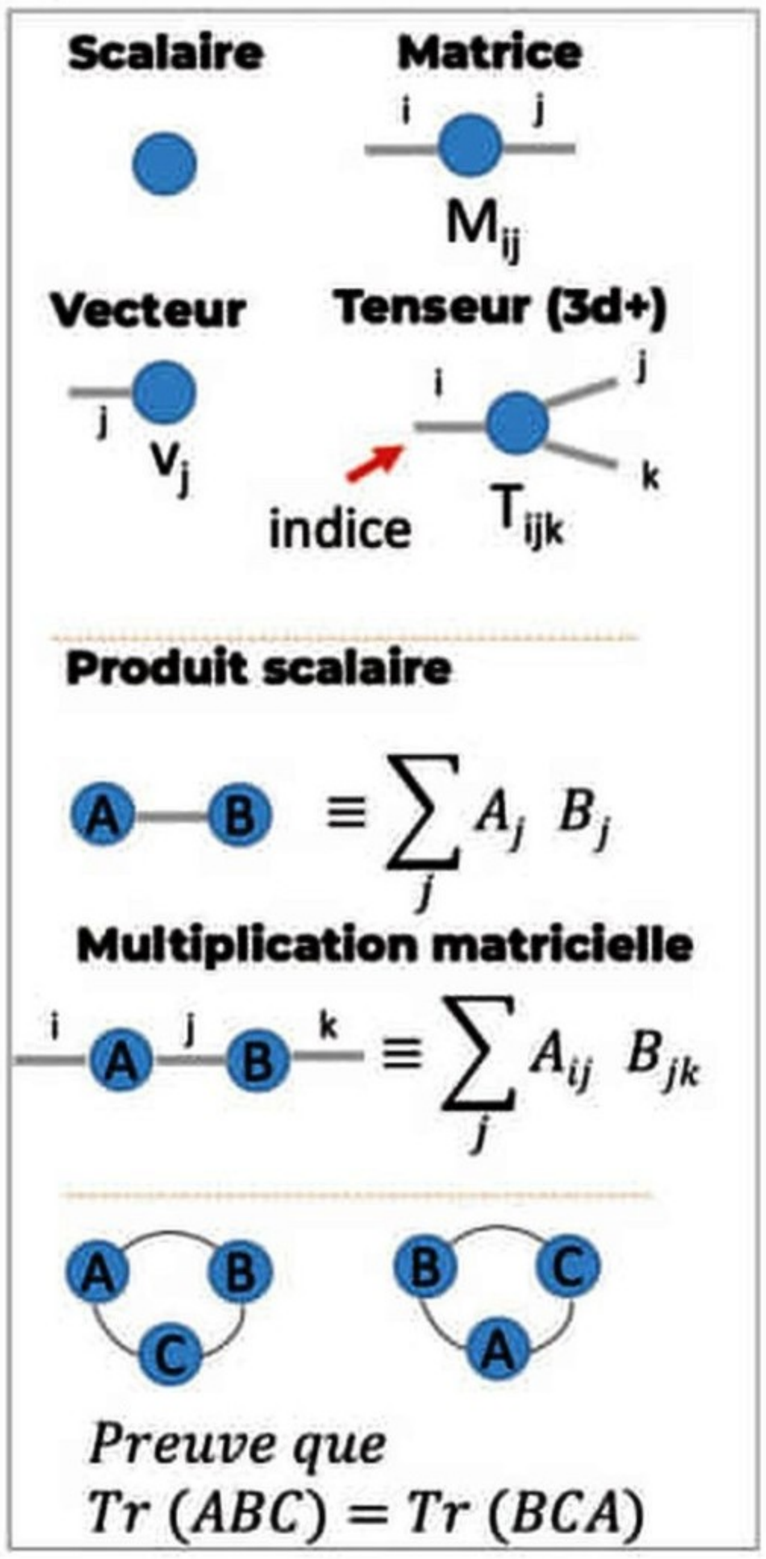
En éliminant les symboles de sommation en supprimant les indices répétés, elle rend les équations plus lisibles et plus faciles à manipuler.

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p}\sum_{l=1}^{q} A_{ij}B_{jk}C_{kl}D_{li} \equiv A_{ij}B_{jk}C_{kl}D_{li}$$

Diagrammes de Réseaux de Tenseurs : Une Image Vaut Mille Mots

Le second outil que nous abordons est la représentation diagrammatique des réseaux de tenseurs. Cette représentation (**Figure 1**) est particulièrement utile pour comprendre intuitivement des opérations complexes. Dans ces diagrammes, chaque nœud représente un tenseur, et les arêtes représentent les indices tensoriels (ie indice d'accès aux données – le nombre d'indices donnant le rang du tenseur). A l'aide de ce formalisme, il devient facile de montrer l'identité mathématique Tr (ABC) = Tr (BCA) - où la trace Tr d'une matrice car-

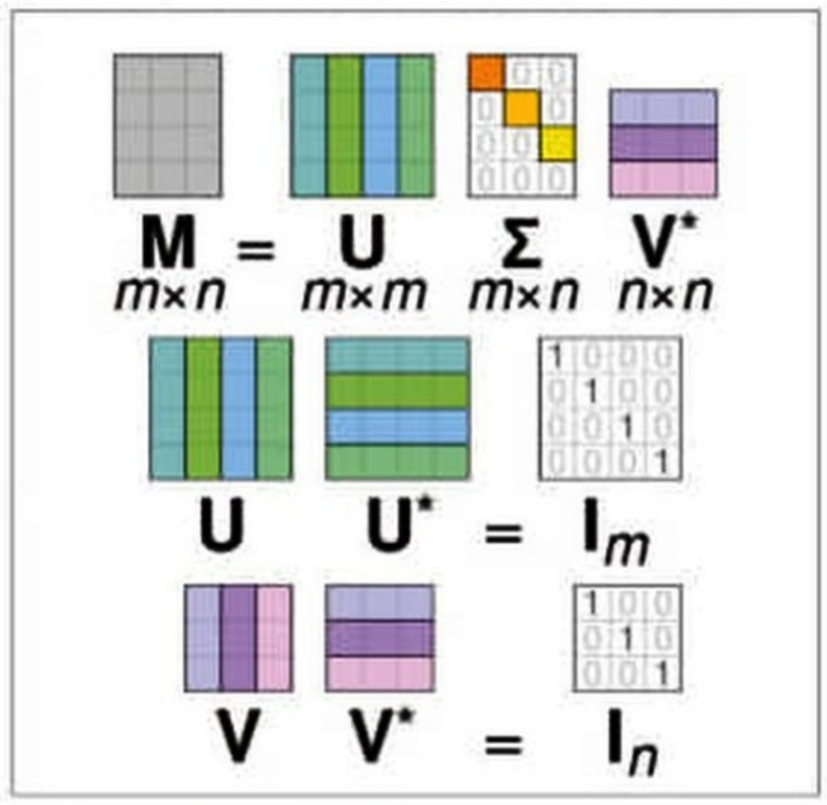
Figure 1



rée est la somme des éléments diagonaux.

L'opération mathématique qui combine deux tenseurs en sommant sur un ou plusieurs de leurs indices s'appelle la contraction (par exemple le produit scalaire ou la multiplication de matrice). Dans un réseau de tenseurs, cette opération peut être beaucoup plus complexe, impliquant de multiples tenseurs avec des dimensions plus élevées.

La décomposition de tenseurs, quant à elle, est le processus de décomposer un tenseur en un ensemble de tenseurs de rang inférieur qui, lorsqu'ils sont contractés, approximent le tenseur original. La technique de décomposition en Valeurs Singulières (SVD), courante en algèbre linéaire est souvent utilisée (**illustration source : Wikipédia**). La décomposition peut réduire considérablement la complexité computationnelle lors de la manipulation de grands tenseurs, et elle est souvent une étape cruciale dans l'optimisation des calculs dans les réseaux de tenseur.



Architectures de base

A ce stade nous sommes à présent équipés pour comprendre les architectures de base qui forment le socle des réseaux de tenseurs **(Figure 2)**.

Matrice Produit d'États (MPS) et Tensor Train (TT)

Les MPS (Matrix Product States) et les TT (Tensor Trains) sont des décompositions de tenseurs de haute dimension en séries ou "chaînes" de tenseurs de dimension plus basse. Le MPS est souvent employé dans le domaine de la physique quantique, en particulier pour traiter des systèmes unidimensionnels, tandis que le terme Tensor Train est plus couramment utilisé en mathématiques appliquées et en informatique pour des applications variées.

Réseaux de PEPS (Projected Entangled Pair States)

Les PEPS sont une généralisation des MPS à des grilles bidimensionnelles ou plus, souvent utilisées pour des systèmes quantiques en deux dimensions.

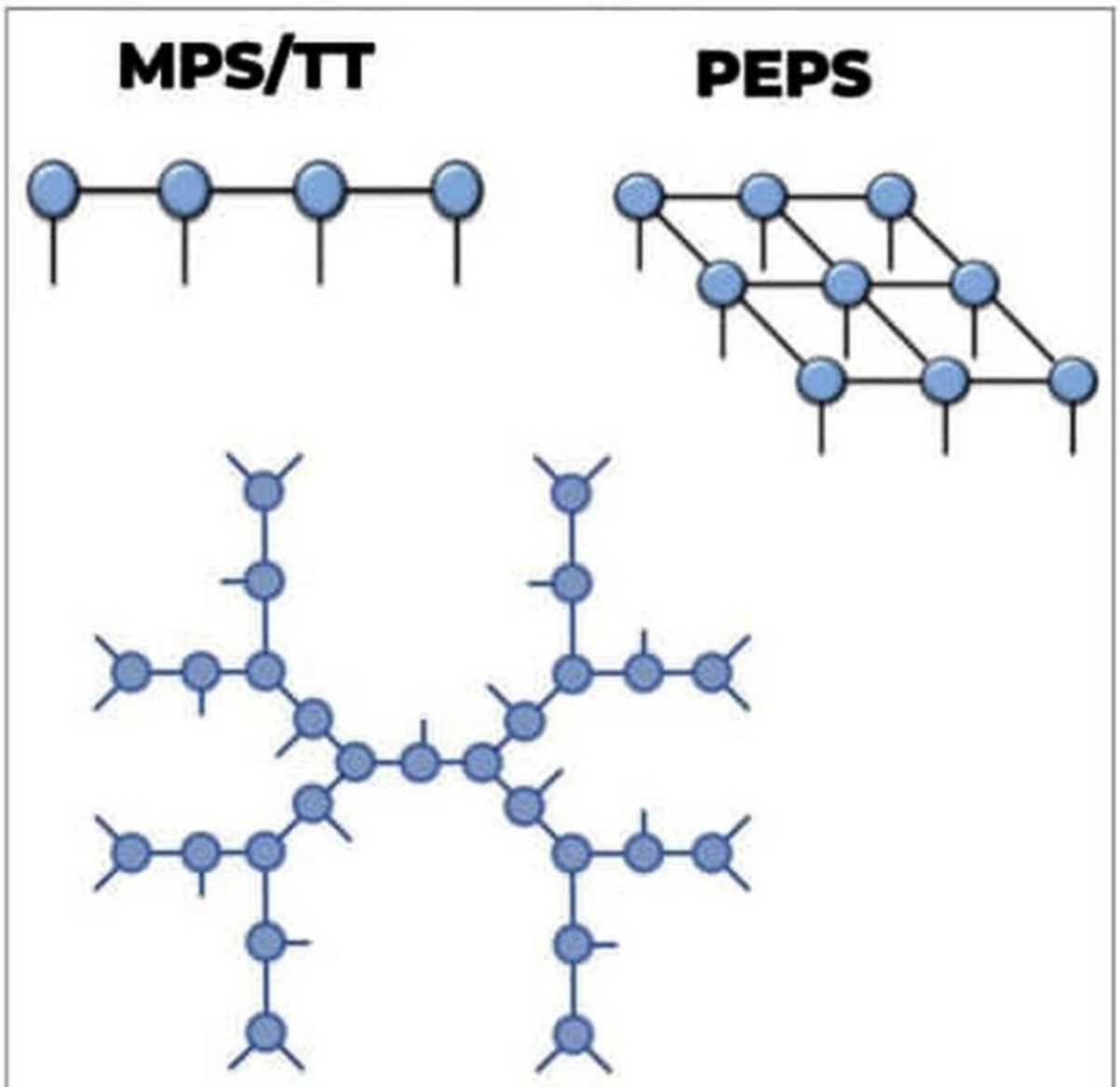
Multiscale Entanglement Renormalization Ansatz (MERA)

Les MERA sont une autre classe de réseaux de tenseurs conçue pour gérer efficacement les états quantiques à plusieurs échelles. Ils utilisent une structure hiérarchique qui ressemble à un arbre et sont particulièrement utiles pour étudier les systèmes critiques, où les corrélations entre les éléments sont à longues portées.

Structures d'Arbres et Graphes plus généraux

Certains réseaux de tenseurs, y compris les MERA, sont organisés sous forme d'arbre (TTN – Tree Tensor Network) pour modéliser des hiérarchies et des relations complexes entre différentes dimensions du tenseur. Toutefois, il faut noter qu'il existe également des structures de graphes de tenseurs plus généraux ou spécifiques, adaptées aux problèmes que l'on cherche à résoudre.

Chaque architecture a ses propres avantages et inconvénients en termes de complexité computationnelle, d'efficacité de stockage, ainsi que des domaines d'application spéci-



Exemple de réseau de tenseurs arborescent

Figure 2

fiques, rendant le domaine des réseaux de tenseurs très riche et polyvalent.

Au-delà des structures : Les opérateurs et algorithmes clés des réseaux de Tenseurs

Après avoir examiné les architectures de base des réseaux de tenseurs comme les MPS, PEPS, et TTN, il convient de se pencher sur les opérateurs et les algorithmes qui exploitent ces structures pour résoudre des problèmes complexes. À ce titre, trois éléments clés méritent d'être abordés : la représentation Matrix Product Operator (MPO), et les algorithmes Density Matrix Renormalization Group (DMRG) et Time-Evolving Block Decimation (TEBD). Ces techniques trouvent des applications non seulement en physique et en chimie quantique, mais également dans des domaines plus larges tels qu'en informatique pour résoudre des problèmes d'optimisation par exemple.

Matrix Product Operator (MPO)

Le MPO n'est pas un algorithme, mais plutôt une représentation qui étend le concept de MPS aux opérateurs. Dans le contexte de la physique quantique, les MPO sont couramment utilisés pour modéliser des éléments tels que les Hamiltoniens. En informatique, particulièrement dans des problèmes d'optimisation, les MPO peuvent également être utilisés pour formaliser des fonctions objectives complexes.

Density Matrix Renormalization Group (DMRG)

Le DMRG est un algorithme qui utilise les MPS pour optimiser la recherche d'états propres, notamment les états de plus grande importance dans un système quantique. Bien que développé initialement pour des systèmes quantiques à faible dimension, le DMRG trouve principalement ses applications en optimisation.

Time-Evolving Block Decimation (TEBD)

Le TEBD est un algorithme destiné à la simulation de l'évolution temporelle de systèmes quantiques. Il utilise une approximation MPS pour traiter efficacement des systèmes de grande taille. Par sa capacité à gérer la dynamique des systèmes quantiques, le TEBD complète les capacités des algorithmes comme le DMRG dans la palette des techniques basées sur les réseaux de tenseurs. Bien que son utilisation en informatique traditionnelle soit encore limitée, il existe un intérêt croissant pour la recherche sur son application potentielle dans des domaines comme l'optimisation combinatoire et l'apprentissage automatique.

Ainsi, les réseaux de tenseurs ne sont pas uniquement des structures de données. Ils servent également de base à des algorithmes et des représentations qui permettent des calculs avancés dans divers domaines. Les algorithmes mentionnés ne représentent que la pointe de l'iceberg dans le domaine en plein essor des réseaux de tenseurs. Avec une variété d'autres techniques algorithmiques émergentes, ces méthodes permettent des applications allant de la modélisation en physique quantique à des défis en apprentissage automatique et en traitement de données. La synergie entre ces algorithmes et les diverses architectures de réseaux de tenseurs constitue un domaine de recherche dynamique. Ensemble, ils offrent une flexibilité et une efficacité computationnelle qui continuent à élargir le champ d'application de cette technologie 'quantum-inspired'.

Outils et Bibliothèques pour l'Implémentation de Réseaux de Tenseur

Bien que Numpy ne soit pas une bibliothèque Python spécifiquement dédiée aux réseaux de tenseurs, elle offre toutes les fonctionnalités basiques nécessaires à leur création et manipulation. Le code ci-dessous fournit un exemple concret : il illustre comment compresser un tenseur à haute dimension grâce à la décomposition en valeurs singulières (SVD), suivi de la troncature des valeurs propres les moins significatives. Le code permet ensuite d'accéder à un élément spécifique du tenseur compressé.

```
import numpy as np

def tt_svd(tensor: np.ndarray, truncerr: float = 10**-6, maxdim = 10**
12) -> list:
  """
  Compress a tensor to a MPS/TT using the TT-SVD algorithm.

  Args:
    tensor: The input tensor
    truncerr: Truncation error for each SVD
    maxdim: Maximum MPS bond dimension
  Return:
    An MPS/TT as a list of order-3 tensors (dummy bonds are added to boundary
tensors)
  """
  dimensions = tensor.shape
  order = len(dimensions)
  mps = []
  virtual_dimension = 1
  for i in range(order-1):
    # Reshape to matrix
    tensor = tensor.reshape(virtual_dimension * dimensions[i], -1)

    # SVD + Truncate
    U, s, Vt = np.linalg.svd(tensor)
    # Truncate singular values such that truncation error is less than or equal
to truncerr
    where_lower_than_truncerr = np.where(np.cumsum(s[::-1]**2)
<= truncerr**2)[0]
    number_of_singular_values_to_discard = 0 if len(where_lower_
than_truncerr) == 0 else int(1 + where_lower_than_truncerr[-1])
    new_virtual_dimension = min(maxdim, max(1, len(s) - number_
of_singular_values_to_discard))

    # Reshape and truncate U matrix, store in return list
    mps.append(U[:,:new_virtual_dimension].reshape(virtual_dimension,
dimensions[i], new_virtual_dimension))

    # Contract s and Vt
    tensor = np.diagflat(s[:new_virtual_dimension]) @ Vt[:new_virtual
_dimension,:]
```

```
    virtual_dimension = new_virtual_dimension
  mps.append(tensor.reshape(virtual_dimension, dimensions[-1], 1))
  return mps# Create a random tensor
tensor = np.random.rand(2,2,2,2,2)# Compress to MPS/TT
mps = tt_svd(tensor, truncerr=10**-6, maxdim=10)from functools
 import reduce
def get_index(mps: list, index: list):
  """
  Retrieve a single element of a tensor represented by an MPS/TT by
performing the matrix product

  Args:
    mps: The MPS/TT
    index: The index of the element to retrieve
  """

  return reduce(np.matmul, (site[:, ind, :] for site, ind in zip(mps,
index)))[0,0]# Retrieve an element
get_index(mps, [0, 0, 1, 0, 0])0.2956102700141592
```

EXPLICATION DU CODE

Dans ce code, nous utilisons la bibliothèque Numpy pour implémenter l'algorithme TT-SVD, qui permet de compresser un tenseur à haute dimension en une structure Matrix Product State (MPS) ou Tensor Train (TT).

Fonction `tt_svd`

```
import numpy as np
def tt_svd(tensor : np.ndarray, truncerr : float = 10**-6, maxdim = 10**
12) -> list:
```

Cette fonction prend en entrée un tenseur `tensor`, une erreur de troncature `truncerr`, et une dimension maximale `maxdim` de lien MPS. Elle retourne une liste de tenseurs d'ordre 3 qui représente le tenseur compressé en format MPS ou TT.

Arguments :

- `**tensor**` : Le tenseur à compresser.
- `**truncerr**` : Erreur de troncature pour chaque décomposition SVD.
- `**maxdim**` : Dimension de lien MPS maximale.

Retour :

- Une liste de tenseurs d'ordre 3 qui forme le MPS/TT.

Fonctionnement

1. **Reshape du tenseur** : Le tenseur est remodelé en une matrice à deux dimensions.

```
tensor = tensor.reshape(virtual_dimension * dimensions[i], -1)
```

2. **SVD et Troncature** : La décomposition en valeurs singulières (SVD) est effectuée sur cette matrice. Ensuite, les valeurs singulières sont tronquées en fonction de `truncerr`.

```
U, s, Vt = np.linalg.svd(tensor)
```

3. **Construction du MPS** : La matrice U est remodelée et stockée dans la liste MPS.

```
mps.append(U[:,:new_virtual_dimension].reshape(virtual_dimension,
 dimensions[i], new_virtual_dimension))
```

Fonction `get_index`

```
def get_index(mps : list, index: list) :
```

Cette fonction récupère un élément unique d'un tenseur représenté par un MPS/TT en effectuant le produit matriciel.

Arguments :

- `**mps**` : Le MPS/TT.
- `**index**` : L'index de l'élément à récupérer.

Retour :

- L'élément du tenseur à l'indice donné.

Cela étant dit, il est important de noter qu'il existe des bibliothèques spécialisées en réseaux de tenseurs, offrant des implémentations plus concises et efficaces.

- **TensorNetwork**: bibliothèque open source pour des opérations tensorielles efficaces et pour des réseaux de tenseurs en Python.
- **ITensors** : conçu spécifiquement pour les réseaux de tenseurs, elle est particulièrement utilisée en physique statistique.
- **TeNpy** : est une bibliothèque Python spécialisée dans les calculs sur les réseaux de tenseurs. Elle est optimisée pour effectuer des simulations sur des systèmes quantiques unidimensionnels (MPS) et offre un ensemble riche d'algorithmes pour des méthodes comme la DMRG.
- **Uni10** : Une bibliothèque en C++ pour les réseaux de tenseurs.
- **OpenMPS** : Une bibliothèque en C++ qui se concentre spécifiquement sur les réseaux de tenseurs sous forme de MPS (Matrix Product States).
- **TT-Toolbox** : Bien que souvent utilisé pour des tenseurs en format "train", cette bibliothèque dispose également d'outils pour des réseaux de tenseurs plus généraux.

Par exemple, l'équivalent du code précédent dans Itensor est bien sûr plus concis.

```
using ITensors

# Create indices
i = Index(2, "i")
j = Index(2, "j")
k = Index(2, "k")
l = Index(2, "l")
m = Index(2, "m")

# Create random ITensor
tensor = randomITensor(i,j,k,l,m)ITensor ord=5 (dim=2|id=26|"i")
(dim=2|id=475|"j") (dim=2|id=291|"k") (dim=2|id=803|"l") (dim=2
|id=510|"m")
NDTensors.Dense{Float64, Vector{Float64}}# Compress tensor to MPS/TT
mps = MPS(tensor, (i, j, k, l, m); truncerr=10^-6, maxdim=10)MPS
[1] ((dim=2|id=26|"i"), (dim=2|id=365|"Link,n=1"))
[2] ((dim=2|id=365|"Link,n=1"), (dim=2|id=475|"j"), (dim=4|id=773
|"Link,n=2"))
[3] ((dim=4|id=773|"Link,n=2"), (dim=2|id=291|"k"), (dim=4|id=16
|"Link,n=3"))
[4] ((dim=4|id=16|"Link,n=3"), (dim=2|id=803|"l"), (dim=2|id=623
|"Link,n=4"))
[5] ((dim=2|id=623|"Link,n=4"), (dim=2|id=510|"m"))function get_
```

```
index(mps::MPS, index::Vector)
  ""
  Retrieve a single element of a tensor represented by an MPS/TT by performing the matrix product

  Args:
    mps: The MPS/TT
    index: The index of the element to retrieve
  ""
  V = ITensor(1.)
  for j=1:length(mps)
    V *= (mps[j] * state(siteind(mps, j), index[j]))
  end
  v = scalar(V)
end

# Retrieve an element
get_index(mps, [1, 1, 2, 1, 1])-1.012932614052549
```

Ces bibliothèques servent de points de départ pour le développement de nouveaux algorithmes ou pour l'application de ces algorithmes à la résolution de problèmes concrets. Elles offrent une variété de fonctions et de modules pré-existants qui facilitent grandement la mise en œuvre, le test et l'optimisation des méthodes mathématiques et informatiques. Ainsi, elles accélèrent non seulement le processus de R&D, mais permettent également une plus grande robustesse et fiabilité dans la résolution de problèmes complexes.

Applications diversifiées des réseaux de tenseurs

Machine Learning

Dans le domaine de l'apprentissage automatique, les techniques tensorielles sont de plus en plus utilisées pour la compression des données et la réduction de la dimensionnalité. À une époque où les grands modèles de langage, tels que ChatGPT, et les applications d'intelligence artificielle générative requièrent des ressources de calcul et d'énergie substantielles, l'importance de la compression des modèles est en croissance. Cette technique est particulièrement pertinente pour le déploiement de modèles sur des dispositifs à capacités limitées, comme les smartphones et les objets connectés (IdO/IoT).

En apprentissage automatique, la décomposition tensorielle est une méthode efficace pour la compression des modèles. Les poids des couches des réseaux neuronaux, souvent stockés sous forme de tenseurs de haute dimension, peuvent être approximés en utilisant des techniques comme la Décomposition par Valeurs Singulières (SVD) ou les Tensor Trains (TT). Ces méthodes permettent une décomposition hiérarchique des tenseurs, réduisant ainsi le nombre total de paramètres à entraîner et à stocker. Le résultat est un modèle plus léger, nécessitant moins de mémoire et de puissance de calcul, tout en conservant une performance comparable à celle du modèle original.

Ces techniques ne sont pas limitées aux réseaux neuronaux et peuvent s'étendre à d'autres types d'algorithmes d'apprentissage machine, offrant ainsi une grande polyvalence. Toutefois, il faut souligner que les gains de compression peuvent varier en fonction de la complexité du modèle et de la qualité de l'approximation tensorielle.

Optimisation combinatoire

Les réseaux de tenseurs peuvent servir à représenter efficacement des fonctions de coût dans des problèmes d'optimisation combinatoire tels que le problème du voyageur de commerce ou l'ordonnancement de tâches. Ils offrent une représentation compacte et précise de l'espace des solutions possibles, ce qui peut faciliter la recherche d'optimums globaux. Cela est rendu possible par le fait que ces problèmes d'optimisation peuvent être mappés sur un Hamiltonien, établissant ainsi un lien naturel entre l'optimisation combinatoire et les systèmes physiques quantiques ou classiques !

Codes de correction d'erreur en calcul quantique

Les codes de correction d'erreurs sont essentiels pour le développement d'ordinateurs quantiques fiables, car les qubits (bits quantiques) sont sensibles aux erreurs dues à la décohérence et aux interactions non désirées avec leur environnement. Les réseaux de tenseurs sont particulièrement prometteurs pour élaborer des codes de correction d'erreurs en informatique quantique en offrant des moyens efficaces de représenter, de manipuler et de corriger les états quantiques.

Simulation

Les réseaux de tenseurs offrent des méthodes efficaces pour le sampling dans les simulations, permettant ainsi une meilleure approximation des distributions de probabilité complexes, souvent nécessaires en physique quantique, mais aussi par extension dans des secteurs et domaines variés comme la finance et l'apprentissage machine.

En conclusion, les réseaux de tenseurs sont non seulement polyvalents, mais ils enrichissent également la science de l'information quantique grâce à leur structure de données, leurs algorithmes et leurs applications variées. Ces calculs peuvent d'ailleurs être effectués sur des machines traditionnelles, augmentant ainsi leur utilité et leur accessibilité. De plus, leur potentiel interdisciplinaire ouvre des voies dans des domaines aussi variés que la physique de la matière condensée, l'apprentissage automatique, l'optimisation d'un portefeuille d'actifs financiers et la correction d'erreur en calcul quantique. Malgré ces atouts, des défis et des questions ouvertes subsistent et méritent une attention accrue. Ces recherches futures pourraient directement contribuer aux algorithmes des prochains ordinateurs quantiques, affirmant ainsi la valeur durable des réseaux de tenseurs dans l'avancement scientifique.

Informatique quantique et langage Rust, une rapide exploration

Le monde Rust se préoccupe-t-il de quantique ? Réciproquement, l'informatique quantique utilise-t-elle le langage Rust pour certaines de ses qualités ? Pour des motifs pédagogiques ? Dans le cadre d'une solution quantique élaborée, voire commerciale ?

Benoît Prieur

Benoît s'intéresse depuis plusieurs années à l'informatique quantique et à Rust, entre autres sujets. Il a publié en 2022 un ouvrage (362 p.) à propos de ce langage aux éditions ENI. En 2023, toujours avec le même éditeur, il a publié une vidéo (1h22) de vulgarisation de l'informatique quantique.

https://www.benoit-prieur.fr/

C'est tout le propos de cet article, dans lequel je mène ma petite "enquête" et je tâche de faire un rapide état des lieux de la thématique Informatique quantique + Rust.

Première piste de recherche, le point d'entrée évident pour commencer à répondre à la question est le suivant : faire une recherche de bibliothèques, que l'on appelle caisses (crates) dans le monde Rust, dédiées à l'informatique quantique. Pour cela nous allons jeter un œil au registre des caisses de la communauté Rust, sur le site web dédié.

```
https://crates.io/
```

Première surprise, la recherche du mot-clé "quantum" donne de nombreux résultats, parmi lesquels plusieurs simulateurs quantiques programmés en Rust, sans qu'ils ne soient pas adossés à une autre solution quantique. L'objectif est en général d'ordre pédagogique.

Parmi ces crates "pédagogiques", celle qui sort du lot est **q1tsim**. Elle est par ailleurs relativement fréquemment utilisée (un usage par jour environ, selon crates.io). C'est la première caisse que nous allons présenter et utiliser ici.

La page sur crates.io :

```
https://crates.io/crates/q1tsim
```

Le dépôt Github associé :

```
https://github.com/Q1tBV/q1tsim
```

Peu de solutions professionnelles ou commerciales, à l'exception notable de **roqoqo**, adossé à la solution quantique de l'entreprise allemande HQS Quantum Simulations.

```
https://quantumsimulations.de/
```

La page sur crates.io :

```
https://crates.io/crates/roqoqo
```

Le dépôt Github associé :

```
https://github.com/HQSquantumsimulations/qoqo
```

Ce sont ces deux caisses (q1tsim et roqoqo) que nous allons découvrir dans la suite de cet article, en les utilisant.

Rust et quantique, par la pratique

Utilisation de la caisse q1tsim

Voyons ce que l'on peut faire et obtenir avec cette caisse. On ouvre Visual Studio Code et on crée un projet Rust exécutable à l'aide de l'outil cargo.

```
cargo new QuantumRustProject
```

On obtient comme résultat un projet Rust contenant le fichier Cargo.toml ainsi qu'un répertoire src contenant le fichier main.rs.

On commence par ajouter la référence à la caisse q1tsim que nous souhaitons utiliser. Pour cela on peut modifier le fichier Cargo.toml à la main, ou bien là encore utiliser l'outil cargo.

```
cargo add q1tsim
```

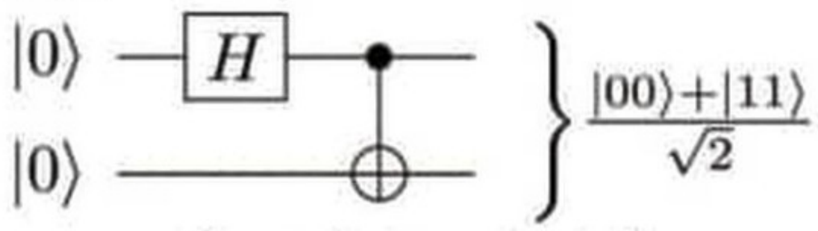
Notre fichier Cargo.toml ressemble alors à ceci.

```
[package]
name = "QuantumRustProject"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
q1tsim = "0.5.0"
```

Maintenant, commençons à utiliser cette caisse quantique, un exemple simple à coder correspondant à l'obtention d'un des états de Bell.

Les états de Bell correspondent à des états intriqués et maximisés entre deux qubits.

L'un des quatre états de Bell s'obtient avec le circuit quantique suivant qui sollicite une porte de Hadamard ainsi qu'une porte CNOT. Le résultat après mesure, dans la partie classique donc, devrait être 00 (le premier bit à 0 et le second également) ou 11. Les cas 01 ou 10 ne devraient pas être présents.



Obtention d'un des quatre états de Bell.

Dans notre fichier main.rs, on commence par déclarer l'objet Circuit issu de la caisse q1tsim que nous allons utiliser.

```
use q1tsim::circuit::Circuit;
```

Dans la fonction main, nous créons maintenant un circuit quantique qui fera l'objet d'une simulation quantique, par la suite. Le circuit quantique possède deux qubits (bits quantiques) en entrée, et deux bits classiques en sortie. En effet, on effectue une mesure quantique sur chacun des deux qubits d'entrée ce qui, du fait de l'effondrement de la

fonction d'onde lors de la mesure quantique, nous fait passer du monde quantique au monde classique.
Donc deux qubits en entrée, deux bits classiques en sortie ; on appelle le constructeur de Circuit avec ces paramètres (deux entrées, deux sorties). L'instance est nommée circuit.

```
let qubits = 2;
let bits_classiques = 2;
let mut circuit = Circuit::new(qubits, bits_classiques);
```

Remarque : *le mot-clé mut indique ici que l'objet circuit en mémoire est mutable. Sans cette précision, en Rust, les objets stockés en mémoire sont immutables par défaut.*

On ajoute ensuite une porte de Hadamard sur la première entrée.

```
circuit.h(0);
```

Puis on ajoute une porte CNOT entre la première entrée et la seconde entrée.

```
circuit.cx(0, 1);
```

Remarque : *les différentes portes quantiques disponibles sont stockées dans /src/gates, un fichier Rust d'extension .rs, par porte quantique.*

```
https://github.com/Q1tBV/q1tsim/tree/master/src/gates
```

Enfin, on ajoute une mesure quantique de chacun des deux bits quantiques.

```
circuit.measure_all(&[0, 1]);
```

Notre circuit quantique est alors prêt à être simulé. Comme toujours lors de travaux en informatique quantique, nous allons solliciter un grand nombre de fois notre circuit quantique. Le monde quantique étant probabiliste, il nous faut en quelque sorte procéder à un grand nombre de tirages aléatoires. Sachant qu'un seul tirage ne nous permettrait pas de tirer quelque conclusion.
Nous allons donc procéder à 10 000 tests et donc 10 000 mesures différentes du circuit.

```
let nb_essais = 10000;
circuit.execute(nb_essais);
```

Il nous faut maintenant pour chaque tirage, savoir si le résultat a été 00, 11, 01 ou 10. Sachant qu'en théorie, nous ne devrions pas avoir de résultat en 01 ou en 10. Pour chacun des quatre cas possibles, nous allons compter le nombre de mesures parmi les 10 000 qui ont donné respectivement 00, 11 et éventuellement 10 et 01. On accède aux résultats de la simulation et on affiche le nombre de mesures respectivement pour 00, 01, 10 et 11.

```
let hist = circuit.histogram_vec();
for resultats in hist.iter()
{
  for resultat in resultats.iter()
  {
    println!("{}", resultat);
  }
}
```

Le code de notre fichier main.rs est alors :

```
use q1tsim::circuit::Circuit;

fn main() {

let qubits = 2;
let bits_classiques = 2;
let mut circuit = Circuit::new(qubits, bits_classiques);
circuit.h(0);
circuit.cx(0, 1);
circuit.measure_all(&[0, 1]);
let nb_essais = 10000;
circuit.execute(nb_essais);
let hist = circuit.histogram_vec();
for resultats in hist.iter()
{
  for resultat in resultats.iter()
  {
    println!("{}", resultat);
  }
}
}
```

Il est temps maintenant de tester notre programme. On commence donc par le compiler.

```
cargo build
```

Bonne nouvelle, le programme compile correctement, même si on obtient quelques warnings.

```
> warning: `QuantumRustProject` (bin "QuantumRustProject") generated 5
warnings
  Finished dev [unoptimized + debuginfo] target(s) in 4.77s
```

Passons à une première exécution de notre programme.

```
cargo run
```

Nous obtenons le résultat suivant dans la console de Visual Studio Code.

```
> 5004
> 0
> 0
> 4998
```

Nous rassemblons le résultat dans le tableau suivant.

Valeurs mesurées	00	01	10	11
Nombre de mesures / 10.000	5004	0	0	4996

Conformément à ce que nous connaissons de cet état de Bell, il n'y a aucune mesure parmi les 10.000 mesures qui donne 01 ou 10. Nous obtenons par ailleurs 5004 mesures en 00 et 4996 en 11, ce qui est presque conforme à l'équiprobabilité théorique entre 00 et 11.
Précisons en conclusion que q1tsim peut être ardu dès lors que l'on cherche à faire des choses plus compliquées. En particulier, car la documentation est relativement peu détaillée, obligeant fréquemment à lire le code de la caisse,

pour comprendre et déduire comment utiliser telle ou telle fonction de mesure quantique par exemple. À noter tout de même qu'il est relativement aisé de définir ses propres portes quantiques en complément de celles déjà existantes, ce qui autorise à produire des programmes quantiques très élaborés.
Passons à présent à la caisse roqoqo de la société HQS.

Utilisation de la caisse roqoqo

L'exemple de programme, fil conducteur de cette section, est un peu plus pointu que pour q1tsim. En effet, nous allons simuler une téléportation quantique. Je vais d'abord rappeler de quoi il s'agit exactement, puis par quel circuit quantique théorique nous pouvons atteindre une telle téléportation. Puis on entrera dans le vif du sujet, la programmation en Rust à l'aide de roqoqo.

La téléportation quantique

On se place dans l'hypothèse selon laquelle Alice détient un

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$:

Alice souhaite transmettre ce qubit, sa valeur du moins, donc les valeurs α et β, à son ami Bob. En réalité le terme téléportation est quelque peu impropre ici, c'est plutôt une reconstitution de la valeur transmise, le qubit original, comme on le verra, sera mesuré quantiquement et est donc à un certain moment détruit.
Pour définir la valeur à transmettre, et ensuite vérifier ce que reçoit Bob, nous allons agir sur deux angles qui participent à construire les valeurs alpha et bêta : les angles theta et phi.

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

Pour rappel, ci-dessous la sphère de Bloch impliquant les angles theta, phi et psi, et leurs participations dans la nature d'un état quantique. **Figure 1**

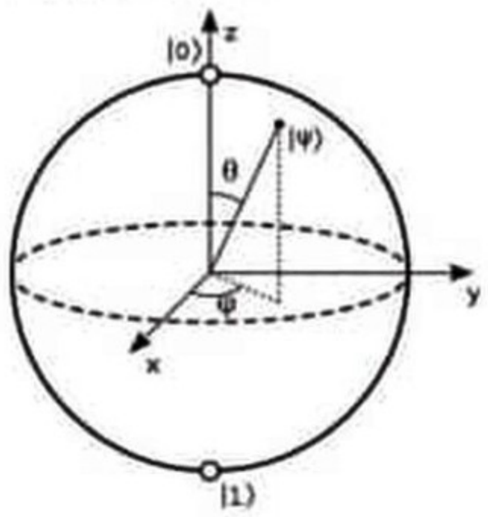
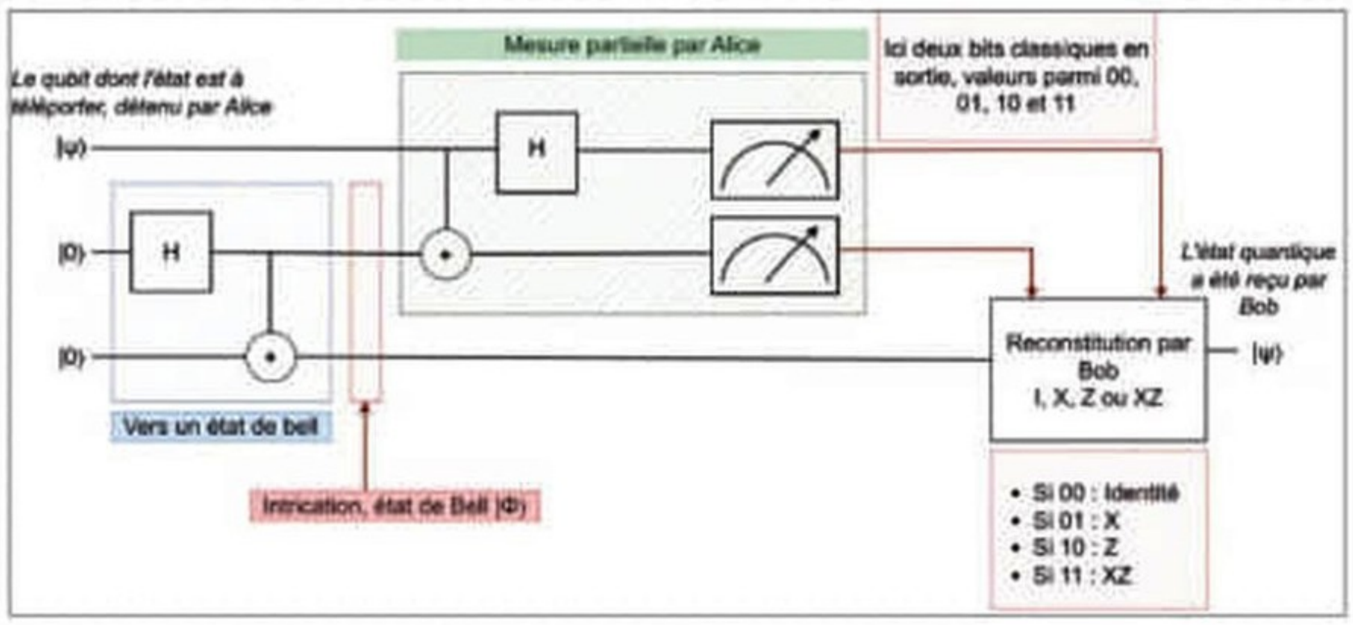


Figure 1



Le schéma suivant représente le circuit quantique de la téléportation.

Plusieurs remarques :

- En entrée, trois qubits. De haut en bas :
  - Le qubit d'Alice qui souhaite le partager à Bob, **qA**.
  - Un qubit que nous appellerons le qubit d'intrication, **ql**.
  - Le qubit de Bob destiné finalement à reconstituer le qubit détenu par Alice, **qB**.
- On commence par obtenir l'intrication de ql et de qB, en obtenant un des états de Bell. C'est exactement le circuit que nous avions conçu précédemment avec q1tsim.
- Puis on obtient l'intrication de qA et ql par l'utilisation d'une porte CNOT et d'une porte de Hadamard.
- À ce stade, par le jeu de la double intrication, la valeur de qA est déjà d'une certaine manière incluse dans qB, modulo une éventuelle rotation, comme nous le verrons ensuite.
- On procède ensuite à une mesure partielle des deux premiers qubits, qA et ql. Cette mesure nous place de fait dans le monde classique. La mesure de qA est égale à 0 ou à 1, et il en est de même pour le qubit d'intrication ql.
- On est alors en capacité de transmettre à Bob, cette information : 00, 01, 10 ou 11.
- Selon la valeur (00, 01, 10 ou 11) on appliquera une transformation avant mesure de qB.
- Selon la valeur, la transformation appliquée est :
- Identité, si 00
  - Porte X, si 01
  - Porte Z, si 10
  - Portes X et Z, si 11

Téléportation quantique avec roqoqo

Bien évidemment on commence par créer un projet Rust que nous nommons TeleportationRoqoqo.

```
cargo new TeleportationRoqoqo
```

On modifie le fichier Cargo.toml pour référencer la caisse roqoqo.

```
cargo add roqoqo
```

On ajoute au passage quelques autres caisses liées à roqoqo dont nous aurons besoin. Après ajouts, le fichier Cargo.toml ressemble à ceci.

```
[package]
name = "TeleportationRoqoqo"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
qoqo_calculator = "1.1.2"
roqoqo = "1.6.0"
roqoqo-quest = "0.11.1"
```

Commençons à présent à coder la téléportation elle-même. L'idée ici est de suivre plus ou moins le schéma et d'ajouter

de proche en proche les différents sous-circuits. On commence par créer une fonction capable de définir une valeur pour le qubit d'Alice, en agissant sur les angles theta et phi.

```
fn creation_qubit_alice(angle_thet: CalculatorFloat, angle_phi: CalculatorFloat) ->
Circuit {
  let mut circuit = Circuit::new();
  circuit += ops::RotateY::new(0, angle_thet);
  circuit += ops::RotateZ::new(0, angle_phi);
  circuit
}
```

Puis on l'utilise immédiatement, avec theta = Pi / 6 et phi = 0. Cela correspond à la valeur que Alice cherchera à transmettre à Bob. Avec phi = 0, on pourra vérifier que l'on obtient bien dans les résultats de Bob, respectivement la valeur alpha égale à cosinus(theta / 2) et la valeur bêta égale à sinus(theta / 2).

```
let circuit_initial = creation_qubit_alice(CalculatorFloat::Float(PI)/6.0, Calculator
Float::Float(0f64));
```

On réalise ensuite le sous-circuit correspondant à l'état de Bell, entre ql et qB, c'est-à-dire entre le qubit du milieu et le qubit de Bob.

```
// On réalise le sous-circuit d'intrication en bas entre le qubit d'intrication et le qubit de Bob
let mut circuit_intrication_bob = Circuit::new();
circuit_intrication_bob += ops::Hadamard::new(1);
circuit_intrication_bob += ops::CNOT::new(1, 2);
```

Notez bien qu'ici le sous-circuit est indépendant, il n'est pas encore connecté au circuit global. On fera cet assemblage à la toute fin, quand nous aurons tous les sous-circuits.

On réalise maintenant le sous-circuit correspondant à la partie verte du schéma : la porte CNOT entre qA et ql, puis la porte de Hadamard sur qA.

```
// On réalise le sous-circuit relatif aux deux premiers qubits
let mut circuit_superieur = Circuit::new();
circuit_superieur += ops::CNOT::new(0, 1);
circuit_superieur += ops::Hadamard::new(0);
```

Puis on réalise le sous-circuit de la mesure partielle.

```
// On réalise le sous-circuit d'intrication en bas entre le qubit d'intrication et le qubit de Bob
let mut circuit_intrication_bob = Circuit::new();
circuit_intrication_bob += ops::Hadamard::new(1);
circuit_intrication_bob += ops::CNOT::new(1, 2);
```

Pour que notre circuit soit complet, il faut ajouter le sous-circuit conditionnel vers Bob.

```
// On réalise le circuit conditionnel selon le résultat classique après mesure
// Pour rappel I, X, Z, ou XZ
let mut circuit_conditionnel_z = Circuit::new();
circuit_conditionnel_z += ops::PauliZ::new(2);
let mut circuit_conditionnel_x = Circuit::new();
circuit_conditionnel_x += ops::PauliX::new(2);
let mut circuit_conditionnel = Circuit::new();
circuit_conditionnel += ops::PragmaConditional::new("registre".to_string(), 1,
circuit_conditionnel_x);
circuit_conditionnel += ops::PragmaConditional::new("registre".to_string(), 0,
circuit_conditionnel_z);
```

Notre circuit est maintenant complet. Nous allons toutefois ajouter un sous-circuit qui nous permettra d'obtenir les valeurs complexes de Bob, et ainsi vérifier que l'on obtient la valeur transmise (en fait, nous allons vérifier que l'angle theta est correctement transmis).

```
// On ajoute un élément de vérification, capable de nous donner le résultat reçu par Bob
// Supposé être celui envoyé par Alice
let mut verification = Circuit::new();
verification += ops::DefinitionComplex::new("psi".to_string(), 8, true);
verification += ops::PragmaGetStateVector::new("psi".to_string(), Some(Circuit::new()));
```

Reste à présent à assembler tous nos sous-circuits.

```
// On a ici l'ensemble de nos sous-circuits, on fait l'assemblage
let circuit_principal = circuit_initial
+ circuit_intrication_bob
+ circuit_superieur
+ circuit_mesure
+ circuit_conditionnel
+ verification;
```

Nous sommes maintenant prêts pour la simulation de la téléportation quantique.

```
let backend = Backend::new(3);
let resultat = backend.run_circuit(&circuit_principal);
let (resultat_bits, resultat_flottant, resultat_complexe) =
resultat.unwrap();

println!("——");
println!("Résultat Bob : {:?}", resultat_complexe["psi"]);
```

On compile, puis on exécute. On obtient en ligne de commande ceci.

```
——
Résultat Bob : [[Complex { re: 0.0, im: 0.0 }, Complex { re: 0.0, im: 0.0 }, Complex { re:
0.965925826289068, im: 0.0 }, Complex { re: 0.0, im: 0.0 }, Complex { re: 0.0, im: 0.0
}, Complex { re: 0.0, im: 0.0 }, Complex { re: 0.2588190451025207, im: 0.0 }, Complex
{ re: 0.0, im: 0.0 }]]
```

Si l'on se rappelle l'équation suivante incluant le cosinus de theta et le sinus de theta ; rappelons-nous également que phi est égale à 0, ainsi la partie réelle, mais aussi la partie imaginaire, ne dépendent que de l'angle theta.

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

Soit theta = Pi / 6. Calculons le cosinus et le sinus de l'angle (theta / 2)

- cosinus ((Pi / 6) / 2) = 0.96592582628
- sinus ((Pi / 6) / 2) = 0.2588190451

Or ce sont bien les valeurs transmises et obtenues par Bob.
Pour rappel.

- 0.9659258262890682
- 0.2588190451025207

Notre téléportation quantique, en Rust avec le simulateur de HQS Quantum Simulations, a donc bien fonctionné.
Les codes de cet article sont disponibles sur Github :
https://github.com/henprieur/ProgrammezQuantique2023_Rust/tree/main

NOUVEAU !

Numéro Spécial de 134 pages

40 ans du Macintosh

*(16,99 € + de frais de port)

Commandez dès maintenant sur technosaures.fr, sur amazon.fr ou sur programmez.com

Programming in the Age of Hybrid Computing

USING NVIDIA CUDA QUANTUM TO PROGRAM CPUS, GPUS AND QPUS

Esperanza Cuenca Gómez
Developer Relations Manager, Quantum Computing

NVIDIA CUDA Quantum: the Platform for Hybrid – Classical Computing

To do algorithm research and build applications for future quantum advantage, a bridging technology is needed to enable dynamic workflows across disparate system architectures. With a unified and open programming model, NVIDIA CUDA Quantum is a first-of-its-kind, open-source platform for integrating and programming QPUs, GPUs, and CPUs in one system. NVIDIA CUDA Quantum enables GPU-accelerated system scalability and performance across heterogeneous QPU, CPU, GPU, and emulated quantum system elements. **Figure 1**

NVIDIA CUDA Quantum consists of both a specification and a compiler NVQ++. It delivers a unified programming model designed for quantum processors (either actual or emulated) in a hybrid setting—that is, CPUs, GPUs, and QPUs working together.

Example: Hybrid Quantum Neural Networks

The example below highlights a hybrid quantum neural network workflow with CUDA Quantum and Pytorch where both layers are GPU accelerated to maximize performance. **Figure 2**

We perform binary classification on the MNIST dataset where data flows through the neural network architecture to the quantum circuit whose output is used to classify hand written digits.

[1]:

```
# Import the relevant packages

# !pip install matplotlib torch torchvision

import
import
from
from
import
import
import

import
from
```

[2]:

```
# GPU utilities
```

```
"nvidia"  # Set CUDAQ to run on GPU's

# If this is True then the NVIDIA drivers are correctly installed

  # Counts the number of GPU's available

  0

"cuda:0" if  else "cpu"
```

[3]:

```
# Training set
 = 140
```

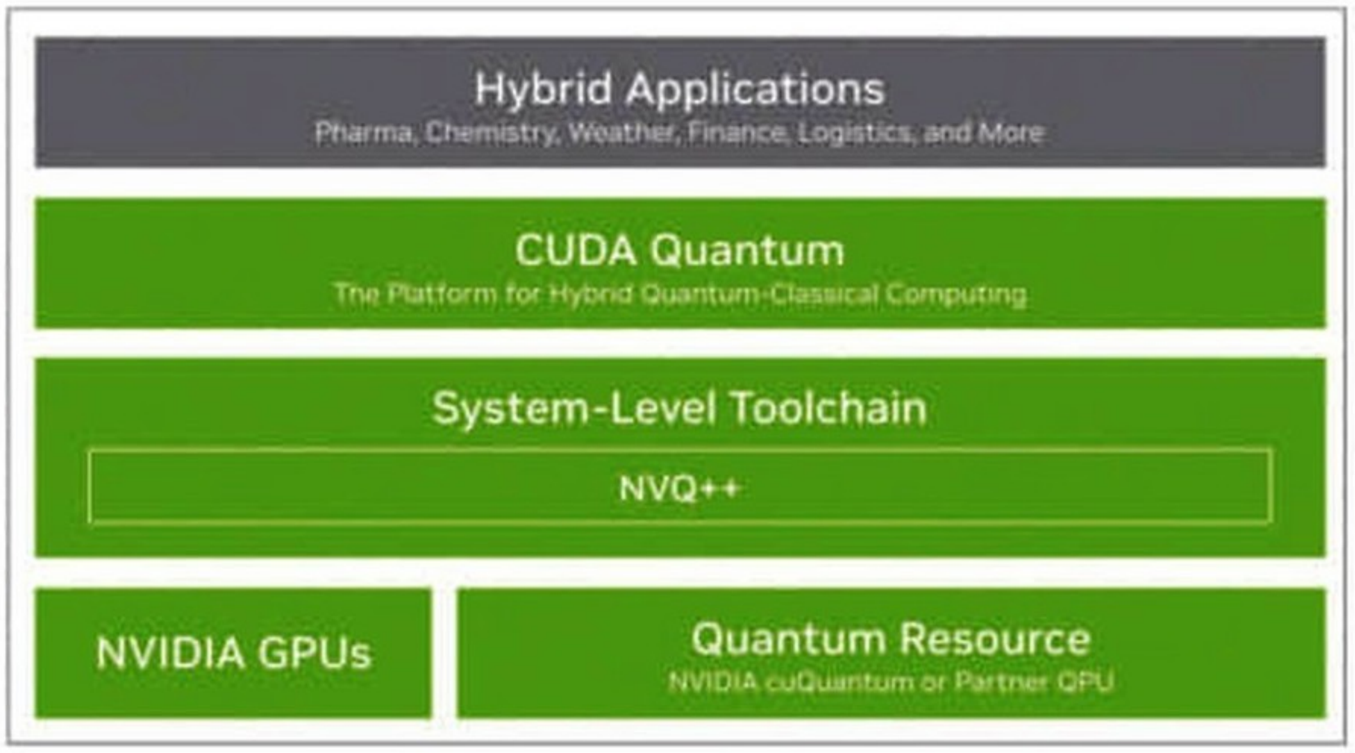


Figure 1: *CUDA Quantum Overview*

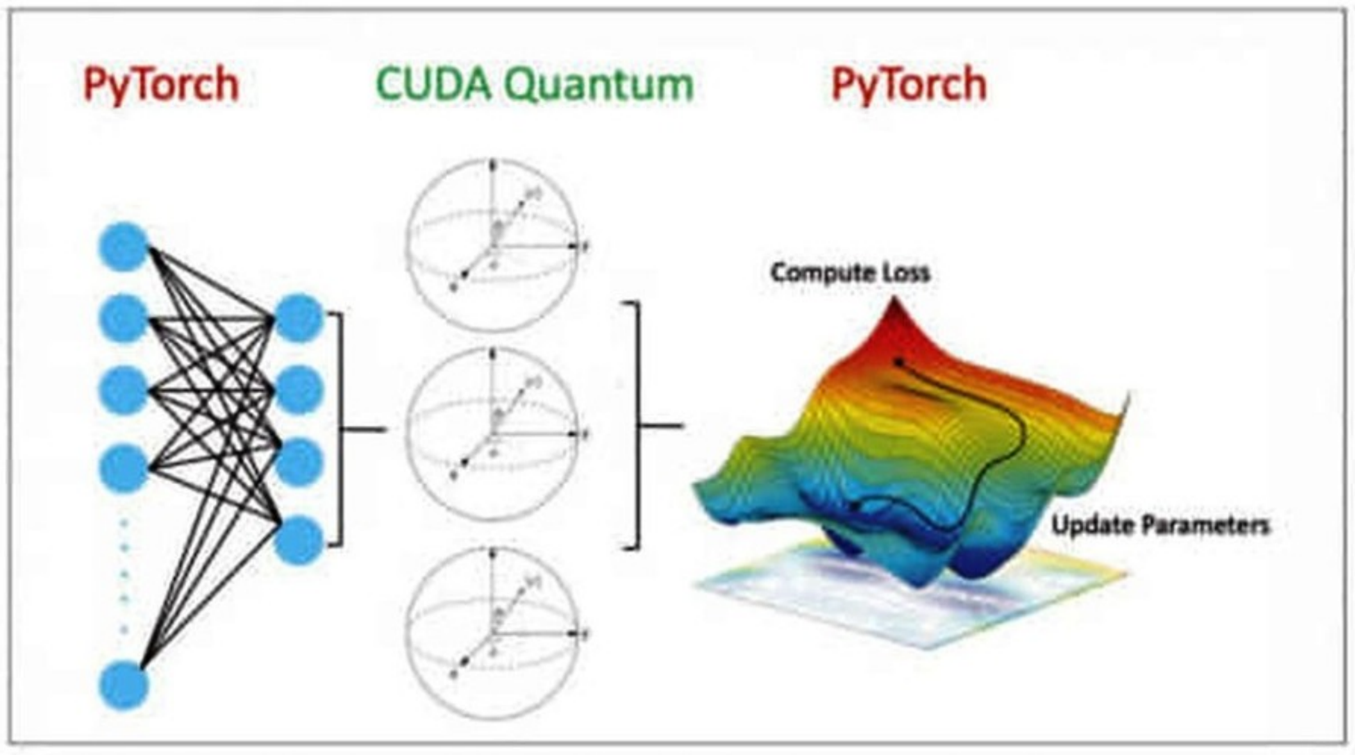


Figure 2: *hybrid quantum neural network workflow with CUDA Quantum and Pytorch*

```
# Leaving only labels 0 and 1
```

```
# Test set
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|[redacted] 9912422/9912422 [00:00<00:00, 72632367.27it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|[redacted] 28881/28881 [00:00<00:00, 85487433.89it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|[redacted] 1648877/1648877 [00:00<00:00, 27770426.19it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|[redacted] 4542/4542 [00:00<00:00, 16394603.07it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

[4]:

```
class QuantumCircuit
    """This class defines the quantum circuit structure and the run method which is used to calculate an expectation value"""

    def __init__(self, qubit_count: int):
        """Define the quantum circuit in CUDA Quantum"""

        # Variational gate parameters which are optimised during training

    def run(self, input_tensor) -> torch.Tensor:
        """Excetute the quantum circuit to output an expectation value"""

        return
```

[5]:

```
class QuantumFunction(Function):
    """Allows the quantum circuit to pass data through it and compute the gradients"""

    @staticmethod
    def forward(ctx, input, quantum_circuit, shift) -> torch.Tensor:

        # Save shift and quantum_circuit in context to use in backward

        # Calculate exp_val

        return

    @staticmethod
    def backward(ctx, grad_output):
        """Backward pass computation via finite difference parameter shift"""

        return
```

```
for  in range len
    return
```

[6]:

```
class QuantumLayer
    """Encapsulates a quantum circuit and a quantum function into a quantum layer"""

    def __init__
        super
        self. # 1 qubit quantum circuit

    def forward
        return
```

[7]:

```
class Net

    def __init__
        super
        # Neural network structure
        # Input is the magnitude of the parameter shifts to calculate gradients

    def forward
        # Reshapes required to satisfy input dimensions to CUDAQ
```

```
    return
```

[8]:

```
# We move our model to the CUDA device to minimise data transfer between GPU and CPU

for  in range
    for  # batch training

    # Forward pass

    # Backward pass

    # Optimize the weights

    if
```

Training [5%] Loss: -1.1866
Training [10%] Loss: -1.3703
Training [15%] Loss: -1.3844
Training [20%] Loss: -1.4073
Training [25%] Loss: -1.4137
Training [30%] Loss: -1.4255
Training [35%] Loss: -1.4483
Training [40%] Loss: -1.4524
Training [45%] Loss: -1.4641
Training [50%] Loss: -1.4565
Training [55%] Loss: -1.4760
Training [60%] Loss: -1.4755
Training [65%] Loss: -1.4795
Training [70%] Loss: -1.4867
Training [75%] Loss: -1.4888

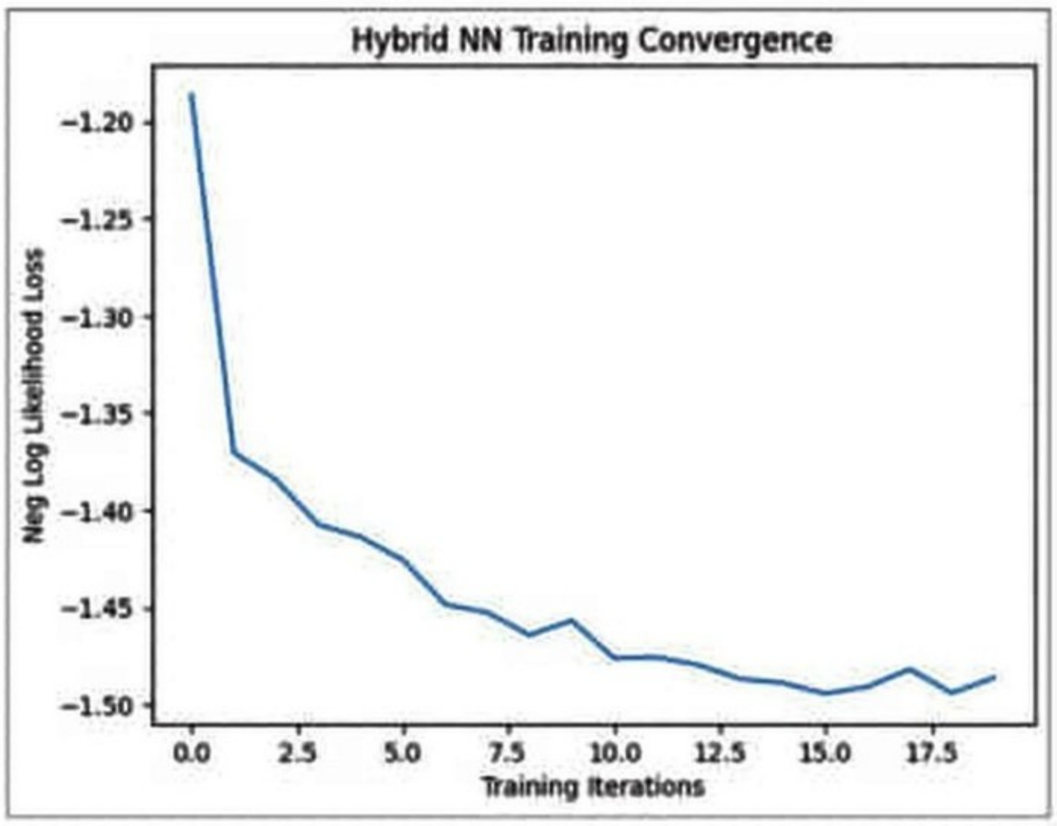


Figure 3

```
Training [80%]  Loss: -1.4942
Training [85%]  Loss: -1.4907
Training [90%]  Loss: -1.4817
Training [95%]  Loss: -1.4939
Training [100%] Loss: -1.4860
```

[9]:

```
Text(0, 0.5, 'Neg Log Likelihood Loss')
```

iigure 3

[10]:

```
# Testing on the test set
```

```
Performance on test data:
    Accuracy: 100.0%
```

Learn more

NVIDIA CUDA Quantum github
NVIDIA CUDA Quantum
Merge Ahead: Researcher Takes Software Bridge to Quantum Computing
NVIDIA Special Address at Q2B: Defining the Quantum Accelerated Supercomputing Platform

PROGRAMMEZ!

Directives de compilation

Programmez! hors-série n°13
HIVER 2023

Directeur de la publication & rédacteur en chef
François Tonic
ftonic@programmez.com

Contacter la rédaction
redaction@programmez.com

Les contributeurs techniques
Axel Terrazzini, Tamuz Danzig, James Clarke, Anne Matsuura, Jean-Michel Torres, Aziz Ngoueya Akanji Benga, Olivier Hess, Anne-Lise Guilmin, Robert Wang, Thomas Ayral, Jason Mueller, Anthony Chagneau, Gérard Fleury, Philippe Lacomme, Caroline Prodhon, Michel Kurek, Benoît Prieur, Esperanza Cuenca Gomez

Maquette
Pierre Sandré

Marketing – promotion des ventes
Agence BOCONSEIL - Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT
Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression
SIB Imprimerie, France

Dépôt légal
A parution

Commission paritaire
1225K78366

ISSN
2279-5001

Abonnement
Abonnement (tarifs France) : 49 € pour 1 an, 79 € pour 2 ans. Etudiants : 39 €. Europe et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 €.
Autres pays : consultez les tarifs sur www.programmez.com.
Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 45 € pour 1 an.
Accès aux archives : 25 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Les partenaires 2023 de

PROGRAMMEZ!

Le magazine des dév - CTO & Tech Lead

Niveau maître Jedi

salesforce

Progress®

Niveau padawan

OSAXiS

Vous voulez soutenir activement Programmez! ?
Devenir partenaires de nos dossiers en ligne et de nos événements ?

Contactez-nous dès maintenant :

ftonic@programmez.com

Informatique quantique :

de la théorie à la pratique

Qu'est-ce que l'informatique quantique ? Quel impact le quantique aura-t-il sur l'avenir des ordinateurs et de la programmation ? Découvrez deux ouvrages d'experts pour mieux appréhender cette révolution technologique, comprendre ses enjeux et vous former à l'évolution des langages de programmation.

Éditions EYROLLES