2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΜΗΧΑΝΙΚΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΔΙΑΔΙΚΤΥΑΚΕΣ & ΦΟΡΗΤΕΣ ΕΦΑΡΜΟΓΕΣ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σύγκριση των framework κβαντικού υπολογισμού Qiskit, PyQuil και κβαντικών Προσομοιωτών

ΓΑΛΑΝΗΣ ΗΛΙΑΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΣΑΒΒΑΣ ΗΛΙΑΣ

ΛΑΡΙΣΑ
ΝΟΕΜΒΡΙΟΣ 2020

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Δρ. Ηλία Σάββα, Καθηγητή του Πανεπιστημίου Θεσσαλίας, για την διαρκή συνεργασία και καθοδήγηση του καθώς και για την ευκαιρία που μου έδωσε να ασχοληθώ με έναν από τους πλέον σύγχρονους και ενδιαφέροντες τομείς της επιστήμης των υπολογιστών.

Θα ήθελα επίσης να ευχαριστήσω:

Την οικογένεια μου, για την υπομονή, την στήριξη τους καθ' όλη την διάρκεια των προπτυχιακών σπουδών μου και για την ώθηση που μου έδωσαν να ξεκινήσω το μεταπτυχιακό.

Την Μαριέττα, που πάντα πιστεύει σε εμένα, είναι στο πλάι μου, μου δίνει κίνητρο, κουράγιο και χάρη στην οποία η διπλωματική μου εργασία ολοκληρώθηκε εγκαίρως.

Περιεχόμενα

1. Περίληψη

Η παρούσα διπλωματική εργασία έχει σκοπό να εξηγήσει όσο πιο κατανοητά γίνεται, τι είναι ένας κβαντικός υπολογιστής, πως λειτουργεί και ποιες πρακτικές εφαρμογές μπορεί να έχει στο μέλλον. Αρχικά θα γίνει αναφορά σε εισαγωγικές έννοιες όπως στο τι είναι η κβαντομηχανική που αποτελεί όλο το υπόβαθρο της σύγχρονης επιστήμης και τεχνολογίας, στην υπέρθεση (supersposition) και στον κβαντικό εναγκαλισμό ή αλλιώς κβαντική διεμπλοκή (quantum entaglement), ιδιότητες πάνω στις οποίες στηρίζεται ένας κβαντικός υπολογιστής και είναι απαραίτητες για την κατανόηση της λειτουργίας του. Θα αναφερθούν γνωστοί κβαντικοί αλγόριθμοι και θα αναπτυχθούν με την βοήθεια των framework Qiskit της IBM και PyQuil της Rigetti. Συγκεκριμένα θα αναπτυχθεί ο αλγόριθμος Deutsch-Jozsa (ο οποίος αποτέλεσε πηγή έμπνευσης άλλων αλγορίθμων όπως ο γνωστός αλγόριθμος του Shor, που αναμένεται να φέρει αλλαγές στην ασφάλεια των δεδομένων όπως την ξέρουμε σήμερα) και ο αλγόριθμος Bernstein-Vazirani. Τέλος θα γίνει σύγκριση μεταξύ των δύο γλωσσών σε ότι έχει να κάνει με την σχεδιαστική τους προσέγγιση, την ευχρηστία, την απόδοση, τους κβαντικούς τους εξομοιωτές, καθώς και τις βιβλιοθήκες που χρησιμοποιούν.

1.1. Τι είναι κβαντομηχανική;

Η κβαντομηχανική είναι μια θεωρία της φυσικής (πλέον πολύ καλά θεμελιωμένη και πειραματικά) η οποία στηρίζεται σε πιθανότητες (ιδιότητα πάνω στην οποία στηρίζεται μαθηματικά ο κβαντικός υπολογισμός) και αναπτύχθηκε για να εξηγήσει φαινόμενα τα οποία η κλασική φυσική δεν μπορούσε. Τέτοια φαινόμενα παρατηρούνται κυρίως σε επίπεδο ατομικών και υποατομικών σωματιδίων. Σε αυτό το επίπεδο οι νόμοι της κλασικής φυσικής παύουν να ισχύουν. Η βασική διαφορά μεταξύ κβαντικής και κλασικής φυσικής είναι η Αρχή της Αβεβαιότητας ή αλλιώς Αρχή της Απροσδιοριστίας. Στην Κβαντομηχανική η αβεβαιότητα των μετρήσεων δεν οφείλεται σε κάποιο σφάλμα των οργάνων όπως στην Κλασική Φυσική, αλλά είναι ενδογενής. Εισάγει λοιπόν ένα αναπόφευκτο στοιχείο αδυναμίας πρόβλεψης και τυχαιότητας με αποτέλεσμα να περνάμε από την Ντετερμινιστική φύση της Κλασικής Φυσικής στην μη Ντετερμινιστή της Κβαντικής Φυσικής (Yanofsky & Mannucci, 2008). Ξεκινάνε λοιπόν να συμβαίνουν φαινόμενα τα οποία ξεφεύγουν από την ανθρώπινη αντίληψη σχετικά με τον κόσμο. Συνεπώς δεν πρέπει να απογοητεύεται κανείς αν διαπιστώσει ότι δεν μπορεί να κατανοήσει εντελώς την Κβαντική Θεωρία. Όπως έχει πει και ο μεγάλος φυσικός Richard Feynman «όποιος ισχυρίζεται ότι καταλαβαίνει την κβαντομηχανική, σημαίνει ότι δεν την καταλαβαίνει».

1.2. Τι είναι κβαντικός υπολογισμός;

Η κβαντομηχανική εκτός από μια ενδιαφέρουσα θεωρία η οποία κλόνισε την αντίληψη που υπήρχε για τον κόσμο είχε και σαν αποτέλεσμα την δημιουργία διαφόρων επιμέρους επιστημονικών τομέων που βασίζονται σε αυτή και την εφαρμόζουν. Ο κβαντικός υπολογισμός είναι ένας συναρπαστικός νέος τομέας ο οποίος μπορούμε να πούμε ότι βρίσκεται εκεί που συμβάλουν η επιστήμη των υπολογιστών, τα μαθηματικά και η φυσική και εκμεταλλεύεται πολλές

από αυτές τις παράξενες και «απόκοσμες» ιδιότητες της κβαντομηχανικής προκειμένου να διευρύνει τους υπολογιστικούς ορίζοντες. (Yanofsky & Mannucci, 2008). Επί της ουσίας οι κβαντικοί υπολογισμοί είναι δράσεις τελεστών που έχουν ως αποτέλεσμα την περιστροφή διανυσμάτων στο χώρο Hilbert. Ο κβαντικός υπολογισμός με την σειρά του όπως ήταν αναμενόμενο αποτέλεσε εφαλτήριο έμπνευσης δημιουργίας των κβαντικών αλγορίθμων. Οι κβαντικοί αλγόριθμοι για πολλά χρόνια βρισκόντουσαν σε καθαρά θεωρητικό επίπεδο καθώς δεν υπήρχε το «υλικό» πάνω στους οποίους θα μπορούσαν να «τρέξουν». Τα τελευταία χρόνια όμως έχει γίνει σημαντική πρόοδος από μεγάλες εταιρίες όπως η Rigetti, η IBM, η Google και άλλες, με αποτέλεσμα να μπορούμε ποια να αναπτύξουμε κάποιους από αυτούς, σε αυτό που ονομάζεται κβαντικός υπολογιστής.

1.3. Τι είναι κβαντικός υπολογιστής;

Κβαντικός Υπολογιστής ονομάζεται μια υπολογιστική συσκευή η οποία εκμεταλλεύεται ιδιότητες της κβαντομηχανικής με στόχο την εκτέλεση υπολογισμών. Η ιδέα για την κατασκευή ενός κβαντικού υπολογιστή προέρχεται από τον μεγάλο φυσικό Richard Feynman. Περί τα 1980 πρότεινε την κατασκευή ενός κβαντικού υπολογιστή ο οποίος εκμεταλλευόμενος τις αρχές της κβαντομηχανικής θα μπορεί να λύσει προβλήματα ταχύτερα από ότι ένας κλασικός υπολογιστής. Από την άλλη πλευρά μέχρι και σήμερα οι επιστήμονες των υπολογιστών χρησιμοποιούν τις μηχανές Turing ως ένα είδος ακαδημαϊκής στενογραφίας που συμπυκνώνει τις αρχές στις οποίες βασίζονται οι κλασικοί υπολογιστές. Οι κύριες ιδιότητες της κβαντομηχανικής που εκμεταλλεύονται οι κβαντικοί υπολογιστές είναι η **υπέρθεση** και η **κβαντική διεμπλοκή**. Σε αντίθεση λοιπόν με έναν κλασικό υπολογιστή στον οποίο η στοιχειώδης μονάδα πληροφορίας είναι το bit σε ένα κβαντικό υπολογιστή έχουμε το κβαντικό bit ή αλλιώς **qubit.** (Hey & Walters, 2005)

1.4. Κβαντικές Ιδιότητες για τον Κβαντικό Υπολογισμό (Το qubit)

1.4.1. ***Κβαντική διεμπλοκή (quantum entaglement):*** Είναι μια κβαντική κατάσταση κατά τη οποία δύο σωματίδια ή ομάδες σωματιδίων αλληλοεπηρεάζονται ακαριαία. Στον πυρήνα της κλασικής φυσικής βρίσκεται το ότι ένα αντικείμενο σε ένα σύστημα μπορεί να επηρεαστεί μόνο από κοντινά σε αυτό αντικείμενα ή δυνάμεις που ασκούνται σε αυτό. Για να διαπιστώσουμε γιατί ένα φαινόμενο συμβαίνει σε ένα συγκεκριμένο σημείο αρκεί να εξετάσουμε όλα τα φαινόμενα και τις δυνάμεις κοντά σε αυτό το σημείο. Αυτό ονομάζεται «τοπικότητα». Στην κβαντομηχανική ένα από τα πιο αξιοθαύμαστα χαρακτηριστικά της είναι ότι μπορεί να προβλέψει συγκεκριμένα αποτελέσματα τα οποία όμως συμβαίνουν σε μη τοπικό επίπεδο. Δύο σωματίδια μπορεί να συνδεθούν ή να «διεμπλακούν» (entagled) με τέτοιο τρόπο ώστε οποιαδήποτε αλλαγή κάνουμε στο ένα, μεταφέρεται ακαριαία και στο άλλο ανεξαρτήτως της απόστασης που τα χωρίζει (μπορεί να βρίσκεται και πολλά έτη φωτός μακριά) και συνεπώς γνωρίζοντας την τιμή του ενός, θα γνωρίζουμε αυτόματα και την τιμή του άλλου. (Yanofsky & Mannucci, 2008) Σε ένα κβαντικό υπολογιστή συνεπώς μπορούμε να έχουμε δύο qubit στα οποία η πληροφορία του ενός θα μεταφέρεται και θα επηρεάζει αμέσως το άλλο.

1.4.2. ***Υπέρθεση (superpositon):*** Υπέρθεση ονομάζεται η ιδιότητα που έχει ένα σωματίδιο να βρίσκεται σε 2 ή περισσότερες καταστάσεις ταυτόχρονα. Όπως και σε ένα κύμα στην κλασική φυσική οι 2 κβαντικές αυτές καταστάσεις μπορούν να «αθροιστούν» και να προκύψει μια νέα κβαντική κατάσταση. (Quantum Supersposition, 2020) Η μέτρηση που θα κάνουμε σε ένα σωματίδιο που βρίσκεται σε υπέρθεση θα το επηρεάσει με αποτέλεσμα να μας δώσει μία από τις 2 καταστάσεις. Σε επίπεδο κβαντικού υπολογιστή αυτό σημαίνει ότι ένα qubit μπορεί να είναι είτε 0, είτε 1 είτε 0 και 1 ταυτόχρονα, είτε μια ενδιάμεση κατάσταση όπου όταν μετρηθεί θα μας δώσει ένα αποτέλεσμα με βάση κάποια πιθανότητα. Η μέτρηση ενός qubit έχει σαν αποτέλεσμα αυτό να «καταρρέει» σε μια συγκεκριμένη κατάσταση και το αποτέλεσμα θα είναι 0 ή 1. (Silva, 2018).

Εκμεταλλευόμενοι τις ιδιότητες αυτές και συνδυάζοντάς τες ανοίγονται νέες προοπτικές υπολογισμών και ανάπτυξης αλγορίθμων οι οποίοι είναι πολύ πιο γρήγοροι από τους αλγορίθμους που τρέχουν οι κλασικοί υπολογιστές. Αντί να έχουμε ένα bit το οποίο θα βρίσκεται σε μια κατάσταση κάθε φορά, με ένα qubit μπορεί να βρίσκεται σε πολλές καταστάσεις ταυτόχρονα. Αυτονόητα συνεπάγεται ένας μεγάλος βαθμός παραλληλισμού, ο οποίος όμως για να αξιοποιηθεί δεν πρέπει να μετρηθεί το qubit. Πρακτικά ενώ το κλασικό bit μπορεί να αποθηκεύσει πληροφορία 0 ή 1, ένα qubit χάρη στην υπέρθεση αποθηκεύεται πληροφορία μεταξύ 0 και 1 ταυτόχρονα. Είναι προφανές ότι με λιγότερα qubits μπορούν να γίνουν πιο πολύπλοκοι υπολογισμοί. Αυτό σημαίνει ότι σε αντίθεση με τους κλασικού υπολογιστές η ισχύς των οποίων αυξάνεται γραμμικά όσο προστίθενται σε αυτούς επιπλέον bits, στους κβαντικούς η υπολογιστική τους ισχύς αυξάνεται εκθετικά αναλόγως πόσα qubits χρησιμοποιούν. Για παράδειγμα αν σε έναν υπολογιστή με 100 qubits προστεθεί ακόμα 1 η ισχύς του θα διπλασιαστεί. Βέβαια οι υπολογισμοί δεν εκτελούνται πραγματικά παράλληλα, αλλά η πληροφορία οργανώνεται με τέτοιο τρόπο ώστε αυτή να διαμοιράζεται σε πολλά κομμάτια, σαν ένα και έπειτα η επεξεργασία της πληροφορίας γίνεται παράλληλα (Silva, 2018). Ένα καλό παράδειγμα για την κατανόηση αυτού του φαινομένου είναι το εξής: Έστω ότι υπάρχουν 2 βιβλία 100 σελίδων το κάθε ένα. Το ένα είναι κλασικό και το άλλο κβαντικό. Στο κλασικό για κάθε μία σελίδα που θα διαβάσουμε εμπλουτίζουμε τις γνώσεις μας σχετικά με το βιβλίο κατά 1%. Στο κβαντικό βιβλίο αν διαβάσουμε τις σελίδες του, θα δούμε μόνο τυχαία γράμματα. Αν διαβάσουμε τις σελίδες του κβαντικού την μία μετά την άλλη πολύ λίγα πράγματα θα ξέρουμε για το βιβλίο. Ο μόνος τρόπος για να ξέρουμε τι πληροφορία περιέχει το βιβλίο, είναι να κοιτάξουμε και τις 100 σελίδες του ταυτόχρονα. Αυτό συμβαίνει γιατί η πληροφορία δεν είναι τυπωμένη στις σελίδες του αλλά βρίσκεται κωδικοποιημένη στη συσχέτιση μεταξύ των σελίδων του. (Silva, 2018)

1.4.3. ***Κβαντική φάση:*** Στην κυματική, στην συμβολή δύο κυμάτων εκτός από το μέγεθος (πλάτος) των κυμάτων παίζει ρόλο και η φάση στην οποία βρίσκονται. Σε ένα κλασικό σύστημα η μετάβαση από μια φάση σε μια άλλη της ύλης είναι παράμετρος

της θερμοκρασίας (πχ. η μεταβολή του πάγου σε νερό όταν έρθει σε επαφή με κάτι που έχει θερμοκρασία μεγαλύτερη των 0 °C) και/ή της πυκνότητας ή κάποιας άλλης μακροσκοπικής ιδιότητας του υλικού. Οι κβαντικές φάσεις είναι κβαντικές καταστάσεις της ύλης σε συνθήκες απόλυτου μηδέν. Ακόμα και σε αυτές τις συνθήκες ένα κβαντομηχανικό σύστημα έχει διακυμάνσεις και μπορεί να έχουμε μετάβαση σε μια άλλη κβαντική φάση από αυτή που βρισκόταν. Νέες κβαντικές φάσεις ανακαλύπτονται ως προϊόν εφαρμογής των παραμέτρων εκτός της θερμοκρασίας με διαφορετική σειρά πάνω στο κβαντικό σύστημα. Μερικές κβαντικές φάσεις είναι αποτέλεσμα υπέρθεσης πολλών άλλων κβαντικών φάσεων. (Sachdev)

1.5. Αρχιτεκτονική Κβαντικών υπολογιστών

Σε φυσικό επίπεδο τα qubit μπορούν να κατασκευαστούν ως εξής:

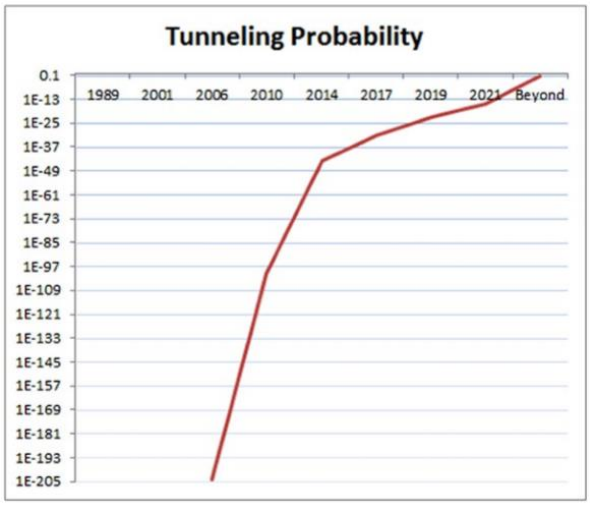
- Με το σπίν ενός σωματιδίου σε ένα μαγνητικό πεδίο όπου το πάνω σημαίνει 0 και το κάτω 1
- Με την πολικότητα ενός μόνο φωτονίου όπου οριζόντια πολικότητα σημαίνει 1 και κάθετη σημαίνει 0.

Τοποθετώντας πολλά qubits μαζί δημιουργούμε κβαντικούς καταχωρητές ή αλλιώς quantum registers. Δημιουργούνται δηλαδή έτσι πύλες οι οποίες επιδρούν πάνω στο qubit και το κάνουν να παίρνει επιθυμητές καταστάσεις. Οι κβαντικές πύλες οφείλουν να ακολουθούν τους κανόνες των κβαντικών διαδικασιών. Συγκεκριμένα αρκετές κβαντικές διαδικασίες πρέπει να είναι αντιστρέψιμες και οι κβαντικές πύλες πρέπει να ακολουθούν τον κανόνα αυτό επίσης. (Yanofsky & Mannucci, 2008). Ο χειρισμός ενός qubit είναι μια ευαίσθητη και σύνθετη διαδικασία. Ένας από τους λόγους που πολλές διαδικασίες πρέπει να είναι αντιστρέψιμες είναι και ότι σε περίπτωση που δεν ήταν, θα είχαμε απώλεια πληροφορίας. Η απώλεια πληροφορίας συνεπάγεται τη δημιουργία θερμότητας, και η θερμότητα με τη σειρά της δημιουργεί τον λεγόμενο «θόρυβο» στους κβαντικούς υπολογιστές. Η ύπαρξη του θορύβου και η λεπτότητα των διαδικασιών που απαιτούνται αποτελεί το κύριο εμπόδιο κατασκευής κβαντικών υπολογιστών με περισσότερα qubits σήμερα.

1.6. Αίτιο ανάγκης για την κατασκευή Κβαντικών Υπολογιστών

Αν και οι κλασικοί υπολογιστές θεωρούνται κάτι σύγχρονο εντούτοις η αρχική τεχνολογία πάνω στην οποία βασίζονται είναι του 1940. Σύμφωνα με τον νόμο του Moore «ο αριθμός των τρανζίστορ ενός πυκνού ολοκληρωμένου κυκλώματος διπλασιάζεται κάθε 2 χρόνια. (Moore's Law, 2020). Την δεκαετία του 1970 τα τρανζίστορ είχαν μέγεθος 10 mm και το 2020 έχουν φτάσει περίπου το μέγεθος των 5nm. Πλέον, επιβεβαιωμένα έχει παρατηρηθεί μια επιβράδυνση του νόμο του Moore. Τα τρανζίστορ δεν μπορούν να μικραίνουν για πάντα και σιγά αλλά σταθερά, φτάνουν στα φυσικά τους όρια με αποτέλεσμα και οι κλασικοί υπολογιστές να φτάνουν στα όρια τους. Το πρόβλημα είναι ότι όσο τα τρανζίστορ φτάνουν το μέγεθος του ατόμου, ξεκινάνε να συμβαίνουν και σε αυτά κβαντικά φαινόμενα, πράγμα που τα καθιστά μη λειτουργικά διότι η συμπεριφορά τους γίνεται απρόβλεπτη. Ένα τέτοιο φαινόμενο είναι το “electron tunneling”. Κατά το φαινόμενο

αυτό ένα σωματίδιο περνάει μέσα από ένα εμπόδιο που υπό κανονικές συνθήκες δεν θα μπορούσε να το περάσει (Hawking, 1998). Αυτό δημιουργεί πολύ μεγάλο πρόβλημα στα τρανζίστορ διότι χάνεται πληροφορία και δημιουργούνται σφάλματα. Για την αντιμετώπιση του προβλήματος αυτού, εκτός από τους κβαντικούς υπολογιστές, έρευνες γίνονται επίσης, στα πεδία της μοριακής ηλεκτρονικής και της οργανικής ηλεκτρονικής.



Πιθανότητα εμφάνισης του φαινομένου quantum tunneling με βάση το μέγεθος των τρανζίστορ (Silva, 2018).

1.7. Τομείς και εφαρμογές κβαντικών υπολογιστών

Οι κβαντικοί υπολογιστές πιστεύεται ότι θα βοηθήσουν σε τομείς όπως η ασφάλεια δεδομένων, η διαχείριση μεγάλων βάσεων δεδομένων, η ιατρική και η τεχνητή νοημοσύνη (κυρίως στον τομέα των τεχνιτών νευρωνικών δικτύων). Μέχρι στιγμής δεν υπάρχει κβαντικός υπολογιστής ικανός να τρέξει κβαντικό αλγόριθμο που θα κάνει κάτι πραγματικά χρήσιμο, παρόλο που τέτοιοι αλγόριθμοι έχουν αναπτυχθεί σε θεωρητικό επίπεδο. Παρόλα αυτά μεγάλες εταιρίες που δραστηριοποιούνται στον τομέα της πληροφορικής, της αυτοκινητοβιομηχανίας (ανάπτυξη αυτόνομων αυτοκινήτων με συνεισφορά των κβαντικών υπολογιστών στην μηχανική μάθηση και στην εύρεση βέλτιστων διαδρομών), εταιρίες που δραστηριοποιούνται στον χρηματοπιστωτικό τομέα (προσομοίωση πιθανών σεναρίων και υπολογισμός του ρίσκου τους) και αμυντικές βιομηχανίες, χρησιμοποιούν κβαντικούς υπολογιστές, με αρκετά χειροπιαστά αποτελέσματα. Δεν φαίνεται πάντως ότι θα αποτελέσουν το αντίπαλο δέος των κλασικών υπολογιστών και θα ότι θα τους αντικαταστήσουν. Πιθανόν θα αποτελέσουν υπολογιστές ειδικού σκοπού. Πιο συγκεκριμένα:

1.7.1. ***Κρυπτογράφηση***: Σχεδόν όλες οι μέθοδοι κρυπτογράφησης σήμερα βασίζονται στην παραγοντοποίηση του γινομένου δύο πολύ μεγάλων πρώτων αριθμών. Για να αποκρυπτογραφηθεί ένα μήνυμα πρέπει να βρεθεί ποιοι 2 πρώτοι αριθμοί συνθέτουν ένα τέτοιον αριθμό. Για παράδειγμα το νούμερο 39 το συνθέτουν οι αριθμοί 3 και 13. Από ένα συγκεκριμένο σημείο και μετά αυτό το πρόβλημα δεν μπορεί να λυθεί από ένα κλασικό υπολογιστή. Το 1994 ο Peter Shor ανέπτυξε ένα τέτοιο αλγόριθμο (Shor,

1999) που θα μπορούσε να επιλύσει το πρόβλημα αυτό σε μερικά λεπτά. (Jaeger, 2018)

1.7.2. ***Λύση βελτιστοποίησης σύνθετων διαδικασιών:*** Η εύρεση βέλτιστης λύσης ανάμεσα σε πολλές, αποτελεί μια δύσκολη διαδικασία. Για παράδειγμα το πρόβλημα του «Κινέζου Ταχυδρόμου» (traveling salesman). Ο ταχυδρόμος πρέπει να επισκεφθεί κάποιους προορισμούς και στόχος του, είναι να περάσει από όλους διανύοντας όμως παράλληλα την μικρότερη δυνατή απόσταση. Με μόλις 15 προορισμούς έχουμε πάνω από 43 δισεκατομμύρια εναλλακτικές διαδρομές. Τέτοιου είδους προβλήματα παρατηρούνται εκτός από τις μεταφορές και στον σχεδιασμό των μικροτσίπ. Οι κβαντικοί υπολογιστές θα επιλύουν τα συγκεκριμένα προβλήματα πολύ πιο γρήγορα από τους κλασικούς υπολογιστές. (Jaeger, 2018)

1.7.3. ***Τεχνητή Νοημοσύνη:*** Τα βαθιά νευρωνικά δίκτυα (deep neural networks) αν συνδυαστούν με τους κβαντικούς υπολογιστές μπορούν να ανιχνεύσουν δομές ανάμεσα σε δεδομένα με μεγάλα επίπεδα θορύβου και κατά συνέπεια να εκπαιδευτούν πολύ πιο γρήγορα.

1.7.4. ***Αναζήτηση σε μεγάλες βάσεις δεδομένων*****:** Κατά την αναζήτηση σε μη ταξινομημένες δομές δεδομένων ένας κλασικός υπολογιστής πρέπει να εξετάσει την κάθε καταχώρηση (data point) ξεχωριστά προκειμένου να την εντοπίσει. Σε μεγάλες βάσεις δεδομένων κάτι τέτοιοι καθιστά τους κλασικούς υπολογιστές μη πρακτικούς. Το 1994 ο Lov Grover δημοσίευσε έναν κβαντικό αλγόριθμο που βρίσκει την καταχώριση στη βάση δεδομένων μετά από $\sqrt{n}$ βήματα όπου n ο αριθμός των καταχωρήσεων. (Grover, 1996)

1.7.5. ***Θεωρητική Χημεία:*** Για την μοντελοποίηση των μορίων κρίσιμο ρόλο παίζει η συμπεριφορά των ηλεκτρονίων κυρίως όσον αφορά την διεμπλοκή (entanglement). Για να κάνουν μια τέτοια μοντελοποίηση οι κλασικοί υπολογιστές χρησιμοποιούν προσομοιωτές (simulators) που προσομοιώνουν την κβαντική συμπεριφορά τους. Για να το κάνουν αυτό μοιραία προβαίνουν σε κάποιες απλουστεύσεις των μοντέλων. Σε έναν κβαντικό υπολογιστή μια τέτοια διαδικασία δεν είναι απαραίτητη καθότι αποτελούν από μόνοι τους ένα «μοντέλο» κβαντικής συμπεριφοράς.

1.8. Quantum Supremacy

Η ανωτερότητα ενός κβαντικού υπολογιστή σε σχέση με έναν κλασικό είναι γνωστή με τον όρο “quantum supremacy”. Με τον όρο αυτό εννοούμε ότι ένας κβαντικός υπολογιστής λύνει ένα πρόβλημα που ακόμα και ο πιο γρήγορος κλασικός υπερ-υπολογιστής θα χρειαζόταν χιλιάδες χρόνια (μη αποδεκτός χρόνος αναμονής). Ενώ αυτό ήταν γνωστό ότι ισχύει σε θεωρητικό επίπεδο για κάποια προβλήματα, η Google το και απέδειξε και πειραματικά το 2019 χρησιμοποιώντας έναν supeconducting κβαντικό υπολογιστή 54 qubit (F, K, R, & al, 2019). Τα αποτελέσματα του εν λόγω πειράματος είναι αμφισβητήσιμα από την IBM (Pednault, Gunnels, Maslov, & Gambetta,

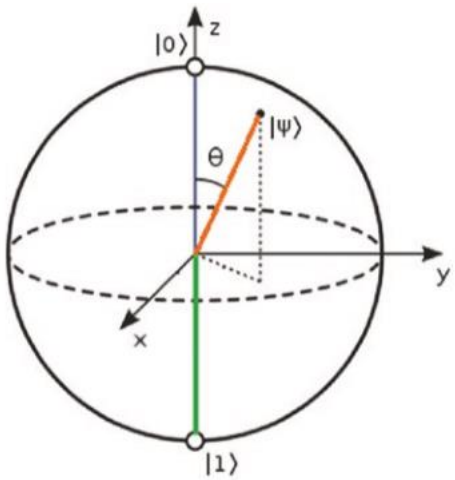
2019) καθώς ναι μεν ο κβαντικός υπολογιστής έλυσε το πρόβλημα σε μερικά δευτερόλεπτα όμως σε ένα κλασικό θα έπαιρνε 2.5 ημέρες , χρόνος ο οποίος είναι αποδεκτός.

2. Αναπαράσταση qubit και λογικών πυλών

Υπάρχουν αρκετοί τρόποι αναπαράστασης του qubit και των λογικών πυλών που ελέγχουν την κατάσταση που βρίσκονται. Τέτοιοι είναι τα kets, αλγεβρικοί πίνακες, γραφικές παραστάσεις, δυσδιάστατα και τρισδιάστατα διανύσματα. Γεωμετρικά τα qubit μπορούν να απεικονιστούν χρησιμοποιώντας ένα σχήμα που ονομάζεται σφαίρα Bloch. Ο κυριότερος συμβολισμός μπορούμε να πούμε ότι είναι τα ket. Η κατάσταση ενός qubit ονομάζεται state. Τα kets ουσιαστικά αναπαριστούν την κυματοσυνάρτηση ενός φυσικού συστήματος και στην περίπτωση της κβαντομηχανικής την πιθανότητα ευρέσεως του συστήματος αυτού εντός ευκλείδειου γεωμετρικού χώρου. Πλέον τα ket από μαθηματικής άποψης είναι πίνακες (Ταμβάκης, 2003).

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Ket 0 και ket 1 με τους αντίστοιχους πίνακες τους. Στην περίπτωση του ket 0 βλέπουμε ότι η πιθανότητα να είναι το qubit σε state 0 όταν μετρηθεί είναι p(0) =1. Αντίστοιχα βλέπουμε και στο ket 1 ότι p(1) = 1.

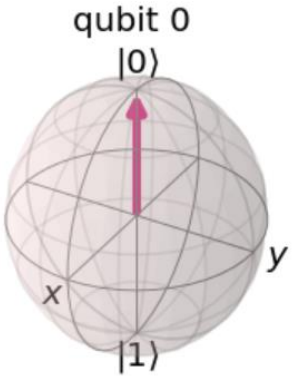


Διάνυσμα σε σφαίρα Bloch που αναπαριστά την κατάσταση ενός qubit

2.1. Λογικές Πύλες Κβαντικού Υπολογιστή

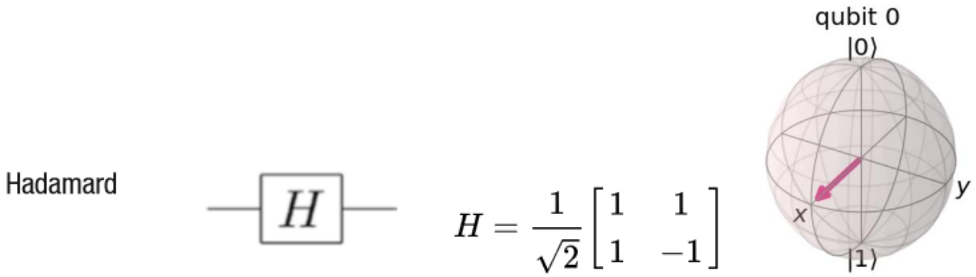
Υπάρχουν αρκετές πύλες που έχουν αναπτυχθεί προκειμένου να ενεργούμε πάνω στα qubit και να καθορίζουμε τα αποτελέσματα τους. Με απλά λόγια αυτό που συμβαίνει σε ένα qubit όταν εφαρμόζεται πάνω του μία πύλη είναι ότι αλλάζει η πιθανότητα του να έρθει 0 ή 1. Οι πιο σημαντικές είναι οι εξής:

Αρχικά ένα qubit βρίσκεται στην κατάσταση state 0 (|0>) πρακτικά αυτό σημαίνει ότι πάντα όταν μετρηθεί το qubit σε αυτή την κατάσταση πάντα το αποτέλεσμα που θα παίρνουμε θα είναι 0.



Αρχική αναπαράσταση qubit με τιμή 0 σε σφαίρα bloch

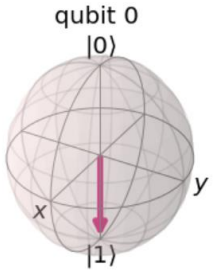
2.1.1. ***Πύλη Hadamard*:** Από τις πιο «χρήσιμες» πύλες που υπάρχουν. Αναπαριστά την περιστροφή του π στον άξονα $(X+Z)/\sqrt{2}$. Είναι δηλαδή η πύλη που βάζει το qubit σε υπέρθεση. Το qubit έχει την ίδια πιθανότητα όταν το μετρήσουμε είτε να είναι 0 είτε 1.



2.1.2. ***Πύλη X(NOT) ή CCNOT*:** Περιστρέφει το qubit 180 μοίρες στον άξονα χ. Είναι πύλη που εφαρμόζεται πάνω σε 2 qubit, το qubit ελέγχου και το qubit στόχο (target). Αν το qubit ελέγχου βρίσκεται σε state 1 τότε το state του qubit στόχου αντιστρέφεται. Αν το qubit ελέγχου βρίσκεται σε state 0 τότε το state του qubit στόχου παραμένει ως έχει.

X (NOT)

$$\mathrm{CNOT} = cX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



2.1.3. ***Πύλη Pauli-Y*:** Εφαρμόζεται πάνω σε ένα qubit. Περιστρέφει ολόκληρη τη σφαίρα Bloch κατά π στον άξονα Y

Y

$$Y = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix}$$

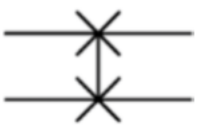
2.1.4. ***Πύλη Pauli-Z*:** Περιστρέφει ολόκληρη στη σφαίρα Bloch κατά π στον άξονα Z

Z

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

2.1.5. ***Πύλη Swap*:** Αλλάζει θέση σε 2 qubits. Πηγαίνει το ένα στη θέση του άλλου.

Swap (S)

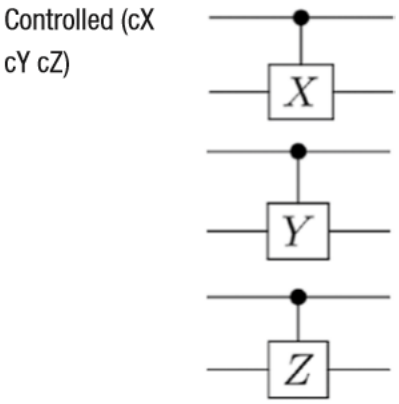


$$\mathrm{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

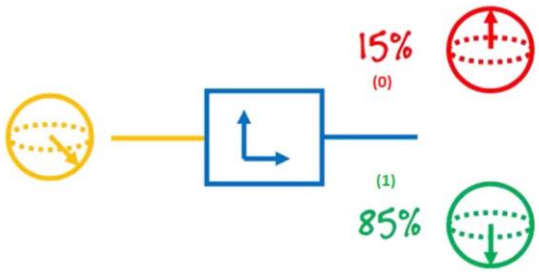
2.1.6. ***Πύλη T*:** Τροποποιεί την φάση της κβαντικής κατάστασης. Μετά την εφαρμογή της σε ένα qubit δεν αλλάζει η πιθανότητα να μετρηθεί |0> ή |1> . Αλλάζει την φάση κατά $\frac{\pi}{4}$. (Sachdev)

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} = \sqrt{S} = \sqrt[4]{Z}$$

2.1.7. ***Controlled (cX cY cZ)*:** Δρα σε 2 ή περισσότερα qubit, όπου το 1 λειτουργεί ως έλεγχος για κάποια λειτουργία. **Μοιάζει κάπως με το λογικό if.**



2.1.8. ***Πύλη μέτρησης*:** παίρνει ένα qubit σε υπέρθεση σαν είσοδο και βγάζει είτε 0 είτε 1. Για την εμφάνιση των μετρήσεων χρησιμοποιούμε κλασικά bits. Μετρώντας ένα μοναδικό qubit, που η κβαντική του κατάσταση αναπαρίσταται από το διάνυσμα a|0> + b|1> = $\begin{bmatrix} a \\ b \end{bmatrix}$, θα δώσει αποτέλεσμα |0> με πιθανότητα $|a|^2$ και |b> με πιθανότητα $|b|^2$.



Παράδειγμα πύλης μέτρησης: Το συγκεκριμένο qubit στην κατάσταση που είναι αν το μετρήσουμε 100 φορές τις 15 θα μας δώσει 0 και τις 85 θα μας δώσει 1.

2.2.Άλλες Κατασκευαστικές Προσεγγίσεις

Αυτός ο τρόπος λειτουργίας ενός κβαντικού υπολογιστή δηλαδή η χρησιμοποίηση κβαντικών πυλών που επιδρούν πάνω στα qubit δεν είναι ο μόνος. Η εταιρία D-Wave έχει κατασκευάσει κβαντικούς υπολογιστές με μερικές χιλιάδες qubits. Εκμεταλλεύεται μια μορφή κβαντικού υπολογισμού που ονομάζεται **quantum annealing ή αδιαβατικός κβαντικός υπολογισμός**. Οι κβαντικοί επεξεργαστές της, έχουν συγκεκριμένη αρχιτεκτονική η οποίο όμως μπορεί να λύσει περιορισμένα προβλήματα βελτιστοποίησης, όπου ο χώρος αναζήτησης είναι διακριτός και υπάρχουν τοπικά ελάχιστα. Δεν μπορούν για παράδειγμα να εκτελέσουν τον αλγόριθμο του Shor. Δεν έχουν λογικές πύλες, δεν μπορούν να επιδράσουν πάνω στα qubits και απλά εκμεταλλεύονται τις ιδιότητες τους. Μια τέτοια ιδιότητα είναι ότι τα qubit τείνουν σε ένα ελάχιστο επίπεδο ενέργειας και η συμπεριφορά τους μπορεί να προβλεφθεί χάρη στο αδιαβατικό θεώρημα. (Silva, 2018). Από την άλλη πλευρά εταιρίες όπως η IBM η Google και η Rigetti χρησιμοποιούν λογικές πύλες στους κβαντικούς τους υπολογιστές για να ελέγξουν τα qubits κάτι το οποίο είναι πολύ πιο δύσκολο και περίπλοκο. Αρκετοί θεωρούν πως οι υπολογιστές που χρησιμοποιούν την τεχνολογία quantum annealing δεν είναι πραγματικοί κβαντικοί υπολογιστές αλλά ένας universal κβαντικός υπολογιστής θα πρέπει να εμπεριέχει την συγκεκριμένη τεχνολογία.

3. Κβαντικοί Προσομοιωτές (Quantum Simulators)

Σχετικά σύγχρονη είναι η ανάπτυξη κλασικών αλγορίθμων για την προσομοίωση κβαντικών συστημάτων. Για να περιγράψουμε αποτελεσματικά ένα κβαντικό σύστημα χρειαζόμαστε ένα εκθετικά αυξανόμενο μεγάλο αριθμό παραμέτρων. Θα ήταν αρκετά χρήσιμο αν καταστάσεις σωματιδίων θα μπορούσαν να υπολογιστούν με την βοήθεια κλασικών υπολογιστών με ένα πιο αποδοτικό τρόπο, τόσο γιατί η πρόσβαση σε πραγματικούς κβαντικούς υπολογιστές είναι αρκετά περιορισμένη και παράλληλα θα μπορούσε να ξεκινήσει η ανάπτυξη κβαντικών αλγορίθμων χωρίς να έχει κατασκευαστεί απαραίτητα το ανάλογο υλικό, αλλά και για να επιβεβαιωθεί ότι οι κβαντικοί επεξεργαστές που κατασκευάζονται λειτουργούν όπως θα έπρεπε. Συνεπώς ήταν αναμενόμενη η ανάπτυξη και δημιουργία κβαντικών προσομοιωτών για όλους τους παραπάνω λόγους. Ένας κβαντικός προσομοιωτής για να λειτουργήσει αποτελεσματικά πρέπει να είναι ικανός να εκτελέσει κατ' ελάχιστο τις παρακάτω λειτουργίες:

3.1. Αναπαράσταση της κατάστασης (state) του κβαντικού υπολογιστή: Επιτυγχάνεται με την αναπαράσταση του κβαντικού καταχωρητή που «εξετάζει» ο προσομοιωτής κάθε φορά. Για την αναπαράσταση ενός κβαντικού καταχωρητή n qubits χρειάζεται ένα κανονικοποιημένο διάνυσμα 2^n μιγαδικών αριθμών

3.2. Τρόπος εκτέλεσης διεργασιών επί του κβαντικού κυκλώματος: Αν και η κατάσταση (state) των qubits αναπαρίσταται με διανύσματα, οι κβαντικές πύλες αναπαρίστανται ως πίνακες της γραμμικής άλγεβρας. Για παράδειγμα αν εφαρμόσουμε την κβαντική πύλη U στο t qubit σε ένα κβαντικό καταχωρητή n qubit τότε θα χρειαζόταν ένας $2^n \times 2^n$ πίνακας. Κάτι τέτοιο εκτός του μεγάλου μεγέθους μνήμης που θα απαιτούσε, θα καθιστούσε και τον προσομοιωτή εξαιρετικά αργό, καθώς η εν λόγω πράξη βασίζεται στο γινόμενο Kronecker. Η εκτέλεση της πράξης αυτής δεδομένων 2 πινάκων $n_1 \times m_1$ και $n_2 \times m_2$ θα είχε χρόνο εκτέλεσης $O(n_1 n_2 m_1 m_2)$. Για να αποφευχθούν όλα αυτά τα ζητήματα ακολουθείται η εξής τακτική. Οι πύλες αποθηκεύονται ως πίνακες 2×2 και είναι μόνο πύλες οι οποίες μπορούν να εφαρμοστούν πάνω σε ένα qubit. Οι πύλες που εφαρμόζονται πάνω σε 2 ή περισσότερα μπορούν να «απλοποιηθούν» στις universal πύλες του ενός qubit. Ένα σετ τέτοιων πυλών είναι οι πύλες Clifford (CNOT, H, S) μαζί με την πύλη T. Στην συνέχεια αφού οι πύλες έχουν «σπάσει» σε πίνακες 2×2 μπορεί να εφαρμοστεί ο αλγόριθμος του σχήματος (Kely, 2018) το εσωτερικό του οποίου μπορεί να τρέξει και παράλληλα. (Kely, 2018)

Input: An n qubit quantum state represented by a column vector $v = (v_1, \ldots v_{2^n})^T$ and a single qubit gate G, represented by a 2×2 matrix, acting on the tth qubit.

```
for i ← 0 to 2^(n−1) do
    a ← the ith integer who's tth bit is 0;
    b ← the ith integer who's tth bit is 1;
    // The following must be
       simultaneously updated
    v_a ← v_a · G_{0,0} + v_b · G_{0,1};
    v_b ← v_b · G_{1,1} + v_a · G_{1,0};
```

3.3. **Μέτρηση των αποτελεσμάτων:** Η μέτρηση των αποτελεσμάτων βασίζεται στον υπολογισμό των πιθανοτήτων που έχει κάθε qubit να βρεθεί σε μια κατάσταση (state). Ενώ η επιλογή ενός αποτελέσματος που βασίζεται στις πιθανότητες αυτές δεν μπορεί να παραλληλοποιηθεί, ο υπολογισμός των πιθανοτήτων μπορεί. Επειδή οι πιθανότητες μπορούν να υπολογιστούν ξεχωριστά από την διαδικασία της μέτρησης, αυτό σημαίνει ότι μπορούν να γίνουν πολλαπλές μετρήσεις χωρίς να χρειάζεται να εφαρμοστούν ξανά όλες οι κβαντικές πύλες από την αρχή. (Kely, 2018)

Για να επιτευχθεί αυτός ο στόχος χρησιμοποιήθηκαν τεχνικές την θεωρίας σχετικά με την κβαντική πληροφορία. Χρησιμοποιώντας matrix product states (MPS) και projected entagled-pair states (PEPS) (Georgescu, Ashhab, & Nori, 2013) μπορεί να γίνει προσομοίωση πολλών qubits σε μια και δύο διαστάσεις. Επιπλέον αυτές οι μέθοδοι μπορούν να συνδυαστούν με τεχνικές Monte Carlo (Monte Carlo Algorithm, 2020) για ακόμα μεγαλύτερη αποδοτικότητα και με παράλληλη χρήση του θεωρήματος Gottesman-Knill. Επίσης από την στιγμή που η αναπαράσταση των κβαντικών καταστάσεων των qubit και η εφαρμογή πυλών πάνω σε αυτά γίνεται με πίνακες της γραμμικής άλγεβρας, έχουν αναπτυχθεί framework της CUDA που αξιοποιούν τις κάρτες γραφικών για να εκτελέσουν προσομοιώσεις συγκεκριμένων κβαντικών υπολογισμών (Babich, Clark, & Joó, 2010) και του OpenCL. Η OpenCL είναι ένα γενικού σκοπού framework για ετερογενή παράλληλο προγραμματισμό που είναι σε θέση να εκμεταλλευτεί κάθε είδους hardware όπως κεντρικούς επεξεργαστές και κάρτες γραφικών διαφορετικών αρχιτεκτονικών. (Kely, 2018)

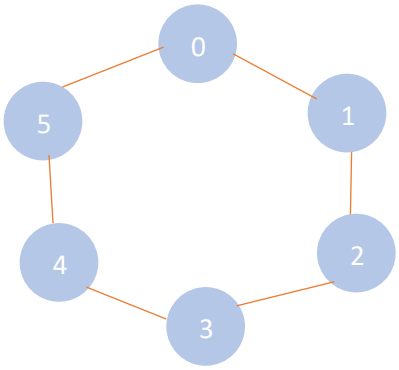
3.4. Θεώρημα Gottesman-Knill

Στον κβαντικό υπολογισμό, το θεώρημα Gottesman-Kill υποστηρίζει ότι τα stabilizer κυκλώματα, κυκλώματα δηλαδή που αποτελούνται αποκλειστικά και μόνο από πύλες του normalizer από το group Pauli ή αλλιώς group Clifford, μπορούν να προσομοιωθούν τέλεια και σε πολυονυμικό χρόνο από έναν propabilistic κλασικό υπολογιστή. Ακόμα και πολύπλοκες καταστάσεις διεμπλοκής qubits, το Clifford group μπορεί να «κατασκευαστεί» κάνοντας χρήση μόνο CNOT, Hadamard και πυλών φάσης. Συνεπάγεται από τα παραπάνω ότι και τα stabilizer κυκλώματα μπορούν να κατασκευαστούν μόνο με αυτές τις πύλες. (Gottesman–Knill theorem, 2020)

3.5. Τοπολογία ενός κβαντικού υπολογιστή

Παρακάτω απεικονίζεται σχηματικό διάγραμμα τοπολογίας κβαντικού υπολογιστή. Τα qubits συμβολίζονται με κύκλο από 0 έως 5. Οι γραμμές συμβολίζουν τα qubits μεταξύ των οποίων μπορούν να εφαρμοστούν κβαντικές πύλες. Για παράδειγμα η κβαντική πύλη CNOT μπορεί να εφαρμοστεί στα qubit 0 και 1 αλλά όχι μεταξύ των qubit 0 και 2. Προκειμένου να εφαρμοστεί μια τέτοια πύλη μεταξύ αυτών των qubit ο compiler της Quil μετατρέπει την πύλη CNOT(0,2) σε λειτουργίες τις οποίες ο κβαντικός υπολογιστής μπορεί να εκτελέσει. Παρόμοιες διαδικασίες

συμβαίνουν και στην Qiskit. Τόσο ο Qasm Simulator όσο και ο QVM έχουν λειτουργίες μέσω των οποίων μπορεί να οριστεί η συνδεσμολογία του εικονικού κβαντικού υπολογιστή.



4. Προγραμματιστικές γλώσσες και framework για κβαντικό υπολογισμό

4.1. Qiskit - IBM

Η Qiskit (Quantum Information Software Kit) είναι μια ανοιχτού κώδικα πλατφόρμα προκειμένου να εργαστεί κάποιος με την κβαντική γλώσσα προγραμματισμού OpenQASM, που χρησιμοποιούν οι κβαντικοί υπολογιστές της IBM (IBM Q Experience). Η Qiskit είναι χωρισμένη σε στοιχεία (elements) κάθε ένα από τα οποία υλοποιεί επί μέρους τμήματα της βιβλιοθήκης. Η προσομοίωση των κβαντικών υπολογισμών βρίσκεται στο «στοιχείο» (element) Aer. Ένα ακόμη σημαντικό στοιχείο είναι το Qiskit Aqua. Εκεί υπάρχουν έτοιμοι, υλοποιημένοι αλγόριθμοι χωρισμένοι σε «πακέτα» ανάλογα το πεδίο (μηχανική μάθηση, χρηματοοικονομικά, χημεία κλπ) πάνω στο οποίο θέλουμε να αναπτύξουμε μια εφαρμογή που θα τρέχει σε κβαντικό υπολογιστή. Η Qiskit είναι διαθέσιμη σε Python, Swift και Javascript (Qiskit Documentation, 2020). Προκειμένου να είναι η σύγκριση όσο το δυνατόν πιο δίκαιη γίνεται, θα εξεταστεί η έκδοση σε Python. Για την εγκατάσταση της Qiskit αλλά και του PyQuil απαιτείται η Python διανομή 3.5+. Προτείνεται η εγκατάσταση τους μέσω της διανομής Anaconda 3 Python που απλοποιεί την διαδικασία καθώς μας δίνει όλα τα dependancies που χρειάζονται προ -εγκατεστημένα. Προτείνεται επίσης η χρήση του jupyter notebook.

4.2. PyQuil – Forest - Rigetti

Το Forest είναι η πλατφόρμα που χρησιμοποιεί η Rigetti για την υλοποίηση κβαντικών αλγορίθμων. Το Forest απαρτίζεται από τα εξής : **PyQuil** που είναι ένα framework - βιβλιοθήκη ανοιχτού κώδικα μέσα στην python. Ένα επίπεδο κάτω από το PyQuil βρίσκεται η **Quil**. Η Quil είναι μια γλώσσα ανοιχτού κώδικα τύπου assembly που χρησιμοποιεί η Rigetti για την ανάπτυξη αλγορίθμων σε κβαντικούς υπολογιστές. Ένα επίπεδο πάνω από το PyQuil βρίσκεται η **Grove**. Η Grove είναι μια βιβλιοθήκη ανοικτού κώδικα της python που περιέχει έτοιμους κβαντικούς

αλγορίθμους (Deutsch-Jozsa, Grover's Search, Bernstein-Vazirani κ.α) και χρησιμοποιεί την προγραμματιστική βιβλιοθήκη της PyQuil. Σημαντικό εργαλείο επίσης είναι και ο QVM. Ο QVM είναι ο κβαντικός προσομοιωτής (simulator) της Rigetti όπου μπορούμε να τρέξουμε δοκιμαστικά τους κβαντικούς αλγορίθμους που έχουμε αναπτύξει τόσο τοπικά όσο και στο cloud (LaRose, 2019). Σε τοπικό επίπεδο υπάρχει περιορισμός ως προς τον αριθμό των qubit λόγο περιορισμένων πόρων όπως είναι φυσικό. Αν έχουμε λογαριασμό στην Rigetti μπορούμε να χρησιμοποιήσουμε απομακρυσμένα τον δικό της simulator με τον αριθμό των διαθέσιμων qubit να αυξάνεται σημαντικά. Επίσης η ύπαρξη λογαριασμού στην Rigetti απλοποιεί την ανάπτυξη του κβαντικού αλγορίθμου στον simulator. Για να τρέξει σε τοπικό επίπεδο απαραίτητη είναι η ανάπτυξη εικονικού server καθώς ο αλγόριθμος πρώτα περνάει από τον quilc compiler και στην συνέχεια από τον QVM για να «τρέξει». Για την εκτέλεση των κβαντικών κυκλωμάτων το PyQuil διαθέτει προσομοιωτή με περίπου 20 qubits

4.3. Άλλες γλώσσες

4.3.1. Q# - QDK - Microsoft

Το QDK είναι ένα ολοκληρωμένο πακέτο λογισμικού από την Microsoft για την ανάπτυξη κβαντικών αλγορίθμων. Το QDK περιέχει την γλώσσα προγραμματισμού Q#. Το συντακτικό της συγκεκριμένης γλώσσας έχει αρκετές διαφορές και δεν μοιάζει με αυτά της Qiskit και της PyQuil. Δεν είναι τόσο «φιλική» γλώσσα για νέους χρήστες. Δεν είναι τόσο πολύ γλώσσα υψηλού επιπέδου υπό την έννοια ότι απαιτείται ακόμα καλύτερη κατανόηση της συμπεριφοράς ενός κβαντικού κυκλώματος από ότι σε άλλες γλώσσες κβαντικού υπολογισμού. Ως αντιστάθμισμα η Q# "δίνει" στον προγραμματιστή ακόμα περισσότερη ελευθερία ως προς την κατασκευή του κυκλώματος και της εν γένη συμπεριφοράς του. Η Microsoft δεν δίνει ακόμα δυνατότητα πρόσβασης σε κάποιο πραγματικό κβαντικό υπολογιστή δεδομένου ότι αυτοί θα βασίζονται σε μια νέα τεχνολογία. Από την άλλη πλευρά στον τοπικό μας υπολογιστή μπορούμε να τρέξουμε τον κβαντικό προσομοιωτή που έχει την δυνατότητα να «τρέξει» κβαντικούς αλγορίθμους μέχρι 30 qubits. Με την συνδρομή στο Azure Cloud ο αριθμός των qubit μπορεί να φτάσει τα 40.

4.3.2. Cirq - Google

Το Cirq είναι ένα framework της python που έχει αναπτυχθεί από την Google και βρίσκεται σε alpha φάση. Αποτελεί μια πολλά υποσχόμενη γλώσσα προγραμματισμού κβαντικών κυκλωμάτων με πολλές δυνατότητες. Ιδιαίτερη βαρύτητα δίνεται στην βελτιστοποίηση των κυκλωμάτων καθώς και στο «υλικό» πάνω στο οποίο πρόκειται να τρέξουν. Προσεγγίζει το ζήτημα της εκτέλεσης των κυκλωμάτων πιο ρεαλιστικά, καθώς η ύπαρξη του θορύβου που χαρακτηρίζει τους κβαντικούς υπολογιστές όπως επίσης και η αρχιτεκτονική τους, καθιστούν συγκεκριμένα κυκλώματα μη εκτελέσιμα πάνω σε συγκεκριμένους κβαντικούς υπολογιστές. Στην παρούσα μορφή της δεν φαίνεται να είναι τόσο «φιλική» σε σύγκριση με την Qiskit και το PyQuil. Η πρόσβαση στις cloud υπηρεσίες της Google που δίνουν την δυνατότητα εκτέλεσης των αλγορίθμων σε προσομοιωτές με περισσότερα qubits και σε πραγματικούς κβαντικούς επεξεργαστές – υπολογιστές είναι περιορισμένη σε μία μικρή ομάδα ατόμων. (Cirq, 2020)

Για τους παραπάνω λόγους η παρούσα διπλωματική εργασία θα επικεντρωθεί στην PyQuil και στην Qiskit δύο framework πιο ώριμα, δοκιμασμένα τα τελευταία χρόνια σε πραγματικές εφαρμογές, με μεγαλύτερη διείσδυση στο ευρύ κοινό και όχι μόνο.

5. Κβαντικοί αλγόριθμοι

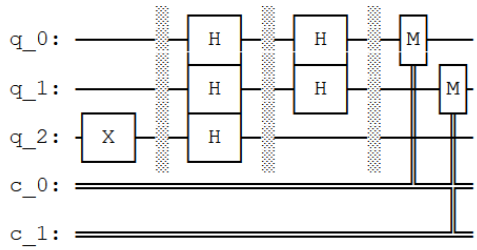
5.1.Deutsch – Jozsa

Έμπνευση για σχεδόν όλους τους κβαντικούς αλγορίθμους, αποτέλεσε ο αλγόριθμος **Deutsch–Jozsa.** Ο αλγόριθμος Deutsch-Jozsa είναι ένα αλγόριθμος «μαύρου κουτιού». Παίρνει σαν είσοδο N bits τα οποία είναι 0 ή 1 και έχει αντίστοιχα N bits σαν έξοδο. Καλούμαστε να εξετάσουμε αν το κουτί μας είναι σταθερό ή ισορροπημένο. Σταθερό είναι το σύστημα αν όλες οι έξοδοι είναι 0 ή όλες είναι 1. Ισορροπημένο αν οι μισές έξοδοι είναι 0 και οι άλλες μισές 1. Ένας **κλασικός υπολογιστής** για να το απαντήσει αν το σύστημα είναι σταθερό ή ισορροπημένο πρέπει να κάνει $\mathbf{2^{n-1}+1}$ περάσματα. Ένας **Κβαντικός υπολογιστής** που θα τρέξει τον αλγόριθμο Deutsch-Jozsa απαντάει στο συγκεκριμένο ερώτημα με **ένα** μόλις πέρασμα.

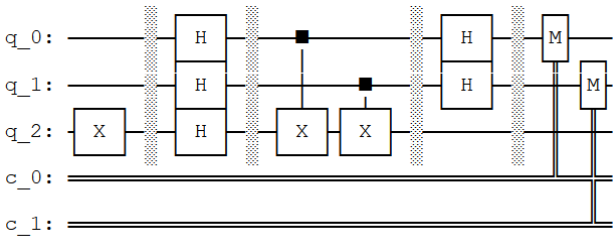
5.1.1.Ο Deutch-Jozsa σε qiskit

```
#συνάρτηση δημιουργίας balanced black box
def balanced_black_box(c):
    c.cx(0,2)
    c.cx(1,2)
    return c

#συνάρτηση δημιουργίας constant black box
def constant_black_box(c):
    return c

############################################################
#### κύκλωμα με balanced black box ########################
############################################################

#δημιουργία κβαντικού κύκλωμα με 3 qubits και 2 κλασικά bits
c = q.QuantumCircuit(3,2)
c.x(2)      #πύλη not στο 3ο qubit

#Πύλες Hadamard πρίν το black box
c.h(0)
c.h(1)
c.h(2)

#εφαρμογή balanced black box στο κύκλωμα
```

```
c = balanced_black_box(c)

#Πύλες Hadamard μετά το black box
c.h(0)
c.h(1)

#πύλες μέτρησης από τα qubit στα κλασικά bit
c.measure([0,1], [0,1])


############################################################
#### κύκλωμα με constant black box ########################
############################################################

c = q.QuantumCircuit(3,2)
c.x(2)      #πύλη not στο 3ο qubit

#Πύλες Hadamard πρίν το black box
c.h(0)
c.h(1)
c.h(2)

#εφαρμογή constant black box στο κύκλωμα
c = constant_black_box(c)

#Πύλες Hadamard μετά το black box
c.h(0)
c.h(1)

#πύλες μέτρησης από τα qubit στα κλασικά bit
c.measure([0,1], [0,1])
```

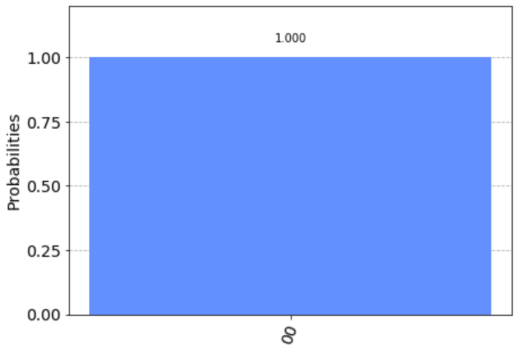


Κύκλωμα Constant Black Box

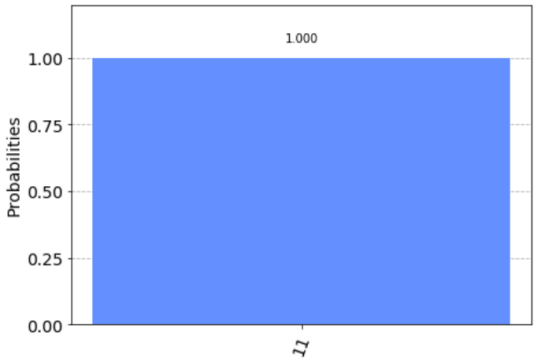


Κύκλωμα Balanced Black Box

Αποτελέσματα στον qasm_simulator

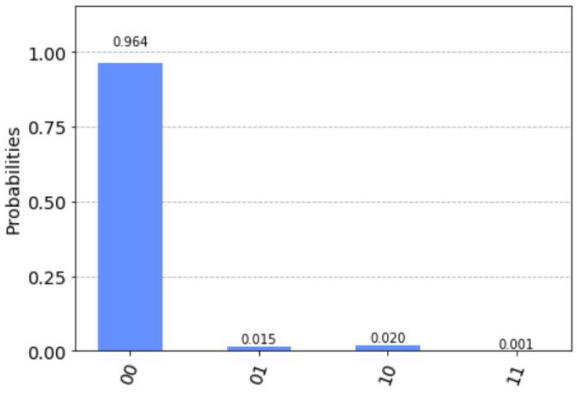


Constant black box

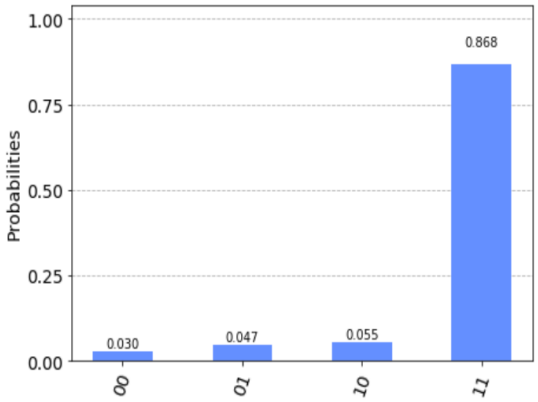


Balanced black box

Τα αποτελέσματα του αλγορίθμου Deutsch-Jozsa αν τον τρέξουμε σε πραγματικούς κβαντικούς υπολογιστές με τη βοήθεια της IBMQ, της qiskit και του jupyter notebook 1500 φορές



Constant black box



Balanced black box

Βλέπουμε ότι σε πραγματικούς κβαντικούς υπολογιστές επιστρέφουν αποτελέσματα πέρα του αναμενομένου. Είναι ο λεγόμενος **θόρυβος** και οφείλεται στο γεγονός ότι οι κβαντικοί υπολογιστές βρίσκονται ακόμα σε πειραματικό στάδιο. Ακόμα όμως και με την περαιτέρω εξέλιξη τους, ένα επίπεδο θορύβου θα υπάρχει πάντα καθώς ένα σύστημα δεν μπορεί ποτέ να είναι απόλυτα απομονωμένο από το περιβάλλον του.

5.1.2.Deutsch Jozsa PyQuil

```
#συνάρτηση δημιουργίας balanced black box
def balanced_black_box(program):
    program += CNOT(0,2)
    program += CNOT(1,2)
    return program

#συνάρτηση δημιουργίας constant black box
def constant_black_box(program):
    return program

############################################################
#### κύκλωμα με balanced black box ########################
############################################################

program += X(2) #πύλη not στο 3ο qubit

#Πύλες Hadamard πρίν το black box
for i in range(3):
    program += H(i)

#εφαρμογή balanced black box στο κύκλωμα
balanced_black_box(program)

#Πύλες Hadamard μετά το black box
for i in range(2):
    program += H(i)

#δέσμευση κλασικών bit στη μνήμη προκειμένου να
ro = program.declare('ro', 'BIT', 2)

#πύλες μέτρησης από τα qubit στα κλασικά bit
program += MEASURE(0, ro[0])
program += MEASURE(1, ro[1])

############################################################
#### κύκλωμα με constant black box ########################
############################################################

program += X(2) #πύλη not στο 3ο qubit
```

```
#Πύλες Hadamard πρίν το black box
for i in range(3):
    program += H(i)

#εφαρμογή constant black box στο κύκλωμα
constant_black_box(program)

#Πύλες Hadamard μετά το black box
for i in range(2):
    program += H(i)

#δέσμευση κλασικών bit στη μνήμη προκειμένου να
ro = program.declare('ro', 'BIT', 2)

#πύλες μέτρησης από τα qubit στα κλασικά bit
program += MEASURE(0, ro[0])
program += MEASURE(1, ro[1])
```

```
X 2
H 0
H 1
H 2
CNOT 0 2
CNOT 1 2
H 0
H 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]
```

Κύκλωμα Balanced black box

Αποτελέσματα Balanced black box: [[1 1]]

```
X 2
H 0
H 1
H 2
H 0
H 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]
```

Κύκλωμα Constant black box

Αποτελέσματα Constant black box: [[0 0]]

5.2.Bernstein-Vazirani

Ο αλγόριθμος Bernstein – Vazirani είναι μια restricted έκδοση του αλγορίθμου Deutch Joza όπου αντί να οριστεί αν το oracle είναι σταθερό ή ισορροπημένο βρίσκει ένα «κριμένο» bit string. Δεδομένης μιας συνάρτησης μαύρου κουτιού *f*, το οποίο παίρνει σαν είσοδο ένα string (συμβολοσειρά) bit και επιστρέφει 0 ή 1 σύμφωνα με:

$f(\{x0,x1,x2,...\}) \rightarrow 0$ ή 1 όπου xn είναι 0 ή 1

Αντί σταθερής ή ισορροπημένης συνάρτησης όπως στο πρόβλημα Deutsch-Josza, εδώ η συνάρτηση είναι εγγυημένο ότι θα επιστρέψει το γινόμενο bit της εισόδου με κάποιο string, S.

Με άλλα λόγια, δεδομένης μιας εισόδου x, $f(x)=s \cdot x \pmod 2$. Αναμένουμε να βρεθεί το S. (Qiskit Documentation, 2020)

5.2.1.Bernstein-Vazirani Qiskit

```
# δήλωση του "κρυφού" string που πρέπει να βρεί ο
# αλγόριθμος και το μέγεθος του
n = 19
s = '0110011111011001111'

#Κύκλωμα μεγέθους n qubit + 1 για το aquila qubit
#και n κλασικά bit για την μέτρηση
bv_circuit = QuantumCircuit(n+1, n)

#ancilla σε state |->
bv_circuit.h(n)
bv_circuit.z(n)
```

```
#Πύλες Hadamard πριν γίνει το  query στο oracle
for i in range(n):
    bv_circuit.h(i)


#Το oracle που δίνει το εσωτερικό γινόμενο
s = s[::-1] # αντιστοφή του string ώστε να ταιριάζει με τρόπο που
#εμφανίζει η qiskit του qubit
for q in range(n):
    if s[q] == '0':
        bv_circuit.i(q)
    else:
        bv_circuit.cx(q, n)

#Πύλες Hadamard μετά το querry στο oracle
for i in range (n):
    bv_circuit.h(i)

#Πύλες μέτρησης στο κύκλωμα
for i in range (n):
    bv_circuit.measure(i,i)
```

5.2.2.Bernestein – Vazirani pyQuil

```
# δήλωση του "κρυφού" string που πρέπει να βρεί ο
# αλγόριθμος και το μέγεθος του
n = 19
s = '0110011111011001111'

#Το ancilla qubit σε state |->
program += H(n)
program += Z(n)

#Πύλες Hadamard πρίν γίνει το querry στο oracle
for i in range(n):
    program += H(i)

#Το oracle που δίνει το εσωτερικό γινόμενο
for q in range(n):
    if s[q] == '0':
        program += I(q)
    else:
```

```
        program += CNOT(q, n)

#Πύλες Hadamard μετά το querry στο oracle
for i in range(n):
    program += H(i)

#ορισμός κλασικών bit για την μέτρηση των αποτελεσμάτων
ro = program.declare('ro', 'BIT', n)

#Πύλες μέτρησης στο κύκλωμα
for i in range(n):
    program += MEASURE(i, ro[i])
```

5.3.Ομοιότητες και Διαφορές Qiskit και PyQuil

Ένα από τα πλεονεκτήματα της PyQuil είναι ότι η ανάπτυξη του κβαντικού κυκλώματος μπορεί να γίνει δυναμικά. Δηλαδή ο αριθμός των κβαντικών bit δεν χρειάζεται να είναι αρχικοποιημένος. Στην Qiskit το κβαντικό κύκλωμα πρέπει να δηλώνεται στην αρχή του προγράμματος ως προς το πόσα κβαντικά και πόσα κλασικά bits θα έχει. Παράδειγμα δήλωσης κυκλώματος με 3 κβαντικά bit και 2 κλασικά:

Qiskit	pyQuil

`#δημιουργία κβαντικού κύκλωμα με 3 qubits και 2 κλασικά bits`
`c = q.QuantumCircuit(3,2)`
`c.x(2)  #πύλη not στο 3ο qubit`
 	
`#Πύλες Hadamard πρίν το black box`
`c.h(0)`
`c.h(1)`
`c.h(2)`
 	
`#εφαρμογή balanced black box στο κύκλωμα`
`c = balanced_black_box(c)`
 	
`#Πύλες Hadamard μετά το black box`
`c.h(0)`
`c.h(1)`
 	
`#πύλες μέτρησης από τα qubit στα κλασικά bit`
`c.measure([0,1], [0,1])`
	`program = Program()`
	`program += X(2)`
	`for i in range(3):`
	`  program +=H(i)`
	
	`for i in range(2):`
	`  program +=H(i)`
	
	`ro = program.declare('ro', 'BIT', 2)`
	
	`program += MEASURE(0, ro[0])`
	`program += MEASURE(1, ro[1])`

Στην PyQuil η δημιουργία των κλασικών bit είναι λίγο πιο πολύπλοκη σε σχέση με την Qiskit καθώς πρέπει ο προγραμματιστής να δεσμεύσει χώρο στην μνήμη. Υπάρχει βέβαια η εναλλακτική στο οι μετρήσεις να γίνουν με την χρήση της συνάρτησης run_and_measure(), μόνο που τότε έχουμε μέτρηση όλων των qubit του κυκλώματος ακόμη και αυτών η μέτρηση των οποίων δεν προσφέρει κάτι στους υπολογισμούς. Με την χρήση της συνάρτησης run() ο χρήστης ορίζει τα κλασικά bits και στην συνέχεια βάζει πύλες μέτρησης από τα qubits που πρέπει να μετρηθούν στα bits.

Παράδειγμα run() και run_and_measure() στην pyQuil

run()

```
ro = program.declare('ro', 'BIT',
 2)

program += MEASURE(0, ro[0])
program += MEASURE(1, ro[1])

print(program)
```

```
X 2
H 0
H 1
H 2
H 0
H 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]
```

```
from pyquil import get_qc

qc = get_qc('3q-qvm')
executable = qc.compile(program)
result = qc.run(executable)
print(result)
```

[[0 0]]

run_and_measure()

```
print(program)
```

```
X 2
H 0
H 1
H 2
H 0
H 1
```

```
from pyquil import get_qc

qc = get_qc('3q-qvm')
results = qc.run_and_measure(program, trial
s=1)
print(results)
```

{0: array([0]), 1: array([0]), 2: array([1])}

6. Quantum Simulators

6.1. Qiskit - Qiskit Aer

Με την χρήση του Aer framework γίνονται κατανοητά τα όρια των κλασικών επεξεργαστών προβάλλοντας σε ποιόν βαθμό μπορούν να «μιμηθούν» κβαντικούς υπολογισμούς. Επίσης το συγκεκριμένο framework δίνει την δυνατότητα ελέγχου πραγματικών κβαντικών υπολογιστών του παρόντος και του βραχυπρόθεσμου μέλλοντος ως προς την ορθότητα της λειτουργίας τους. Αυτό επιτυγχάνεται με το να φτάσει η προσομοίωση στα όρια της, καθώς και προσομοιώνοντας τα αποτελέσματα ρεαλιστικού θορύβου πάνω σε πραγματικούς κβαντικούς υπολογιστές. Εν ολίγοις μπορούμε να πούμε ότι είναι ένα framework ιδανικό για debugging και ανάπτυξη πειραματικών κβαντικών αλγορίθμων. Το Aer δίνει πρόσβαση σε υψηλών επιδόσεων κβαντικούς εξομοιωτές, υλοποιημένους σε C++, όπως επίσης και εργαλεία για την δημιουργία «θορύβου» κατά την διάρκεια εκτέλεσης των κυκλωμάτων πάνω σε πραγματικούς κβαντικούς υπολογιστές. Οι κβαντικοί προσομοιωτές που περιέχει το framework είναι:

6.1.1. ***Qasm Simulator*:** Επιτρέπει την ιδανική, πολλαπλή και με προσομοίωση παρουσίας θορύβου εκτέλεση των κβαντικών κυκλωμάτων της qiskit και επιστρέφει αριθμό εμφανίσεων του αποτελέσματος. Στην ουσία μιμείται μια πραγματική συσκευή (κβαντικό υπολογιστή) και επιστρέφει τις τιμές των κλασικών καταχωρητών του κυκλώματος.

6.1.2. ***State vector Simulator*:** Επιτρέπει την ιδανική και με μία προσπάθεια εκτέλεση κβαντικών κυκλωμάτων και επιστρέφει το State vector της προσομοίωσης

6.1.3. ***Unitary Simulator*:** Επιτρέπει την ιδανική μίας προσπάθειας εκτέλεση κβαντικών κυκλωμάτων και επιστρέφει τον μοναδιαίο πίνακα γραμμικής Άλγεβρας του κυκλώματος. Στο κβαντικό κύκλωμα δεν θα πρέπει να υπάρχουν πύλες μέτρησης.

6.2. PyQuil

6.2.1. ***QVM*:** Ο κβαντικός προσομοιωτής του Forest SDK. Ο QVM εκτελείται τοπικά και έχει φυσικό περιορισμό τα 19 qubits λόγω έλλειψης πόρων του συστήματος. Δίνει την δυνατότητα προσομοίωσης εκτέλεσης του κβαντικού κυκλώματος σε κβαντικό υπολογιστή με παρουσία/απουσία θορύβου καθώς και ορισμού της τοπολογίας του κβαντικού υπολογιστή. Ο QVM είναι ιδανικός για την εκτέλεση κβαντικών κυκλωμάτων μικρού βάθους πράγμα που επιβεβαιώνεται και από τις μετρήσεις που πραγματοποιήθηκαν στην παρούσα διπλωματική εργασία.

6.2.2. ***Wave Function Simulator*:** Επιστρέφει την κυματοσυνάρτηση του κβαντικού συστήματος – κυκλώματος

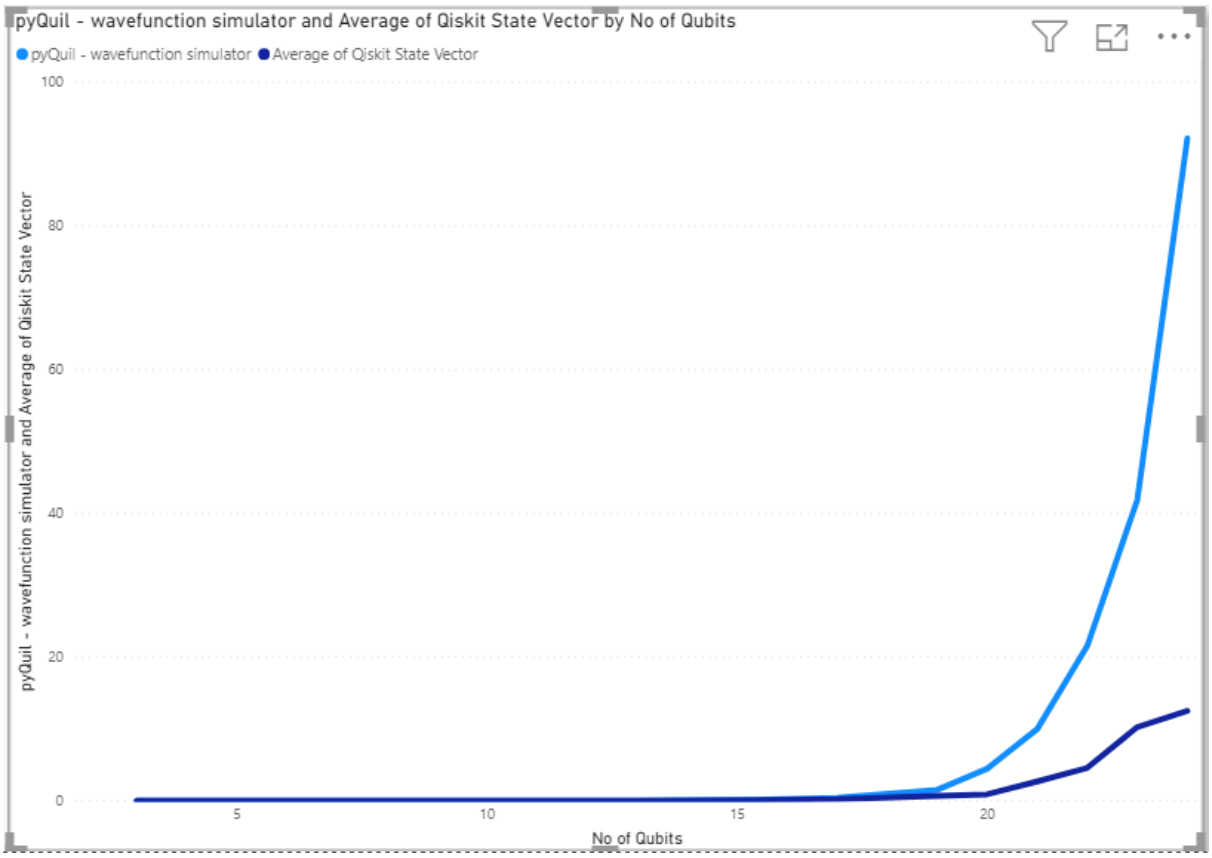
6.2.3. ***Program Unitary*:** Επιστρέφει τον μοναδιαίο πίνακα γραμμικής Άλγεβρας του κβαντικού κυκλώματος

7. Σύγκριση κβαντικών προσομοιωτών

Για την σύγκριση των κβαντικών προσομοιωτών χρησιμοποιήθηκε πρόγραμμα με πύλες Hadamard που εφαρμόζονταν σε κάθε qubit του κυκλώματος. Οι μετρήσεις έγιναν σε laptop: Acer Swift SF314-52 με λειτουργικό σύστημα Windows 10, CPU: intel core i5-8250U και μνήμη Ram: 8 GB.

7.1. State Vector vs Wavefunction Simulator

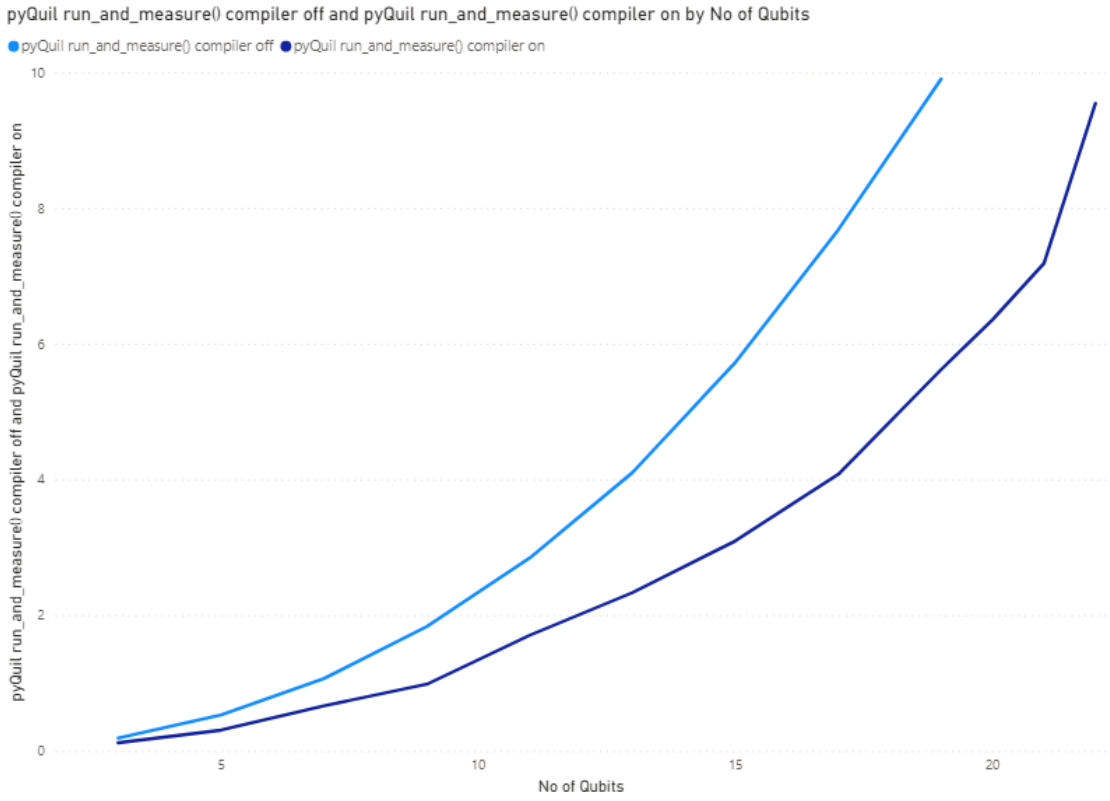
No of Qubits	Depth	Qiskit - State Vector	pyQuil - wavefunction simulator
3	3	0,0045	0,008
5	5	0,0076	0,006
7	7	0,0089	0,0101
9	9	0,0129	0,0117
11	11	0,0214	0,0153
13	13	0,0327	0,029
15	15	0,0676	0,0888
17	17	0,1986	0,4318
19	19	0,6584	1,5163
20	20	0,8982	4,4799
21	21	2,6919	9,9882
22	22	4,589	21,5641
23	23	10,2635	41,8959
24	24	12,505	92,2068



Χρόνος εκτέλεσης του κυκλώματος σε προσομοιωτή της κυματοσυνάρτησης. Μέχρι τα 19 qubits οι χρόνοι εκτέλεσης είναι παρόμοιοι. Από τα 20 qubits και έπειτα η διαφορές είναι πολύ μεγάλες προς όφελος της Qiskit.

7.2.Αντίκτυπο του compiler στον χρόνο εκτέλεσης

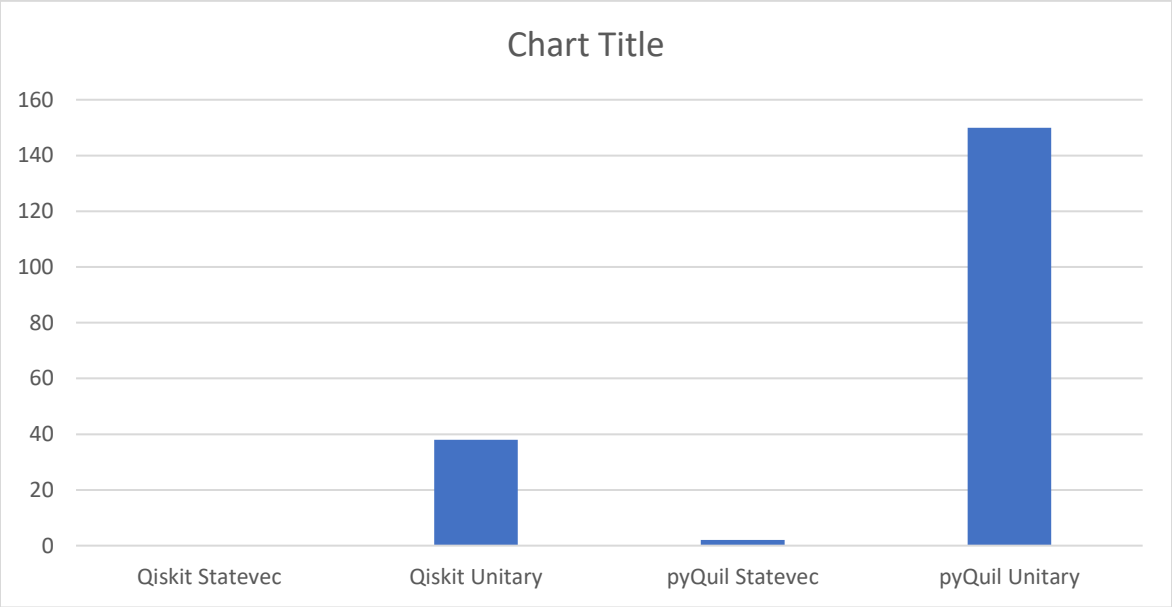
No of Qubits	Depth	pyQuil compiler off	pyQuil compiler on
3	3	0,1979	0,1285
5	5	0,5392	0,3158
7	7	1,0747	0,6737
9	9	1,8416	0,9917
11	11	2,8513	1,713
13	13	4,1129	2,3424
15	15	5,7353	3,1047
17	17	7,6885	4,0849
19	19	9,9115	5,6292
20	20		6,3643
21	21		7,1945
22	22		9,5511



Η προσομοίωση του κυκλώματος χρησιμοποιώντας τον compiler υποδιπλασιάζει τον χρόνο που απαιτείται για την εκτέλεση του. Από τα 20 qubits και πάνω το σύστημα δεν μπόρεσε να ανταποκριθεί στις ανάγκες της προσομοίωσης όταν ο compiler ήταν απενεργοποιημένος.

7.3.Σύγκριση προσομοιωτών wavefunction και unitay

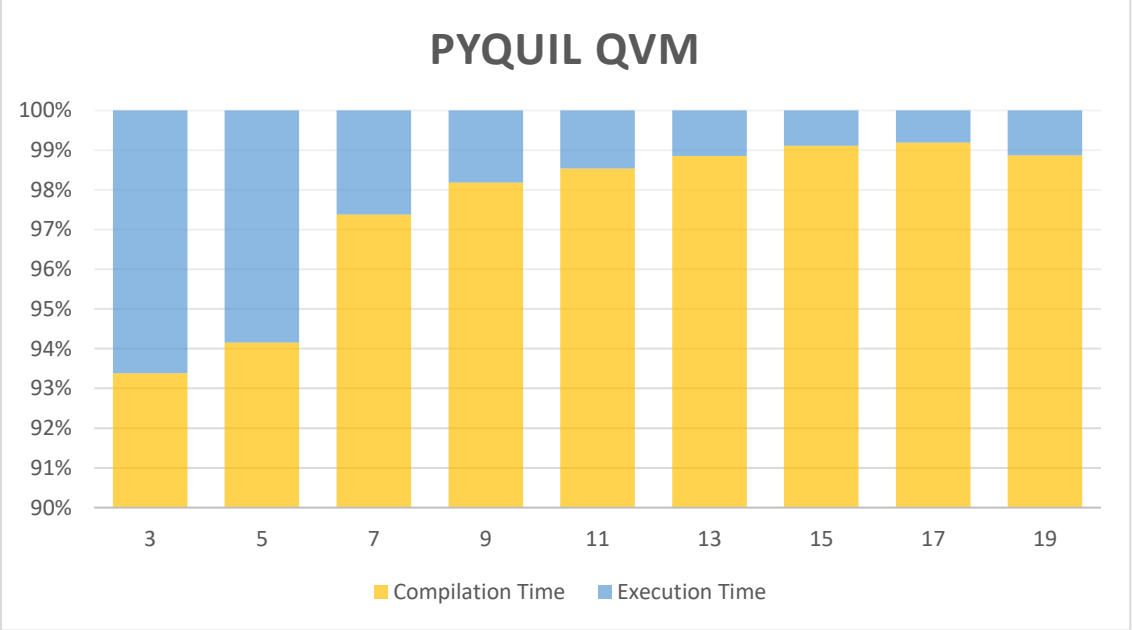
	Qubits	Qiskit Statevec	Qiskit Unitary	pyQuil Statevec	pyQuil Unitary
Bernstain - Vazirani	10	0,0388	37,94	2,0616	149,952



Χρόνος εκτέλεσης του αλγορίθμου Bernstain-Vazirani με 10 qubits σε statevector προσομοιωτή και σε Unitary εξομοιωτή. Ο χρόνος εκτέλεσης είναι πολλαπλάσιος στην περίπτωση του Unitary καθώς αποθηκεύεται όλη η πληροφορία του κυκλώματος.

7.4.Σύγκριση χρόνου compilation και εκτέλεσης

No of Qubits	Compilation Time	Execution Time
3	0,1877	0,0133
5	0,3736	0,0232
7	1,0328	0,0278
9	1,7985	0,0332
11	2,8084	0,0417
13	4,1938	0,0486
15	5,7388	0,0515
17	7,7709	0,0632
19	8,818	0,1011



Ποσοστό επί του συνόλου του χρόνου. Συντριπτικά περισσότερος χρόνος χρειάζεται για το compile του κβαντικού κυκλώματος παρά για την εκτέλεση του στην εικονική μηχανή.

8. Συμπεράσματα

Και τα δύο framework που εξετάστηκαν, Qiskit και PyQuil είναι φιλικά προς τον χρήστη σε ότι αφορά την γενική τους χρήση και την ανάπτυξη γνωστών κβαντικών αλγορίθμων. Και τα δύο framework μέσω των βιβλιοθηκών τους, παρέχουν εργαλεία για τον καθορισμό της τοπολογίας των κβαντικών εικονικών μηχανών αλλά και για την προσομοίωση του θορύβου, τομέα κρίσιμου για την ανάπτυξη πραγματικών κβαντικών υπολογιστών, με την PyQuil να στοχεύει λίγο περισσότερο σε αυτόν τον τομέα.

8.1. Ευχρηστία

Η Qiskit είναι λίγο πιο «φιλική» σε ότι έχει να κάνει με την ευκολία της εγκατάστασης αλλά και σε ότι έχει να κάνει με την απεικόνιση των κυκλωμάτων που αναπτύσσει ο προγραμματιστής (για την PyQuil απαιτείται η εγκατάσταση εξωτερικών εργαλείων για κάτι αντίστοιχο). Το γεγονός ότι δεν χρειάζεται να υπάρχει κάποιος server για το compilation και για να εκτελεστεί το κβαντικό κύκλωμα σε μια εικονική μηχανή την καθιστά απλή στην χρήση της. Επίσης η πλειονότητα των εργαλείων που θα χρειαστεί ο χρήστης για την ανάπτυξη των αλγορίθμων είναι ενσωματωμένα στο framework και αρκεί μια απλή εντολή για την χρήση τους. Από την άλλη πλευρά η PyQuil απαιτεί μια μεγαλύτερη εξοικείωση με το κβαντικό υπολογισμό. Στα θετικά της συγκαταλέγεται το γεγονός ότι είναι από τις πρώτα framework για κβαντικούς υπολογισμούς που χρησιμοποιούν ταυτόχρονα κβαντικούς και κλασικούς καταχωρητές, με αποτέλεσμα την ομαλή εκτέλεση υβριδικών αλγορίθμων. Τέτοια αλγόριθμοι είναι που χρησιμοποιούνται για την ώρα τουλάχιστον για την ανάπτυξη εφαρμογών που κάνουν χρήση κβαντικών υπολογισμών κυρίως στον τομέα της μηχανικής μάθησης. Συνεπώς ως προς αυτό το κομμάτι εύκολα εξάγεται το συμπέρασμα ότι είναι ένα πιο ώριμο framework σε σχέση με το Qiskit. Στα θετικά της επίσης συγκαταλέγεται το γεγονός ότι μπορεί να γίνει εύκολα compile το κύκλωμα και στην συνέχεια, το εκτελέσιμο που θα παραχθεί να παραχθεί σε native Quil να εκτελεστεί με μεγάλη ταχύτητα σε οποιαδήποτε κβαντική μηχανή της Rigetti είτε εικονική είτε πραγματική. Συμπερασματικά το Qiskit φαίνεται σαν ένα framework πιο υψηλού επιπέδου, υπό την έννοια ότι δεν χρειάζεται τόσο βαθιά γνώση του κβαντικού υπολογισμού για την ανάπτυξη ενός κβαντικού κυκλώματος, αλλά το PyQuil όμως μέσω της διαδικασίας που απαιτεί για την εκτέλεση των κυκλωμάτων του (Quil compiler και QVM) ο χρήστης μαθαίνει καλύτερα πως λειτουργεί ο κβαντικός υπολογισμός και οι κβαντικοί εξομοιωτές.

8.2. Ταχύτητα κβαντικών εξομοιωτών

Η σύγκριση των δύο γλωσσών έγινε σε επίπεδο κβαντικών προσομοιωτών καθώς υπήρχε αδυναμία πρόσβασης στις κβαντικές μηχανές της Rigetti. Σύμφωνα με τις μετρήσεις που έγιναν στην παρούσα διπλωματική εργασία ο κβαντικός προσομοιωτής της Qiskit μπορεί να εκτελέσει κυκλώματα με μεγαλύτερο αριθμό qubit απ’ ότι ο αντίστοιχος του PyQuil. Επίσης κατά κανόνα οι προσομοιωτές της Qiskit ήταν πιο γρήγοροι σε σχέση με αυτούς του PyQuil αλλά σε ότι είχε να κάνει με την κβαντικά κυκλώματα με μικρό αριθμό qubit οι διαφορές τους δεν ήταν μεγάλες.

8.3.Προοπτικές

Η Qiskit λόγω της ανοιχτής πολιτικής που ακολουθεί σε ότι έχει να κάνει με την εύκολη πρόσβαση στους κβαντικούς της υπολογιστές και την ευχρηστία της, έχει καταφέρει να προσελκύσει μεγάλο αριθμό ατόμων οι οποίοι εμπλέκονται άμεσα η έμμεσα με την ανάπτυξη της. Αυτό έχει σαν αποτέλεσμα η εύρεση πηγών, οδηγών, κβαντικών αλγορίθμων και πληροφοριών σχετικά με αυτή να είναι πολύ πιο εύκολη απ' ότι τα αντίστοιχα στο PyQuil. Η IBM επίσης επιλέγοντας να αναπτύσσει την Qiskit εκτός από την python σε Swift αλλά πολύ περισσότερο η επιλογή της να την αναπτύσσει σε javascript μπορεί να αποβεί εξαιρετικά χρήσιμη στην ευκολία ανάπτυξης διαδικτυακών και φορητών εφαρμογών στο μέλλον.

9. Βιβλιογραφία

Babich, R., Clark, M., & Joó, B. (2010). Parallelizing the QUDA Library for Multi-GPU Calculations in Lattice Quantum Chromodynamics. *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis on* , (σσ. 1-11). Ανάκτηση από https://academic.microsoft.com/paper/2295191825

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. *Nature, 549*(7671), 195-202. Ανάκτηση από https://academic.microsoft.com/paper/2559394418

Bouwmeester, D., Ekert, A., & Zeilinger, A. (2000). *The Physics of Quantum Information.* Ανάκτηση από https://academic.microsoft.com/paper/101435204

Chiara, M. L., Giuntini, R., Leporini, R., & Sergioli, G. (2014). *Quantum Computation and Logic. How Quantum Computers Have Inspired Logical Investigations.* Cham: Springer.

Cirq. (2020, Novemember). Ανάκτηση από Cirq Documentation: https://cirq.readthedocs.io/en/stable/

F, A., K, A., R, B., & al, e. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 505–510.

Georgescu, I., Ashhab, S., & Nori, F. (2013). Quantum Simulation. *Reviews of Modern Physics*. Ανάκτηση από https://academic.microsoft.com/paper/3037443854

Gottesman–Knill theorem. (2020, 11 10). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Gottesman%E2%80%93Knill_theorem

Grover, L. (1996). *A Fast Quantum Mechanical Algorithm for Database Search.* New York: Association for Computing Machinery.

Hawking, S. (1998). *Το χρονικό του χρόνου.* Κάτοπτρο.

Hey, T., & Walters, P. (2005). *Το νέο κβαντικό σύμπαν.* Αθήνα: Κάτοπτρο.

Jaeger, L. (2018). *The Second Quantum Revolution, From Entanglement to Quantum Computing and Other Super-Technologies.* Cham: Springer.

Kely, A. (2018). *Simulating Quntum Computers Using OpenCL.*

Ladd, T., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., & O'Brien, J. (2010). Quantum Computing. *Nature*. Ανάκτηση από https://academic.microsoft.com/paper/3037853801

LaRose, R. (2019). *Overview and Comparison of Gate Level Quantum Software Platforms.*

Monte Carlo Algorithm. (2020, 10). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Monte_Carlo_algorithm

Moore's Law. (2020, October). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Moore%27s_law

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information.* New York: Cambridge University Press.

Pednault, E., Gunnels, J., Maslov, D., & Gambetta, J. (2019, October 21). *On "Quantum Supremacy"*. Ανάκτηση από IBM: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/

Qiskit Documentation. (2020, October). Ανάκτηση από Qiskit: https://qiskit.org/documentation/

Quantum Computing. (2020, October). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Quantum_computing

Quantum Supersposition. (2020, 10). Ανάκτηση από Wikipedia: https://en.wikipedia.org/wiki/Quantum_superposition

Sachdev, S. (χ.χ.). *Quantum Phase Transitions.* Cambridge, MA, USA: Harvard University.

Shor, P. (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *Siam Review, 41*(2), 303-332. Ανάκτηση από https://academic.microsoft.com/paper/2137147061

Silva, V. (2018). *Practical Quantum Computing for Developers: Programming Quantum Rigs in the Cloud using Python, Quantum Assembly Language and IBM QExperience.* CARY, NC, USA: Apress.

Smith, R. S., Curtis, M. J., & Zeng, W. J. (2016). *A Practical Quantum Instruction Set Architecture.*

Yanofsky, N. S., & Mannucci, M. A. (2008). *Quantum Computing for Computer Scientists.* New York: Cambridge University Press.

Ταμβάκης, Κ. (2003). *Εισαγωγή στην Κβαντομηχανική.* LEADER BOOKS.

10. Παράρτημα (Πηγαίος Κώδικας)

10.1. Bernstein-Vazirani_pyQuil.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

n = 19
s = '0110011111011001111'

program += H(n)
program += Z(n)

#Apply Hadamard before querring the oracle
for i in range(n):
    program += H(i)

#Apply the inner product oracle
for q in range(n):
    if s[q] == '0':
        program += I(q)
    else:
        program += CNOT(q, n)

#Apply Hadamards after querring the oracle
for i in range(n):
    program += H(i)

#setting up measure bits
ro = program.declare('ro', 'BIT', n)

#apply measure gates
for i in range(n):
    program += MEASURE(i, ro[i])

print(program)
```

```
from pyquil import get_qc

qc = get_qc('20q-qvm')
executable = qc.compile(program)
start_time = time.time()
result = qc.run(executable)
end_time = time.time()
print(result)
print(end_time - start_time)
```

10.2. Bernstein-Vazirani-Qiskit.py

```
#!/usr/bin/env python
# coding: utf-8
#initialization
import matplotlib.pyplot as plt
import numpy as np

#importing qiskit
from qiskit import IBMQ, BasicAer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute

#import basic plot tools
from qiskit.visualization import plot_histogram

#import time
import time

# setting the number of qubits as well as the secret string

n = 19
s = '0110011111011001111'

# Bernstein - Vazirani

#We need a circuit with n qubits, plus one ancilla qubit
#Also need n classical bits to write the output to
bv_circuit = QuantumCircuit(n+1, n)

#put ancilla in state |->
bv_circuit.h(n)
bv_circuit.z(n)

#Apply Hadamard gates before quering the oracle
for i in range(n):
    bv_circuit.h(i)

#Apply barrier (for beauty reasons)
bv_circuit.barrier()

#Apply the inner product oracle
s = s[::-1] # reversing string to fit qiskit's qubit ordering
```

```
for q in range(n):
    if s[q] == '0':
        bv_circuit.i(q)
    else:
        bv_circuit.cx(q, n)

#Apply  barrier
bv_circuit.barrier()

#Apply Hadamard Gates after querring the oracle
for i in range (n):
    bv_circuit.h(i)

#Apply Measure Gates
for i in range (n):
    bv_circuit.measure(i,i)

bv_circuit.draw()

#use local simulator
backend = BasicAer.get_backend('qasm_simulator')
shots = 1
start = time.time()
results = execute(bv_circuit, backend=backend, shots = shots).result()
end = time.time()
answer = results.get_counts()

plot_histogram(answer)
print(end-start)
```

10.3. Bernstein-Vazirani_pyQuil_10qubits.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

n = 10
s = '0110011111'

program += H(n)
program += Z(n)

#Apply Hadamard before querring the oracle
for i in range(n):
    program += H(i)

#Apply the inner product oracle
for q in range(n):
    if s[q] == '0':
        program += I(q)
    else:
        program += CNOT(q, n)

#Apply Hadamards after querring the oracle
for i in range(n):
    program += H(i)

#setting up measure bits
ro = program.declare('ro', 'BIT', n)

#apply measure gates
for i in range(n):
    program += MEASURE(i, ro[i])

print(program)

from pyquil import get_qc
```

```
total_time = 0
qc = get_qc('11q-qvm')
for i in range(10):
    start_time = time.time()
    executable = qc.compile(program)
    result = qc.run(executable)
    end_time = time.time()
    total_time += (end_time-start_time)
print(result)
mean_time = total_time/10
print(mean_time)
```

10.4. Bernstein-Vazirani_qiskit_10qubits.py

```
#!/usr/bin/env python
# coding: utf-8

#initialization
import matplotlib.pyplot as plt
import numpy as np

#importing qiskit
from qiskit import IBMQ, BasicAer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute

#import basic plot tools
from qiskit.visualization import plot_histogram

#import time
import time

# setting the number of qubits as well as the secret string

n = 10
s = '0110011111'
# Bernstein - Vazirani

#We need a circuit with n qubits, plus one ancilla qubit
#Also need n classical bits to write the output to
bv_circuit = QuantumCircuit(n+1, n)

#put ancilla in state |->
bv_circuit.h(n)
bv_circuit.z(n)

#Apply Hadamard gates before quering the oracle
for i in range(n):
    bv_circuit.h(i)

#Apply barrier (for beauty reasons)
bv_circuit.barrier()

#Apply the inner product oracle
s = s[::-1] # reversing string to fit qiskit's qubit ordering
```

```
for q in range(n):
    if s[q] == '0':
        bv_circuit.i(q)
    else:
        bv_circuit.cx(q, n)

#Apply  barrier
bv_circuit.barrier()

#Apply Hadamard Gates after querring the oracle
for i in range (n):
    bv_circuit.h(i)

#Apply Measure Gates
for i in range (n):
    bv_circuit.measure(i,i)

bv_circuit.draw()

#use local simulator
backend = BasicAer.get_backend('qasm_simulator')
shots = 1
total_time = 0

for i in range(10):
    start = time.time()
    results = execute(bv_circuit, backend=backend, shots = shots).result()
    end = time.time()
    total_time += (end-start)
    answer = results.get_counts()

mean_time = total_time/10
print(mean_time)
plot_histogram(answer)
```

10.5. Bernstein-Vazirani_pyQuil_10qubits _run_and_measure_compiler_on.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

n = 10
s = '0110011111'

program += H(n)
program += Z(n)

#Apply Hadamard before querring the oracle
for i in range(n):
    program += H(i)

#Apply the inner product oracle
for q in range(n):
    if s[q] == '0':
        program += I(q)
    else:
        program += CNOT(q, n)

#Apply Hadamards after querring the oracle
for i in range(n):
    program += H(i)

print(program)

from pyquil import get_qc

total_time = 0
qc = get_qc('11q-qvm')
start_time = time.time()
result = qc.run_and_measure(program, trials = 1)
end_time = time.time()
total_time = (end_time-start_time)
print(result)
```

```
mean_time = total_time/1
print(mean_time)
```

10.6. Bernstein-Vazirani_pyQuil_unitary.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

n = 10
s = '0110011111'

program += H(n)
program += Z(n)

#Apply Hadamard before querring the oracle
for i in range(n):
    program += H(i)

#Apply the inner product oracle
for q in range(n):
    if s[q] == '0':
        program += I(q)
    else:
        program += CNOT(q, n)

#Apply Hadamards after querring the oracle
for i in range(n):
    program += H(i)

print(program)

from pyquil.simulation.tools import program_unitary

start_time = time.time()
unitary = program_unitary(program, 11)
end_time = time.time()
print(end_time - start_time)
```

10.7. Bernstein-Vazirani_qiskit_unitary.py

```
#!/usr/bin/env python
# coding: utf-8

#initialization
import qiskit
#importing qiskit
unitary_simulator = qiskit.Aer.get_backend('unitary_simulator')
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute
#import time
import time

# setting the number of qubits as well as the secret string
n = 10
s = '0110011111'

# Bernstein - Vazirani

#We need a circuit with n qubits, plus one ancilla qubit
#Also need n classical bits to write the output to
bv_circuit = QuantumCircuit(n+1, n)

#put ancilla in state |->
bv_circuit.h(n)
bv_circuit.z(n)

#Apply Hadamard gates before quering the oracle
for i in range(n):
    bv_circuit.h(i)

#Apply barrier (for beauty reasons)
bv_circuit.barrier()

#Apply the inner product oracle
s = s[::-1] # reversing string to fit qiskit's qubit ordering
for q in range(n):
    if s[q] == '0':
        bv_circuit.i(q)
    else:
        bv_circuit.cx(q, n)

#Apply  barrier
```

```
bv_circuit.barrier()

#Apply Hadamard Gates after querring the oracle
for i in range (n):
    bv_circuit.h(i)

bv_circuit.draw()

#use unitary simulator
start_time = time.time()
result = qiskit.execute(bv_circuit, unitary_simulator).result()
unitary = result.get_unitary(bv_circuit)
end_time = time.time()
print(end_time-start_time)
```

10.8. Deutsch-Jozsa_pyQuil.py

```
#!/usr/bin/env python
# coding: utf-8

# Deutch Jozsa with balanced black box

from pyquil import Program
from pyquil.gates import *

program = Program()

def balanced_black_box(program):
    program += CNOT(0,2)
    program += CNOT(1,2)
    return program

def constant_black_box(program):
    return program

program += X(2)

for i in range(3):
    program += H(i)

print(program)

balanced_black_box(program)

print(program)

for i in range(2):
    program += H(i)

print(program)

ro = program.declare('ro', 'BIT', 2)
program += MEASURE(0, ro[0])
program += MEASURE(1, ro[1])
print(program)

from pyquil import get_qc
import time
```

```
qc = get_qc('3q-qvm')
executable = qc.compile(program)
start_time = time.time()
result = qc.run(executable)
end_time = time.time()
print(result)
print(end_time-start_time)
```

10.9. Deutsch-Jozsa _qiskit.py

```
#!/usr/bin/env python
# coding: utf-8

#import τα απαραίτητα εργαλεία και τους προσομοιωτές που θα χρησιμοποιηθούν
#καλό είνα να χρησιμοποιείται συχνά η συνάρτηση draw() σε ενδιάμεσα βήματα
#ώστε να υπάρχει μια εικόνα του κυκλώματός μετά την εφαρμογή των πυλών
import qiskit as q
from qiskit.visualization import plot_histogram

get_ipython().run_line_magic('matplotlib', 'inline')

qasm_simulator = q.Aer.get_backend('qasm_simulator')
statevec_simulator = q.Aer.get_backend('statevector_simulator')

#αν το black box είναι balanced θα έχουμε σαν αποτέλεσμα 11
#αν το black box είναι constant θα έχουμε σαν αποτέλεσμα 00

#συνάρτηση δημιουργίας balanced black box
def balanced_black_box(c):
    c.cx(0,2)
    c.cx(1,2)
    return c

#συνάρτηση δημιουργίας constant black box
def constant_black_box(c):
    return c

############################################################
#### κύκλωμα με balanced black box ########################
############################################################

#δημιουργία κβαντικού κύκλωμα με 3 qubits και 2 κλασικά bits
c = q.QuantumCircuit(3,2)
c.x(2)      #πύλη not στο 3ο qubit

#Πύλες Hadamard πρίν το black box
c.h(0)
c.h(1)
c.h(2)

#εφαρμογή balanced black box στο κύκλωμα
```

```
c = balanced_black_box(c)

#Πύλες Hadamard μετά το black box
c.h(0)
c.h(1)

#πύλες μέτρησης από τα qubit στα κλασικά bit
c.measure([0,1], [0,1])

#βλέπουμε το κυκλωμά πως έχει διαμορφωθεί
c.draw()

import time

#εκτέλεση κυκλώματος και εμφάνιση αποτελεσμάτων
start_time = time.time()
counts = q.execute(c, backend=qasm_simulator, shots=1).result().get_counts()
end_time = time.time()
print(end_time-start_time)
plot_histogram([counts])

############################################################
#### κύκλωμα με constant black box ########################
############################################################

c = q.QuantumCircuit(3,2)
c.x(2)      #πύλη not στο 3ο qubit

#Πύλες Hadamard πρίν το black box
c.h(0)
c.h(1)
c.h(2)

#εφαρμογή constant black box στο κύκλωμα
c = constant_black_box(c)

#Πύλες Hadamard μετά το black box
c.h(0)
c.h(1)

#πύλες μέτρησης από τα qubit στα κλασικά bit
c.measure([0,1], [0,1])
```

```
#βλέπουμε το κυκλωμά πως έχει διαμορφωθεί
c.draw()

#εκτέλεση κυκλώματος και εμφάνιση αποτελεσμάτων
counts = q.execute(c, backend=qasm_simulator, shots=1000).result().get_counts()
plot_histogram([counts])
```

10.10. n_qubits_n_H_pyQuil.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

qubits = 19
depth = 19

#generate qvm string name
quantum_computer = str(qubits)
backend = "'" + quantum_computer + "q-qvm'"
print(backend)

for i in range(depth):
    for j in range(qubits):
        program += H(j)

from pyquil import get_qc

qc = get_qc('19q-qvm')

total_time_run = 0
total_time_compile = 0
for i in range(10):
    start_time_compile = time.time()
    executable = qc.compile(program)
    start_time_run = time.time()
    result = qc.run(executable)
    end_time = time.time()
    total_time_run += (end_time-start_time_run)
    total_time_compile += (end_time - start_time_compile)

mean_time_run = round(total_time_run/10, 4)
mean_time_compile = round(total_time_compile/10, 4)

print("pyQuil + COMPILE =" + str(mean_time_compile))
print("pyQuil =" + str(mean_time_run))
```

10.11. n_qubits_n_H_Qiskit_Qasm.py

```
#!/usr/bin/env python
# coding: utf-8

#initialization
import matplotlib.pyplot as plt
import numpy as np

#importing qiskit
from qiskit import Aer
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute

#import time
import time

n = 23
depth = 23

exp_circuit = QuantumCircuit(n)

for i in range(depth):
    for j in range(n):
        exp_circuit.h(j)

exp_circuit.draw()

backend = Aer.get_backend('qasm_simulator')

shots = 1
total_time = 0
for i in range(100):
    start_time = time.time()
    result = execute(exp_circuit, backend=backend, shots=shots).result()
    end_time = time.time()
    total_time += (end_time-start_time)

mean_time = round((total_time/100), 4)
print(mean_time)
print(answer)
```

10.12. n_qubits_n_H_pyQuil_wavefunction.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
from pyquil.api import WavefunctionSimulator

import time

program = Program()

qubits = 3
depth = 3

for i in range(depth):
    for j in range(qubits):
        program += H(j)

wf_sim = WavefunctionSimulator()

total_time = 0
for i in range(10):
    start_time = time.time()
    wavefunction = wf_sim.wavefunction(program)
    end_time = time.time()
    total_time += (end_time-start_time)

mean_time = round(total_time/10, 4)

print("pyQuil Wavefunction time = " + str(mean_time))
```

10.13. n_qubits_n_H_Qiskit_statevector.py

```
#!/usr/bin/env python
# coding: utf-8

#initialization
import matplotlib.pyplot as plt
import numpy as np

#importing qiskit
from qiskit import Aer
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute

#import time
import time

n = 5
depth = 5

exp_circuit = QuantumCircuit(n)

for i in range(depth):
    for j in range(n):
        exp_circuit.h(j)

exp_circuit.draw()

backend = Aer.get_backend('statevector_simulator')

shots = 1
total_time = 0
for i in range(100):
    start_time = time.time()
    result = execute(exp_circuit, backend=backend, shots=shots).result()
    end_time = time.time()
    total_time += (end_time-start_time)

mean_time = round((total_time/100), 4)
print(mean_time)
print(answer)
```

10.14. n_qubits_n_H_pyQuil_run_and_measure.py

```
#!/usr/bin/env python
# coding: utf-8

from pyquil import Program
from pyquil.gates import *
import time

program = Program()

qubits = 15
depth = 15

#generate qvm string name
quantum_computer = str(qubits)
backend = "'" + quantum_computer + "q-qvm'"
print(backend)

for i in range(depth):
    for j in range(qubits):
        program += H(j)

from pyquil import get_qc

qc = get_qc('15q-qvm')
total_time = 0
for i in range(10):
    start_time = time.time()
    result = qc.run_and_measure(program, 1)
    end_time = time.time()
    total_time += end_time - start_time
print(round(total_time/10,4))
```