ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΜΙΚΡΟΗΛΕΚΤΡΟΝΙΚΗ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσομοιωτής κβαντικών κυκλωμάτων στο αναπτυξιακό περιβάλλον παράλληλης επεξεργασίας CUDA

Αθανάσιος Δημητρίου Κατσιώτης

Επιβλέπων: **Δημήτριος Γκιζόπουλος,** Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2017

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη προσομοιωτή κβαντικών κυκλωμάτων στο αναπτυξιακό περιβάλλον CUDA

Αθανάσιος Δ. Κατσιώτης
Α.Μ.: MM265

ΕΠΙΒΛΕΠΩΝ: **Δημήτριος Γκιζόπουλος,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ **Όνομα Επώνυμο,** Τίτλος (π.χ Αναπληρωτής Καθηγητής) (εάν υπάρχει)**:**

Ιούνιος 2017

ΠΕΡΙΛΗΨΗ

Οι κβαντικοί υπολογιστές είναι μια ερευνητική περιοχή με αυξανόμενο ενδιαφέρον. Η ιδέα τους είναι θεμελιωδώς διαφορετική από τους κλασικούς ψηφιακούς υπολογιστές και για τον λόγο αυτό απαιτούν νέες τεχνολογίες κατασκευής, νέους τρόπους σχεδιασμού και νέους αλγόριθμους. Η υπάρχουσα υποδομή κβαντικών υπολογιστών είναι μηδαμινή, πράγμα που δυσκολεύει την έρευνα αφού δεν υπάρχουν πραγματικά συστήματα μεγάλης κλίμακας για την υλοποίηση και τον έλεγχο της θεωρίας και των ιδεών.

Αυτή η εργασία έχει σκοπό να βοηθήσει στην επίλυση αυτού του προβλήματος με την δημιουργία ενός προσομοιωτή κβαντικών κυκλωμάτων πάνω στα οποία θα δοκιμάζονται κβαντικοί αλγόριθμοι. Το εργαλείο αυτό θα είναι εύκολα προσβάσιμο και δεν θα απαιτεί πολλούς πόρους για την εκτέλεση του. Θα προσομοιώνει δίνοντας έμφαση στην καλύτερη απόδοση από άποψη χρόνου εκτέλεσης και από άποψη δυνατότητας να λειτουργεί για μεγάλο αριθμού qubits.

Για να επιτευχθεί ο σκοπός αποφασίστηκε να γίνει χρήση καρτών γραφικών (GPU), οι αρχιτεκτονικές των οποίων επιτρέπουν την μαζικά παράλληλη εκτέλεση των πράξεων που απαιτεί η προσομοίωση κβαντικών υπολογιστών με κλασικούς υπολογιστές. Συγκεκριμένα επιλέχθηκε η πλατφόρμα ανάπτυξης CUDA της NVidia που πρόκειται για δημοφιλή και διαδεδομένη επιλογή, και οι κάρτες της υπάρχουν σε πολυάριθμα εργαστήρια.

Το πρώτο βήμα της εργασίας, ήταν η μελέτη της απαιτούμενης θεωρίας. Αυτό περιλαμβάνει την κατανόηση των θεμελιωδών κβαντικών στοιχείων, των κβαντικών πυλών, των κβαντικών υπολογισμών και πως αυτοί υλοποιούνται. Από άποψη των GPU, περιλαμβάνει τις υπάρχουσες αρχιτεκτονικές, τα χαρακτηριστικά τους, πως υλοποιούν τους παράλληλους υπολογισμούς και πως γίνεται σωστή χρήση για εκμετάλλευση των δυνατοτήτων τους στο έπακρο.

Μετά την κατανόηση της θεωρίας, ακολούθησε η θεωρητική σχεδίαση του αλγορίθμου που θα εκτελεί τους υπολογισμούς. Ακόμα, σχεδιάστηκε η μορφή που θα έχει η είσοδος του χρήστη στο εργαλείο, πως τα δεδομένα θα διαβάζονται, θα αποθηκεύονται και θα επεξεργάζονται. Στην συνέχεια έγινε η ανάπτυξη του κώδικα με προσοχή στην τήρηση των παραπάνω. Από κει και πέρα ξεκίνησε πρακτική δοκιμή του κώδικα σε πραγματικές κάρτες, ώστε να γίνουν φανερά τα αποτελέσματα της θεωρητικής σχεδίασης και να ακολουθήσει βελτίωση με πρακτική εφαρμογή. Υλοποιήθηκαν γνωστοί κβαντικοί αλγόριθμοι και κυκλώματα και πάρθηκαν μετρήσεις για την απόδοσή τους με χρήση ειδικών εργαλείων.

Η λειτουργία του προσομοιωτή με λίγα λόγια είναι η εξής. Ο χρήστης εισάγει ένα αρχείο κειμένου το οποίο περιέχει το κύκλωμα προς εκτέλεση γραμμένο με μια γλώσσα περιγραφής που σχεδιάστηκε γι’ αυτό το σκοπό. Γίνεται ανάλυση του κειμένου, μεταφέρονται δεδομένα στην κάρτα γραφικών και εκτελούνται οι απαραίτητες πράξεις. Όταν ολοκληρωθούν όλοι οι υπολογισμοί, επιστρέφουν τα αποτελέσματα από την κάρτα και γίνεται καταγραφή τους σε αρχεία κειμένου ώστε να μπορούν να διαβαστούν από τον χρήστη.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κβαντικοί Υπολογιστές

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: κβαντικά κυκλώματα, παράλληλη επεξεργασία, GPU, CUDA, προσομοίωση

ABSTRACT

Quantum computers is a research field of study with increasing interest. The idea behind them is fundamentally different from the classical digital computers and for that reason they require new manufacturing technologies, new design methods and new algorithms. The existing infrastructure of quantum computers is minimal, and makes the research difficult due to the lack of large scale real systems for implementation and testing of theory and ideas.

The purpose of this master's thesis is to help in the search for a solution to this problem with the creation of a quantum circuit simulator on which they can be tested quantum algorithms. This tool will be easily accessible and it will not require a lot of resources for its execution. It will simulate emphasizing on better performance in terms of execution time and in terms of the ability of performing on a large number of qubits.

To achieve this purpose it was decided to use graphics cards (GPU), the architectures of which allow the massive parallel execution of operations, a fact that serves the enormous amount of operations that quantum computations require. Specifically the NVidia CUDA development platform was chosen because it is popular and widely used option, and the there are numerous laboratories with that type of cards.

The first step of this project was the study of the required theory. This includes the understanding of the fundamental quantum elements, the quantum gates, the quantum calculations and how they are implemented. In terms of the GPU, it includes the existing architectures, their characteristics, how they implement parallel computations and how to be properly used to exploit their potential to the fullest.

After understanding the theory, the next step was to design the theoretical algorithm that performs the calculations. In addition, the form of the user's input to the tool was designed, how the data will be read, stored and processed. Then, the real code was developed with attention to the compliance of the above. After that, the testing of the code started on real cards, so to become apparent the results of the theoretical design and to improve the code by practice. Some known quantum algorithms and circuits were implemented and their performance was analyzed by using special tools.

The operation of the simulator in short, is the following. The user enters a text file which contains the circuit written in a description language designed for this purpose. The tool analyses the text, sends the data to the card and the necessary operations are executed. When all the calculations are completed, the results return from the card and they are written in text files in order the user can read them.

SUBJECT AREA: Quantum Computers

KEYWORDS: quantum circuits, GPU, CUDA, simulation, parallel computation

Αφιερωμένο ή όχι στην γάτα του Schrödinger

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα αρχικά να ευχαριστήσω τον καθηγητή Δημήτριο Γκιζόπουλο που με εμπιστεύτηκε να αναλάβω αυτήν την εργασία, αν και δεν γνώριζε την δουλειά μου σε προπτυχιακό επίπεδο, με θέμα που δεν είναι αρκετά διαδεδομένο στα ελληνικά πανεπιστήμια ώστε να υπάρχει δυνατό θεωρητικό υπόβαθρο.

Ακόμα ευχαριστώ τον κύριο Αρχιμήδη Παυλίδη ο οποίος όντας γνώστης του αντικειμένου με βοήθησε σημαντικά στην κατανόηση της θεωρίας των κβαντικών υπολογιστών και είχε ενεργεί συμμετοχή στις διαδικασίες ελέγχου σωστής λειτουργίας και απασφαλμάτωσης. Οι ιδέες του και η εμπειρία του ήταν πολύτιμες στην σχεδίαση του εργαλείου και στο τελικό αποτέλεσμα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η εργασία έγινε κατά την διάρκεια του χειμερινού εξαμήνου 2016-2017 (Οκτώβρης-Μάρτης) στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών.

1. ΚΒΑΝΤΙΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

Στο πρώτο μέρος παρουσιάζονται κάποια βασικά κομμάτια της θεωρίας των κβαντικών υπολογιστών που βοηθούν στην κατανόηση της λειτουργίας τους, στον τρόπο σχεδίασης κυκλωμάτων και αλγορίθμων και στην μέθοδο που πραγματοποιούνται οι κβαντικοί υπολογισμοί. Στόχος είναι να υπάρξει η θεωρητική βάση για να γίνουν κατανοητές αργότερα οι επιλογές κατά την διαδικασία της σχεδίασης, με βάση τους περιορισμούς και τις δυνατότητες που προκύπτουν από αυτήν την ανάλυση. Αυτά τα απαραίτητα βασικά κομμάτια είναι το κβαντικό bit ή qubit, ο κβαντικός καταχωρητής, οι κβαντικές πύλες, το κβαντικό κύκλωμα και οι κβαντικοί υπολογισμοί. Η παρουσίαση ξεκινάει από τα θεμελιώδη και προχωράει σε συνδυαστικά θέματα ώστε να κτιστεί βήμα βήμα η απαραίτητη γνώση.

1.1 Κβαντικό bit – Qubit

Το bit των ηλεκτρονικών υπολογιστών είναι το θεμελιώδες στοιχείο κλασικής πληροφορίας, μπορεί δηλαδή να αναπαραστήσει τις δύο λογικές καταστάσεις, 0 και 1, που στο φυσικό κόσμο αντιστοιχούν σε δύο διακριτές τιμές του ίδιου φυσικού μεγέθους, όπως δύο τιμές τάσης ή ρεύματος, δύο θέσεις ενός διακόπτη, η ύπαρξη ή η έλλειψη μαγνήτισης σε ένα υλικό κτλ. Η αναπαράσταση αυτή είναι αυστηρά δυαδική και μπορούν να εφαρμοστούν σε αυτήν όλοι οι κανόνες των δυαδικών αριθμών και της άλγεβρας Boole.

Από την άλλη μεριά, το qubit είναι το θεμελιώδες στοιχείο της κβαντικής πληροφορίας. Η κατάσταση ενός qubit είναι ένα διάνυσμα που ανήκει σε έναν δισδιάστατο μιγαδικό διανυσματικό χώρο εσωτερικού γινομένου. Με τον συμβολισμό *ket* $|\mathbf{0}\rangle$ και $|\mathbf{1}\rangle$ αναπαρίστανται οι δύο καταστάσεις της υπολογιστικής βάσης. Η κατάσταση του qubit κάθε στιγμή δίνεται από τον γραμμικό συνδυασμό ή αλλιώς την υπέρθεσή των δύο βασικών καταστάσεων:

$$|\boldsymbol{q}\rangle = \boldsymbol{a}|\mathbf{0}\rangle + \boldsymbol{b}|\mathbf{1}\rangle \tag{1.1}$$

Οι συντελεστές *a* και *b* είναι μιγαδικοί αριθμοί και ονομάζονται πλάτη πιθανότητας, δηλαδή ισχύει ότι $|a|^2$ και $|b|^2$ είναι οι πιθανότητες το qubit να βρεθεί στην αντίστοιχη βασική κατάσταση μετά από μία προβολική μέτρηση στην υπολογιστική βάση, και επομένως:

$$|\boldsymbol{a}|^2 + |\boldsymbol{b}|^2 = \mathbf{1} \tag{1.2}$$

Αυτό σημαίνει ότι αν μετρήσουμε με κάποιο τρόπο την κατάσταση του qubit θα πάρουμε μία από τις βασικές καταστάσεις με την αντίστοιχη πιθανότητα. Το κβαντικό σύστημα του qubit καταρρέει από την υπέρθεση σε μια κατάσταση βάσης όταν μετρείται. Αυτή είναι μία θεμελιωδώς διαφορετική συμπεριφορά του qubit σε σχέση με το κλασικό bit και πρόκειται για αποκλειστικά κβαντικό φαινόμενο.

Οι δύο καταστάσεις βάσης μπορούν να γραφτούν σαν διανύσματα δύο στοιχείων όπως φαίνεται στην 1.3:

$$|\mathbf{0}\rangle = \begin{bmatrix}\mathbf{1}\\\mathbf{0}\end{bmatrix} \text{ και } |\mathbf{1}\rangle = \begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix} \tag{1.3}$$

Συνεπώς η υπέρθεσή τους γράφεται και αυτή σαν διάνυσμα:

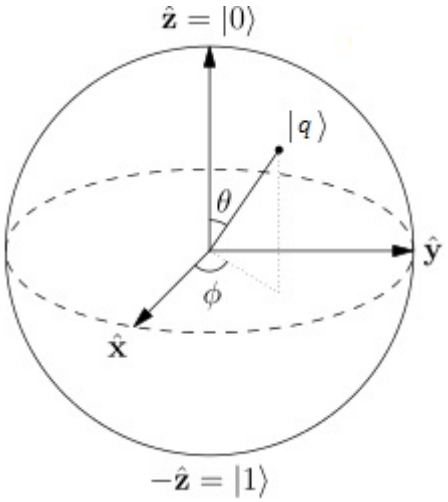
$$|\boldsymbol{q}\rangle = \boldsymbol{a}|\mathbf{0}\rangle + \boldsymbol{b}|\mathbf{1}\rangle = \boldsymbol{a}\begin{bmatrix}\mathbf{1}\\\mathbf{0}\end{bmatrix} + \boldsymbol{b}\begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix} = \begin{bmatrix}\boldsymbol{a}\\\boldsymbol{b}\end{bmatrix} \tag{1.4}$$

Το qubit υπάρχει στον φυσικό κόσμο σαν ένα απλό κβαντικό σύστημα, όπως για παράδειγμα ένα ηλεκτρόνιο μέσα σε ένα μαγνητικό πεδίο, το spin του οποίου μπορεί να βρίσκεται σε δύο βασικές καταστάσεις, spin up $|\uparrow\rangle$ και spin down $|\downarrow\rangle$, ανάλογα με την ευθυγράμμιση του με τις μαγνητικές γραμμές, αλλά μπορεί να βρεθεί σε οποιοδήποτε συνδυασμό αυτών των δύο καταστάσεων.

1.2 Η σφαίρα του Bloch

Δεδομένου ότι τα α και b είναι μιγαδικοί αριθμοί, ένα qubit περιγράφεται σαν ένα διάνυσμα σε πραγματικό χώρο τεσσάρων διαστάσεων. Όμως η μία διάσταση μπορεί να εξαλειφθεί με βάση την ιδιότητα $|\mathrm{a}|^2 + |\mathrm{b}|^2 = 1$. Η υπέρθεση μπορεί να γραφεί στην γενική μορφή:

$$|\boldsymbol{q}\rangle = \boldsymbol{e^{i\gamma}}\left(\boldsymbol{cos}\frac{\boldsymbol{\theta}}{\mathbf{2}}|\mathbf{0}\rangle + \boldsymbol{e^{i\varphi}sin}\frac{\boldsymbol{\theta}}{\mathbf{2}}|\mathbf{1}\rangle\right) \tag{1.5}$$



Σχήμα 1.1 : Η σφαίρα του Bloch

Με βάση την (1.5) και το γεγονός ότι ο παράγοντας $e^{iγ}$ δεν είναι παρατηρήσιμος φυσικά μετά από μία μέτρηση, ένα qubit μπορεί να αναπαρασταθεί σαν ένα διάνυσμα σε χώρο τριών διαστάσεων (παράμετροι γωνίας θ και φ) με τη βοήθεια της σφαίρας του Bloch.

Το διάνυσμα $|q\rangle$ έχει την αρχή του στο κέντρο της σφαίρας και το πέρας του στην επιφάνεια της. Το μήκος του όπως και η ακτίνα της σφαίρας είναι ίσα με 1 (αφού $|a|^2+|b|^2=1$). Η θέση του καθορίζεται από τις γωνίες θ και φ, όπου η θ καθορίζει τις τιμές των πλατών πιθανότητας, α και b, ενώ η φ ονομάζεται γωνία φάσης. Στον βόρειο πόλο της σφαίρας βρίσκεται η κατάσταση $|0\rangle$ ενώ στον νότιο η $|1\rangle$. Ένα κλασικό bit θα μπορούσε να βρεθεί μόνο στους δύο πόλους της σφαίρας, ενώ το qubit μπορεί να είναι ένα οποιοδήποτε σημείο πάνω στην επιφάνειά της εξ' αιτίας της κβαντικής ιδιότητας της υπέρθεσης.

1.3 Κβαντικός καταχωρητής

Συνδυάζοντας τα θεμελιώδη στοιχεία μνήμης μπορούν να δημιουργηθούν μεγαλύτερες δομές αποθήκευσης πληροφορίας. Έτσι στους κλασικούς υπολογιστές με τοποθέτηση bit στην σειρά δημιουργούνται καταχωρητές οι οποίοι μπορούν σε μια χρονική στιγμή να αποθηκεύουν μία εκ των 2^n διαφορετικών καταστάσεων, όπου n ο αριθμός των bits. Με αντίστοιχο τρόπο δημιουργούνται κβαντικοί καταχωρητές, τοποθετώντας στην σειρά ένα πλήθος από qubits. Η πληροφορία που αποθηκεύεται τώρα είναι μεγαλύτερη, όπως γίνεται αντιληπτό από τις διαφορές των bit και qubit. Συγκεκριμένα ένας κβαντικός καταχωρητής n qubits μπορεί να βρίσκεται σε υπέρθεση όλων των 2^n καταστάσεων με κάποιο πλάτος πιθανότητας. Η κατάσταση ενός κβαντικού καταχωρητή δίνεται από το τανυστικό γινόμενο των καταστάσεων των qubits από τα οποία αποτελείται ο καταχωρητής και συμβολίζεται:

$$|\boldsymbol{q}\rangle = |\boldsymbol{q_1}\rangle \otimes |\boldsymbol{q_0}\rangle = |\boldsymbol{q_1}\rangle\,|\boldsymbol{q_0}\rangle = |\boldsymbol{q_1 q_0}\rangle \tag{1.6}$$

Το τανυστικό γινόμενο είναι μια πράξη μεταξύ διανυσμάτων που ορίζεται ως εξής. Για δύο διανύσματα $A = \begin{bmatrix} a_0 \\ \alpha_1 \end{bmatrix}$ και $B = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ ισχύει ότι

$$\boldsymbol{A \otimes B} = \begin{bmatrix} \boldsymbol{a_0} \\ \boldsymbol{\alpha_1} \end{bmatrix} \otimes \begin{bmatrix} \boldsymbol{b_0} \\ \boldsymbol{b_1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha_0 * b_0} \\ \boldsymbol{\alpha_0 * b_1} \\ \boldsymbol{\alpha_1 * b_0} \\ \boldsymbol{\alpha_1 * b_1} \end{bmatrix} \tag{1.7}$$

Κάθε στοιχείο του πρώτου διανύσματος πολλαπλασιάζεται με ολόκληρο το δεύτερο δημιουργώντας ένα νέο διάνυσμα με διάσταση το γινόμενο των διαστάσεων των δύο αρχικών διανυσμάτων. Αυτή η πράξη επεκτείνεται και σε μητρώα ως το γινόμενο Kronecker:

$$\boldsymbol{A \otimes B} = \begin{bmatrix} \boldsymbol{a_{0,0}} & \boldsymbol{a_{0,1}} \\ \boldsymbol{a_{1,0}} & \boldsymbol{a_{1,1}} \end{bmatrix} \otimes \begin{bmatrix} \boldsymbol{b_{0,0}} & \boldsymbol{b_{0,1}} \\ \boldsymbol{b_{1,0}} & \boldsymbol{b_{1,1}} \end{bmatrix}$$

$$= \begin{bmatrix} a_{0,0} & \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{0,1} & \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ a_{1,0} & \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{1,1} & \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix} \quad (1.8)$$

$$= \begin{bmatrix} \alpha_{0,0} * b_{0,0} & \alpha_{0,0} * b_{0,1} & \alpha_{0,1} * b_{0,0} & \alpha_{0,1} * b_{0,1} \\ \alpha_{0,0} * b_{1,0} & \alpha_{0,0} * b_{1,1} & \alpha_{0,1} * b_{1,0} & \alpha_{0,1} * b_{1,1} \\ \alpha_{1,0} * b_{0,0} & \alpha_{1,0} * b_{0,1} & \alpha_{1,1} * b_{0,0} & \alpha_{1,1} * b_{0,1} \\ \alpha_{1,0} * b_{1,0} & \alpha_{1,0} * b_{1,1} & \alpha_{1,1} * b_{1,0} & \alpha_{1,1} * b_{1,1} \end{bmatrix}$$

Έχοντας ορίσει το τανυστικό γινόμενο βλέπουμε πως αυτό εφαρμόζεται στον καταχωρητή. Αν ορίσουμε τα qubits ως

$$|q_1\rangle = a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \text{ και } |q_0\rangle = c|0\rangle + d|1\rangle = \begin{bmatrix} c \\ d \end{bmatrix}$$

παίρνουμε:

$$|q\rangle = |q_1\rangle \otimes |q_0\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)$$

$$= (\alpha * c)|0\rangle \otimes |0\rangle + (a * d)|0\rangle \otimes |1\rangle + (b * c)|1\rangle \otimes |0\rangle + (b * d)|1\rangle \otimes |1\rangle$$

Αν ορίσουμε $(a * c) = p_0$, $(a * d) = p_1$, $(b * c) = p_2$, $(b * d) = p_3$:

$$p_0|00\rangle + p_1|01\rangle + p_2|10\rangle + p_3|11\rangle \quad (1.9)$$

Ή διαφορετικά:

$$|q\rangle = |q_1\rangle \otimes |q_0\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \alpha * c \\ \alpha * d \\ b * c \\ b * d \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \quad (1.10)$$

Επομένως η κατάσταση του κβαντικού καταχωρητή εκφράζεται από τον γραμμικό συνδυασμό ή υπέρθεση τεσσάρων βασικών καταστάσεων. Αυτές οι καταστάσεις ορίζονται ως εξής:

$$|0\rangle \otimes |0\rangle = |0\rangle|0\rangle = |00\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|\mathbf{0}\rangle\otimes|\mathbf{1}\rangle = |\mathbf{0}\rangle|\mathbf{1}\rangle = |\mathbf{01}\rangle = \begin{bmatrix}\mathbf{1}\\\mathbf{0}\end{bmatrix}\otimes\begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\\mathbf{1}\\\mathbf{0}\\\mathbf{0}\end{bmatrix} \quad (1.11)$$

$$|\mathbf{1}\rangle\otimes|\mathbf{0}\rangle = |\mathbf{1}\rangle|\mathbf{0}\rangle = |\mathbf{10}\rangle = \begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix}\otimes\begin{bmatrix}\mathbf{1}\\\mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\\mathbf{0}\\\mathbf{1}\\\mathbf{0}\end{bmatrix}$$

$$|\mathbf{1}\rangle\otimes|\mathbf{1}\rangle = |\mathbf{1}\rangle|\mathbf{1}\rangle = |\mathbf{11}\rangle = \begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix}\otimes\begin{bmatrix}\mathbf{0}\\\mathbf{1}\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\\mathbf{0}\\\mathbf{0}\\\mathbf{1}\end{bmatrix}$$

Από τα παραπάνω μπορούμε να γράψουμε:

$$|\boldsymbol{q}\rangle = |\boldsymbol{q_1}\rangle\otimes|\boldsymbol{q_0}\rangle = \begin{bmatrix}\boldsymbol{a}\\\boldsymbol{b}\end{bmatrix}\otimes\begin{bmatrix}\boldsymbol{c}\\\boldsymbol{d}\end{bmatrix} = \begin{bmatrix}\boldsymbol{p_0}\\\boldsymbol{p_1}\\\boldsymbol{p_2}\\\boldsymbol{p_3}\end{bmatrix} = \boldsymbol{p_0}\begin{bmatrix}\mathbf{1}\\\mathbf{0}\\\mathbf{0}\\\mathbf{0}\end{bmatrix} + \boldsymbol{p_1}\begin{bmatrix}\mathbf{0}\\\mathbf{1}\\\mathbf{0}\\\mathbf{0}\end{bmatrix} + \boldsymbol{p_2}\begin{bmatrix}\mathbf{0}\\\mathbf{0}\\\mathbf{1}\\\mathbf{0}\end{bmatrix} + \boldsymbol{p_3}\begin{bmatrix}\mathbf{0}\\\mathbf{0}\\\mathbf{0}\\\mathbf{1}\end{bmatrix}$$

$$= \boldsymbol{p_0}|\mathbf{00}\rangle + \boldsymbol{p_0}|\mathbf{01}\rangle + \boldsymbol{p_2}|\mathbf{10}\rangle + \boldsymbol{p_3}|\mathbf{11}\rangle \quad (1.12)$$

Τα p_0, p_1, p_2, p_3 είναι τα αντίστοιχα πλάτη πιθανότητας κάθε κατάστασης και ισχύει ότι

$$|\boldsymbol{p_0}|^2 + |\boldsymbol{p_1}|^2 + |\boldsymbol{p_2}|^2 + |\boldsymbol{p_3}|^2 = \mathbf{1} \quad (1.13)$$

Η επέκταση σε μεγαλύτερο αριθμό qubits γίνεται κατά αντίστοιχο τρόπο. Η κατάσταση ενός καταχωρητή n qubits περιγράφεται από:

$$|\boldsymbol{q}\rangle = |\boldsymbol{q_{n-1}}\rangle\otimes|\boldsymbol{q_{n-2}}\rangle\otimes\ldots\otimes|\boldsymbol{q_1}\rangle\otimes|\boldsymbol{q_0}\rangle = |\boldsymbol{q_{n-1}}\ldots\boldsymbol{q_1 q_0}\rangle \quad (1.14)$$

Οι βασικές καταστάσεις είναι 2^n. Όπως και στην περίπτωση του ενός qubit το διάνυσμα κατάστασης μας δείχνει την πιθανότητα να πάρουμε κάποια από τις βασικές καταστάσεις όταν κάνουμε μια μέτρηση.

Με την κατανόηση του κβαντικού καταχωρητή γίνεται αντιληπτή η φύση των κβαντικών υπολογιστών και η διαφορά με τους κλασικούς υπολογιστές. Ο μη ντετερμινισμός που παρουσιάζεται κατά την μέτρηση δεν μας επιτρέπει να γνωρίζουμε την ακριβή κατάσταση ενός καταχωρητή παρά μόνο όταν αυτός μετρηθεί. Επομένως η πληροφορία που αποθηκεύεται είναι μεγάλη καθώς όλες οι καταστάσεις συνυπάρχουν σε μια υπέρθεση που καθορίζεται από τα πλάτη πιθανότητας. Αυτό είναι το μεγάλο πλεονέκτημα των κβαντικών υπολογιστών, δηλαδή η δυνατότητα να δρουν ταυτόχρονα σε όλες τις καταστάσεις σε μια μορφή παράλληλης επεξεργασίας. Ένας καταχωρητής n qubits

μπορεί να επεξεργάζεται 2^n κλασικά bits πληροφορίας. Επίσης αυτό είναι και μειονέκτημα για την συγκεκριμένη εργασία που σκοπός της είναι να προσομοιωθεί αυτή η υπέρθεση από κλασικά στοιχεία μνήμης, επειδή τα n qubits χρειάζονται 2^n bits για να αποθηκευτούν.

1.4 Κβαντικές πύλες

Στα κλασικά ψηφιακά κυκλώματα η πληροφορία μεταδίδεται μέσα από διαύλους σε μορφή τάσης, ρεύματος κτλ. Για να επεξεργαστούμε και να αλλάξουμε αυτήν την πληροφορία την οδηγούμε μέσα από λογικές πύλες, οι οποίες είναι υλοποιημένες από τρανζίστορ πάνω σε υλικά όπως το πυρίτιο, και δρουν με βάση ενός προκαθορισμένου τρόπο που περιγράφεται από έναν πίνακα αληθείας. Έτσι ένα κύκλωμα αποτελείται από την είσοδο που με διαύλους περνάει μέσα από ένα δίκτυο πυλών ώστε να γίνει ένας υπολογισμός και καταλήξει στην έξοδο. Η είσοδος και η έξοδος μπορεί να είναι καταχωρητές ή κάποιο άλλο μέσο.

Στα κβαντικά κυκλώματα η πληροφορία υπόκειται σε επεξεργασία μέσω κβαντικών πυλών. Οι κβαντικές πύλες επιδρούν σε ένα, δύο ή και τρία qubits για να αλλάξουν την κατάσταση τους με έναν προκαθορισμένο τρόπο. Η επίδραση αυτή περιγράφεται από μητρώα 2x2, 4x4 και 8x8 αντιστοίχως. Αυτά τα μητρώα πολλαπλασιάζονται με το διάνυσμα κατάστασης για να προκύψει η κατάσταση μετά από την εφαρμογή της πύλης. Για παράδειγμα σ' ένα qubit στην κατάσταση $|0\rangle$ ενεργεί η πύλη αδράνειας και έχουμε:

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

Δεν μπορεί ένα οποιοδήποτε μητρώο να αντιπροσωπεύει μια κβαντική πύλη. Μια απαραίτητη προϋπόθεση είναι το μητρώο της να είναι ορθομοναδιαίο (U*U=UU*=I) επειδή δεν πρέπει να μεταβάλει το μήκος του διανύσματος κατάστασης και να μην μεταβάλει τις τιμές των εσωτερικών γινομένων μεταξύ δύο διανυσμάτων κατάστασης.

Παρακάτω γίνεται περιγραφή των κβαντικών πυλών που θα προσομοιώνονται με το συγκεκριμένο εργαλείο.

1.5 Κβαντικές πύλες που δρουν σε ένα qubit

1.5.1 Η πύλη αδράνειας

Όπως είδαμε η πύλη αδράνειας δεν επιφέρει αλλαγή στο qubit που ενεργεί. Συμβολίζεται με ***I*** και περιγράφεται από το μητρώο:

$$\boldsymbol{I} = \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\end{bmatrix} \tag{1.15}$$

Αφού η κατάσταση μένει αμετάβλητη έχουμε ότι:

$$\boldsymbol{I}|\boldsymbol{q}\rangle = |\boldsymbol{q}\rangle$$

Ο πίνακας 1.1 δείχνει την δράση της πύλης ***I.*** Είναι το ανάλογο του πίνακα αληθείας.

Πίνακας 1.1 : Η δράση της πύλης αδράνεια

$\|q_{before}\rangle$	$\|q_{after}\rangle$
$\|\mathbf{0}\rangle$	$\|\mathbf{0}\rangle$
$\|\mathbf{1}\rangle$	$\|\mathbf{1}\rangle$

1.5.2 Η πύλη Hadamard

Η πύλη αυτή έχει την δυνατότητα να θέτει ένα qubit σε υπέρθεση των βασικών καταστάσεων με ίση πιθανότητα. Συμβολίζεται με ***H*** και περιγράφεται από το μητρώο:

$$\boldsymbol{H} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix} \tag{1.16}$$

Ας δούμε το αποτέλεσμα στις βασικές καταστάσεις:

$$\boldsymbol{H}|\mathbf{0}\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix}\begin{bmatrix}\mathbf{1}\\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\\ \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\end{bmatrix} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{0}\rangle + \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{1}\rangle \tag{1.17}$$

$$\boldsymbol{H}|\mathbf{1}\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix}\begin{bmatrix}\mathbf{0}\\ \mathbf{1}\end{bmatrix} = \begin{bmatrix}\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\\ -\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\end{bmatrix} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{0}\rangle - \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{1}\rangle \tag{1.18}$$

Και στις δύο περιπτώσεις η πιθανότητα να μετρηθεί μία από τις δύο βασικές καταστάσεις είναι 0,5.

Αν το qubit βρίσκεται σε μία από τις καταστάσεις υπέρθεσης (1.17) και (1.18) και εφαρμοστεί η πύλη Hadamard τότε η κατάσταση που προκύπτει είναι η αντίστοιχη βασική κατάσταση. Οι δεύτερη πύλη δηλαδή απαλείφει την δράση της πρώτης (H^{-1}=H ή H^2=I).

$$\boldsymbol{H}\left(\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{0}\rangle+\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{1}\rangle\right)=\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix}\begin{bmatrix}\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\\ \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\end{bmatrix}=\begin{bmatrix}\mathbf{1}\\ \mathbf{0}\end{bmatrix}=|\mathbf{0}\rangle \quad (1.19)$$

$$\boldsymbol{H}\left(\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{0}\rangle-\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}|\mathbf{1}\rangle\right)=\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix}\begin{bmatrix}\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\\ -\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\end{bmatrix}=\begin{bmatrix}\mathbf{0}\\ \mathbf{1}\end{bmatrix}=|\mathbf{1}\rangle \quad (1.20)$$

Πίνακας 1.2 : Η δράση της πύλης Hadamard

$\lvert q_{before}\rangle$	$\lvert q_{after}\rangle$
$\lvert\mathbf{0}\rangle$	$\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{0}\rangle+\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{1}\rangle$
$\lvert\mathbf{1}\rangle$	$\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{0}\rangle-\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{1}\rangle$
$\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{0}\rangle+\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{1}\rangle$	$\lvert\mathbf{0}\rangle$
$\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{0}\rangle-\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\lvert\mathbf{1}\rangle$	$\lvert\mathbf{1}\rangle$

1.5.3 Η πύλη Pauli X

Η πύλη Pauli X κάνει αντιστροφή μεταξύ των δύο βασικών καταστάσεων. Είναι το ανάλογο της κλασικής πύλης NOT. Αν δράσει πάνω σε qubit που βρίσκεται σε υπέρθεση καταστάσεων ανταλλάσσει τα πλάτη πιθανότητας τους. Ακόμα η λειτουργία της αντιστοιχεί σε περιστροφή του διανύσματος στην σφαίρα του Bloch κατά 180^{o} γύρω από τον άξονα x, όπως φαίνεται στο Σχήμα 1.1. Συμβολίζεται με $\boldsymbol{X}$ και περιγράφεται από το μητρώο:

$$\boldsymbol{X}=\begin{bmatrix}\mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{0}\end{bmatrix} \quad (1.20)$$

Επομένως αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$\boldsymbol{X}(\boldsymbol{a}|\mathbf{0}\rangle+\boldsymbol{b}|\mathbf{1}\rangle)=\begin{bmatrix}\mathbf{0} & \mathbf{1}\\ \mathbf{1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{bmatrix}=\begin{bmatrix}\boldsymbol{b}\\ \boldsymbol{a}\end{bmatrix}=\boldsymbol{b}|\mathbf{0}\rangle+\boldsymbol{a}|\mathbf{1}\rangle \quad (1.21)$$

Πίνακας 1.3 : Η δράση της πύλης Pauli X

$\|q_{before}\rangle$	$\|q_{after}\rangle$
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$
$a\|0\rangle + b\|1\rangle$	$b\|0\rangle + a\|1\rangle$

1.5.4 Η πύλη Pauli Y

Η πύλη αυτή είναι αντίστοιχα η περιστροφή του διανύσματος στην σφαίρα του Bloch κατά 180ο γύρω από τον άξονα y. Συμβολίζεται με ***Y*** και περιγράφεται από το μητρώο:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{1.22}$$

Αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$Y(a|0\rangle + b|1\rangle) = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -ib \\ ia \end{bmatrix} = -ib|0\rangle + ia|1\rangle \tag{1.23}$$

Πίνακας 1.4 : Η δράση της πύλης Pauli Y

$\|q_{before}\rangle$	$\|q_{after}\rangle$
$\|0\rangle$	$i\|1\rangle$
$\|1\rangle$	$-i\|0\rangle$
$a\|0\rangle + b\|1\rangle$	$-ib\|0\rangle + ia\|1\rangle$

1.5.5 Η πύλη Pauli Z

Η πύλη αυτή είναι αντίστοιχα η περιστροφή του διανύσματος στην σφαίρα του Bloch κατά 180ο γύρω από τον άξονα z. Συμβολίζεται με ***Z*** και περιγράφεται από το μητρώο:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.24}$$

Αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$\boldsymbol{Z(a|0\rangle + b|1\rangle)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{-b} \end{bmatrix} = \boldsymbol{a|0\rangle - b|1\rangle} \quad (1.25)$$

Πίνακας 1.5 : Η δράση της πύλης Pauli Z

$\boldsymbol{\|q_{before}\rangle}$	$\boldsymbol{\|q_{after}\rangle}$
$\boldsymbol{\|0\rangle}$	$\boldsymbol{\|0\rangle}$
$\boldsymbol{\|1\rangle}$	$\boldsymbol{-\|1\rangle}$
$\boldsymbol{a\|0\rangle + b\|1\rangle}$	$\boldsymbol{a\|0\rangle - b\|1\rangle}$

1.5.6 Η πύλη μετατόπισης φάσης

Η πύλη αυτή δρα πάνω σε ένα qubit αλλάζοντας μόνο την γωνία φάσης του. Συμβολίζεται με $\boldsymbol{R_{z(\varphi)}}$ και περιγράφεται από το μητρώο:

$$\boldsymbol{R_{z(\varphi)}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{e^{i\varphi}} \end{bmatrix} \quad (1.26)$$

Αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$\boldsymbol{R_{\varphi}(a|0\rangle + b|1\rangle)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{e^{i\varphi}} \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{e^{i\varphi} b} \end{bmatrix} = \boldsymbol{a|0\rangle + e^{i\varphi} b|1\rangle} \quad (1.27)$$

Πίνακας 1.6 : Η δράση της πύλης μετατόπισης φάσης

$\boldsymbol{\|q_{before}\rangle}$	$\boldsymbol{\|q_{after}\rangle}$
$\boldsymbol{\|0\rangle}$	$\boldsymbol{\|0\rangle}$
$\boldsymbol{\|1\rangle}$	$\boldsymbol{e^{i\varphi}\|1\rangle}$
$\boldsymbol{a\|0\rangle + b\|1\rangle}$	$\boldsymbol{a\|0\rangle + e^{i\varphi} b\|1\rangle}$

Παρατήρηση 1: Η πύλη Z είναι η $\boldsymbol{R_{z(\varphi)}}$ για φ = π.

Παρατήρηση 2: Στην συνέχεια και για το συγκεκριμένο εργαλείο, η πύλη μετατόπισης φάσης θα πάρει τρεις μορφές, την $\boldsymbol{R_d}$ κατά την οποία ο χρήστης εισάγει την γωνία περιστροφής σε μοίρες και την $\boldsymbol{R_k}$ κατά την οποία ο χρήστης εισάγει έναν ακέραιο $\boldsymbol{k}$ και η

γωνία περιστροφής υπολογίζεται σε rad(ακτίνια) με βάση τον τύπο $\frac{2\pi}{2^k}$ και την R_z κατά την οποία ο χρήστης εισάγει έναν πραγματικό z από 0 ως 1 που υπολογίζει την γωνία σε ακτίνια με βάση τον τύπο $2\pi z$.

1.5.7 Η πύλη μετατόπισης φάσης S

Είναι μια ειδική περίπτωση της πύλης μετατόπισης φάσης για γωνία 90ο . Συμβολίζεται με S και περιγράφεται από το μητρώο:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{1.28}$$

Αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$S(a|0\rangle + b|1\rangle) = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ ib \end{bmatrix} = a|0\rangle + ib|1\rangle \tag{1.29}$$

Πίνακας 1.7 : Η δράση της πύλης μετατόπισης φάσης S

$\|q_{before}\rangle$	$\|q_{after}\rangle$
$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$i\|1\rangle$
$a\|0\rangle + b\|1\rangle$	$a\|0\rangle + ib\|1\rangle$

1.5.8 Η πύλη μετατόπισης φάσης T

Είναι μια ειδική περίπτωση της πύλης μετατόπισης φάσης για γωνία 45ο . Συμβολίζεται με T και περιγράφεται από το μητρώο:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{1.30}$$

Αν δράσει σε μια υπέρθεση έχει το αποτέλεσμα:

$$T(a|0\rangle + b|1\rangle) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ e^{i\pi/4}b \end{bmatrix} = a|0\rangle + e^{i\pi/4}b|1\rangle \tag{1.31}$$

Πίνακας 1.8 : Η δράση της πύλης μετατόπισης φάσης T

$\|q_{before}\rangle$	$\|q_{after}\rangle$
$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$e^{i\pi/4}\|1\rangle$
$a\|0\rangle + b\|1\rangle$	$a\|0\rangle + e^{i\pi/4}b\|1\rangle$

1.6 Κβαντικές πύλες που δρουν σε δύο qubit

1.6.1 Η πύλη ελεγχόμενου ΟΧΙ - CNOT

Η πύλη δρα σε δύο qubits, το qubit ελέγχου(control) και του qubit στόχο(target). Όταν το πρώτο είναι σε κατάσταση |0⟩ τότε το δεύτερο παραμένει στην κατάσταση που βρίσκεται. Διαφορετικά, αν είναι σε κατάσταση |1⟩ τότε το qubit στόχος αλλάζει κατάσταση όπως αν επιδρούσε πάνω του η πύλη X. Είναι επομένως μια πύλη NOT ή Pauli X που ελέγχεται από ένα qubit. Συμβολίζεται με ***CNOT*** και περιγράφεται από το μητρώο:

$$\boldsymbol{CNOT} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \tag{1.32}$$

Παρατηρούμε ότι το κάτω δεξί μπλοκ του μητρώου είναι το μητρώο της πύλης Pauli X. Με αντίστοιχο τρόπο δημιουργούνται κι άλλες ελεγχόμενες (controlled) πύλες. Αν συμβολίσουμε το qubit ελέγχου με *c* και το στόχο με *t*, η λειτουργία συμβολίζεται με:

$$\boldsymbol{CNOT}|\boldsymbol{c_b t_b}\rangle = |\boldsymbol{c_a t_a}\rangle$$

Πίνακας 1.9 : Η δράση της πύλης ελεγχόμενου ΟΧΙ – CNOT

$\|c_b t_b\rangle$	$\|c_a t_a\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|10\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

Για παράδειγμα αν η αρχική κατάσταση είναι |10⟩:

$$CNOT|10\rangle = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = |11\rangle$$

1.6.2 Η πύλη εναλλαγής – SWAP

Η πύλη αυτή εναλλάσσει τις καταστάσεις των δύο qubits πάνω στα οποία επιδρά. Επομένως δεν επιφέρει αλλαγές στις καταστάσεις |00⟩ και |11⟩. Συμβολίζεται με ***SWAP*** και περιγράφεται από το μητρώο:

$$\boldsymbol{SWAP} = \begin{bmatrix}\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}\end{bmatrix} \tag{1.33}$$

Αν συμβολίσουμε τα qubits με q_1 και q_0, η λειτουργία συμβολίζεται με:

$$\boldsymbol{SWAP}|\boldsymbol{q_{1b}q_{0b}}\rangle = |\boldsymbol{q_{1a}q_{0a}}\rangle$$

Πίνακας 1.10 : Η δράση της πύλης εναλλαγής - SWAP

$\|\boldsymbol{q_{1b}q_{0b}}\rangle$	$\|\boldsymbol{q_{1a}q_{0a}}\rangle$
$\|\mathbf{00}\rangle$	$\|\mathbf{00}\rangle$
$\|\mathbf{01}\rangle$	$\|\mathbf{10}\rangle$
$\|\mathbf{10}\rangle$	$\|\mathbf{01}\rangle$
$\|\mathbf{11}\rangle$	$\|\mathbf{11}\rangle$

Για παράδειγμα αν η αρχική κατάσταση είναι |10⟩:

$$SWAP|10\rangle = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |10\rangle$$

1.6.3 Η πύλη ελεγχόμενης μετατόπισης φάσης

Πρόκειται για την περίπτωση μετατόπισης φάσης όπως ορίστηκε για ένα qubit, χρησιμοποιώντας ένα επιπλέον qubit ελέγχου. Συμβολίζεται με $\boldsymbol{CR_{z(\varphi)}}$ αν και όπως είδαμε σε προηγούμενη παρατήρηση θα χρησιμοποιούμε και τους εναλλακτικούς ορισμούς για τις $\boldsymbol{CR_z}$, $\boldsymbol{CR_k}$ και $\boldsymbol{CR_d}$. Περιγράφεται από το μητρώο:

$$\boldsymbol{CR_{z(\varphi)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\varphi} \end{bmatrix}} \quad (1.34)$$

Πίνακας 1.11 : Η δράση της πύλης ελεγχόμενης μετατόπισης φάσης

$\|c_b t_b\rangle$	$\|c_a t_a\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|10\rangle$
$\|11\rangle$	$e^{i\varphi}\|11\rangle$

1.6.4 Η ελεγχόμενη πύλη Pauli Z

Αναφέρθηκε προηγουμένως ότι ουσιαστικά η πύλη Pauli Z είναι μια πύλη μετατόπισης φάσης 180^o. Εδώ έχουμε την ελεγχόμενη έκδοση. Συμβολίζεται με ***CZ*** και περιγράφεται από το μητρώο:

$$\boldsymbol{CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}} \quad (1.35)$$

Πίνακας 1.12 : Η δράση της πύλης ελεγχόμενης Pauli Z

$\|c_b t_b\rangle$	$\|c_a t_a\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|10\rangle$
$\|11\rangle$	$-\|11\rangle$

1.7 Κβαντικές πύλες που δρουν σε τρία qubit

1.7.1 Η πύλη Toffoli ή διπλά ελεγχόμενου OXI

Σε αυτήν την περίπτωση προστίθεται ένα τρίτο qubit το οποίο χρησιμοποιείται και αυτό για τον έλεγχο της κατάστασης του qubit στόχου. Και τα δύο qubit ελέγχου πρέπει να

βρίσκονται στην κατάσταση |1⟩, ώστε να αλλάξει η κατάσταση του στόχου. Συμβολίζεται με ***Tof*** ή ***CCNOT*** και περιγράφεται από το μητρώο:

$$\boldsymbol{CCNOT} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \tag{1.35}$$

Όπως και στην περίπτωση της CNOT το κάτω δεξιά 2x2 μπλοκ του μητρώου είναι το μητρώο της πύλης NOT ή Pauli X. Αντίστοιχα κατασκευάζονται και όλες οι διπλά ελεγχόμενες πύλες.

Αν συμβολίσουμε τα qubits ελέγχου με c_1, c_0 και το στόχο με t, η λειτουργία συμβολίζεται με:

$$\boldsymbol{CCNOT}|\boldsymbol{c_{1b}c_{0b}t_b}\rangle = |\boldsymbol{c_{1a}c_{0a}t_a}\rangle$$

Πίνακας 1.13 : Η δράση της πύλης Toffoli ή διπλά ελεγχόμενου NOT

$\|\boldsymbol{c_{1b}c_{0b}t_b}\rangle$	$\|\boldsymbol{c_{1a}c_{0a}t_a}\rangle$
$\|\mathbf{000}\rangle$	$\|\mathbf{000}\rangle$
$\|\mathbf{001}\rangle$	$\|\mathbf{001}\rangle$
$\|\mathbf{010}\rangle$	$\|\mathbf{010}\rangle$
$\|\mathbf{011}\rangle$	$\|\mathbf{011}\rangle$
$\|\mathbf{100}\rangle$	$\|\mathbf{100}\rangle$
$\|\mathbf{101}\rangle$	$\|\mathbf{101}\rangle$
$\|\mathbf{110}\rangle$	$\|\mathbf{111}\rangle$
$\|\mathbf{111}\rangle$	$\|\mathbf{110}\rangle$

Για παράδειγμα αν η αρχική κατάσταση είναι |110⟩:

$$CCNOT|110\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |111\rangle$$

1.7.2 Η πύλη Fredkin ή ελεγχόμενης εναλλαγής

Το επιπλέον qubit χρησιμοποιείται σαν έλεγχος για την λειτουργία της εναλλαγής, δηλαδή όταν είναι στην κατάσταση |1⟩, τα δύο qubits στόχοι εναλλάσσουν καταστάσεις. Συμβολίζεται με ***Fred*** ή ***CSWAP*** και περιγράφεται από το μητρώο:

$$\boldsymbol{CSWAP} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad (1.36)$$

Αν συμβολίσουμε το qubit ελέγχου με *c* και τους στόχους με t_1, t_0 η λειτουργία συμβολίζεται με:

$$\boldsymbol{CSWAP}|\boldsymbol{c_{1b}t_{1b}t_{0b}}\rangle = |\boldsymbol{c_{1a}t_{1a}t_{0a}}\rangle$$

Πίνακας 1.14 : Η δράση της πύλης Fredkin ή ελεγχόμενης εναλλαγής

$\|\boldsymbol{c_{1b}t_{1b}t_{0b}}\rangle$	$\|\boldsymbol{c_{1a}t_{1a}t_{0a}}\rangle$
$\|\mathbf{000}\rangle$	$\|\mathbf{000}\rangle$
$\|\mathbf{001}\rangle$	$\|\mathbf{001}\rangle$
$\|\mathbf{010}\rangle$	$\|\mathbf{010}\rangle$
$\|\mathbf{011}\rangle$	$\|\mathbf{011}\rangle$
$\|\mathbf{100}\rangle$	$\|\mathbf{100}\rangle$
$\|\mathbf{101}\rangle$	$\|\mathbf{110}\rangle$
$\|\mathbf{110}\rangle$	$\|\mathbf{101}\rangle$
$\|\mathbf{111}\rangle$	$\|\mathbf{111}\rangle$

Για παράδειγμα αν η αρχική κατάσταση είναι |110⟩:

$$CWAP|110\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |101\rangle$$

1.7.3 Η πύλη διπλά ελεγχόμενης μετατόπισης φάσης

Η περίπτωση της αλλαγής φάσης με δύο qubit ελέγχου. Συμβολίζεται με $\boldsymbol{CCR_{z(\varphi)}}$ (εδώ θα χρησιμοποιηθούν οι συμβολισμοί $\boldsymbol{CCR_z}$, $\boldsymbol{CCR_k}$ και $\boldsymbol{CCR_d}$. Περιγράφεται από το μητρώο:

$$\boldsymbol{CCR}_{\boldsymbol{z(\varphi)}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{e^{i\varphi}} \end{bmatrix} \quad (1.37)$$

Πίνακας 1.15 : Η δράση της πύλης διπλά ελεγχόμενης μετατόπισης φάσης

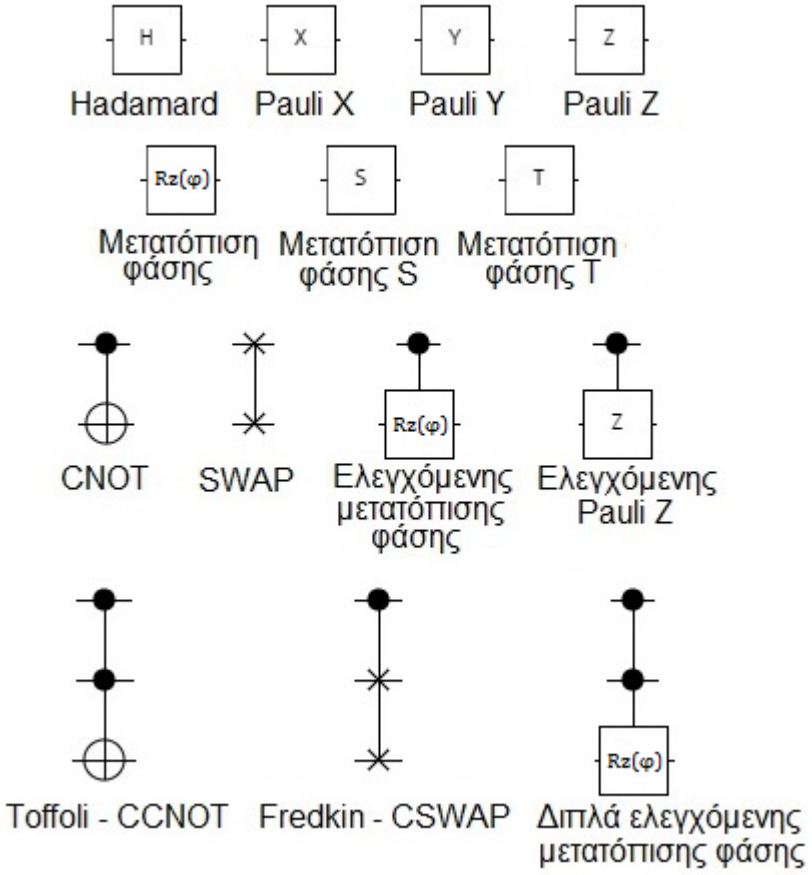
$\|c_{1b}t_{1b}t_{0b}\rangle$	$\|c_{1a}t_{1a}t_{0a}\rangle$
$\|\mathbf{000}\rangle$	$\|\mathbf{000}\rangle$
$\|\mathbf{001}\rangle$	$\|\mathbf{001}\rangle$
$\|\mathbf{010}\rangle$	$\|\mathbf{010}\rangle$
$\|\mathbf{011}\rangle$	$\|\mathbf{011}\rangle$
$\|\mathbf{100}\rangle$	$\|\mathbf{100}\rangle$
$\|\mathbf{101}\rangle$	$\|\mathbf{101}\rangle$
$\|\mathbf{110}\rangle$	$\|\mathbf{110}\rangle$
$\|\mathbf{111}\rangle$	$e^{i\varphi}\|\mathbf{111}\rangle$

1.8 Κβαντικά κυκλώματα και υπολογισμοί

Αφού έγινε η περιγραφή των εννοιών των qubits, του κβαντικού καταχωρητή και των λειτουργιών όλων των κβαντικών πυλών που θα προσομοιώνονται, σειρά έχει να αναλυθεί ο τρόπος σύνθεσης των κβαντικών κυκλωμάτων, πως γίνονται οι υπολογισμοί και οι μετρήσεις για την εύρεση της κατάστασης του καταχωρητή μετά την επίδραση τους.

1.8.1 Συμβολισμός κβαντικών πυλών

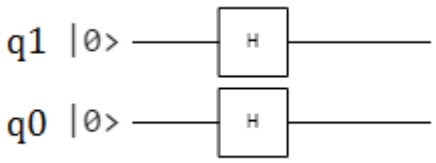
Πριν δούμε πως παριστάνουμε τα κβαντικά κυκλώματα, παρουσιάζονται σύντομα στο σχήμα 1.2 οι αναπαραστάσεις των πυλών που θα χρησιμοποιηθούν.



Σχήμα 1.2 : Αναπαραστάσεις κβαντικών πυλών

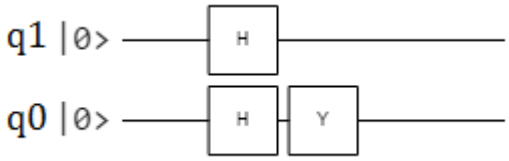
1.8.2 Μορφή κβαντικών κυκλωμάτων

Τα κβαντικά κυκλώματα επομένως είναι μια σειρά από ενέργειες (πύλες) που εφαρμόζονται με χρονική σειρά πάνω στα qubits ενός κβαντικού καταχωρητή. Μπορούμε να τα αναπαραστήσουμε σχηματικά όπως φαίνεται στο Σχήμα 1.3:



Σχήμα 1.3 : Παράδειγμα 1 κβαντικού κυκλώματος

Το παράδειγμα δείχνει ένα κβαντικό καταχωρητή των 2 qubits, σε καθένα από τα οποία εφαρμόζεται μια πύλη Hadamard. Η αρχική κατάσταση είναι |00⟩. Οι γραμμές δεν αντιστοιχούν σε διαύλους μέσα στους οποίους διέρχεται η πληροφορία, αλλά αναπαριστούν τη χρονική διαδοχή με την οποία εφαρμόζονται οι πύλες στα αντίστοιχα qubits. Επομένως στο Σχήμα 1.4 στο q0 θα εφαρμοστεί πρώτα η πύλη Hadamard και στην συνέχεια η Pauli Y.

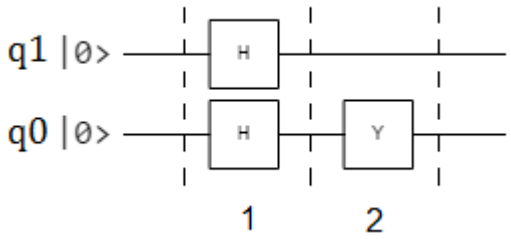


Σχήμα 1.4 : Παράδειγμα 2 κβαντικού κυκλώματος

Όπως φαίνεται από τα παραδείγματα, ο αριθμός των γραμμών του κυκλώματος είναι ίσος με τον αριθμό των qubits. Το λιγότερο σημαντικό qubit είναι το πιο χαμηλό και το περισσότερο σημαντικό είναι το ποιο ψηλό. Η χρονική σειρά είναι από αριστερά προς τα δεξιά.

1.8.3 Κβαντικός υπολογισμός

Ο υπολογισμός της κατάστασης ενός κβαντικού καταχωρητή μετά την εφαρμογή ενός κβαντικού κυκλώματος είναι μια διαδικασία βημάτων. Κάθε βήμα εφαρμόζει μέχρι μια πύλη στο αντίστοιχο qubit. Το παράδειγμα του Σχήματος 1.4 έχει δύο βήματα στην εκτέλεση του όπως φαίνεται στο Σχήμα 1.5. Ο αριθμός των απαιτούμενων βημάτων λέγεται και βάθος κυκλώματος (circuit depth).



Σχήμα 1.5 - Βήματα εκτέλεσης

Ο υπολογισμός κάθε βήματος γίνεται με την χρήση του γινόμενου Kronecker. Συγκεκριμένα υπολογίζεται το γινόμενο Kronecker των μητρώων που περιγράφουν τις

πύλες του κάθε βήματος και στην συνέχεια το αποτέλεσμα πολλαπλασιάζεται με το διάνυσμα κατάστασης του καταχωρητή για να προκύψει η νέα κατάσταση. Αν δεν υπάρχει πύλη σε κάποιο qubit χρησιμοποιείται η πύλη αδράνειας. Αν υπάρχει πύλη που δρα σε περισσότερα qubits είναι σαν να έχει υπολογιστεί το επιμέρους γινόμενο. Ας δούμε τον υπολογισμό του προηγούμενου παραδείγματος αναλυτικά.

Στο πρώτο βήμα έχουμε δύο πύλες Hadamard. Επομένως υπολογίζεται το γινόμενο των δύο μητρώων:

$$\boldsymbol{H \otimes H} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Η αρχική κατάσταση του καταχωρητή ήταν η $|00\rangle$. Με την εφαρμογή του πρώτου βήματος θα γίνει:

$$\boldsymbol{H \otimes H}|\mathbf{00}\rangle = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} = \frac{1}{2}|\mathbf{00}\rangle + \frac{1}{2}|\mathbf{01}\rangle + \frac{1}{2}|\mathbf{10}\rangle + \frac{1}{2}|\mathbf{11}\rangle$$

Στο δεύτερο βήμα εφαρμόζεται μια πύλη Y στο πρώτο qubit, ενώ στο δεύτερο δεν υπάρχει πύλη που σημαίνει ότι χρησιμοποιείται πύλη αδράνειας.

$$\boldsymbol{I \otimes Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = \begin{bmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{bmatrix}$$

Γίνεται εφαρμογή στην προηγούμενη κατάσταση:

$$\boldsymbol{I \otimes Y}\left(\frac{1}{2}|\mathbf{00}\rangle + \frac{1}{2}|\mathbf{01}\rangle + \frac{1}{2}|\mathbf{10}\rangle + \frac{1}{2}|\mathbf{11}\rangle\right) = \begin{bmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} -i\frac{1}{2} \\ i\frac{1}{2} \\ -i\frac{1}{2} \\ i\frac{1}{2} \end{bmatrix}$$

$$= -i\frac{1}{2}|\mathbf{00}\rangle + i\frac{1}{2}|\mathbf{01}\rangle - i\frac{1}{2}|\mathbf{10}\rangle + \frac{1}{2}|\mathbf{11}\rangle$$

Από το παράδειγμα γίνεται εμφανές ότι για ένα κβαντικό κύκλωμα των n qubits χρειάζονται τουλάχιστον ένα διάνυσμα 2^n και ένα μητρώο 2^nx2^n τα στοιχεία των οποίων είναι μιγαδικοί αριθμοί. Επομένως οι απαιτήσεις για μνήμη αυξάνονται εκθετικά όσο αυξάνεται ο αριθμός των qubits. Αργότερα θα φανεί πως με μια ιδιότητα μπορεί να μειωθεί η χρήση της μνήμης κατά πολύ.

Σε αυτό το κεφάλαιο περιγράφτηκαν με την σειρά το qubit και οι κβαντικές πύλες, από τα οποία κατασκευάστηκαν ο κβαντικός καταχωρητής και τα κβαντικά κυκλώματα αντίστοιχα. Τέλος, παρουσιάστηκε η βηματική εκτέλεση για τον υπολογισμό της κατάστασης ενός καταχωρητή μετά την επίδραση ενός κυκλώματος.

1.8.4 Πύλες μέτρησης

Αυτό που κάνουν οι συγκεκριμένες πύλες είναι να υπολογίζουν την κατανομή πιθανότητας για τα qubit στα οποία θα οριστούν. Κάθε πύλη μέτρησης j που είναι τοποθετημένη στα qubits k_j , j=0,…M-1 δημιουργεί μία διακριτή κατανομή πιθανότητας δύο τιμών $P_{kj}(0)$ και $P_{kj}(1)$, που αντιστοιχούν στις πιθανότητες το qubit k_j να μετρηθεί 0 ή 1 αντίστοιχα. Για περισσότερες από μια πύλες ο αριθμός των τιμών των πιθανοτήτων είναι 2^M, όπου M ο αριθμός των πυλών. Οι δυαδικές τιμές 0,1 της διακριτής μεταβλητής x_{k_j} της κατανομής του κάθε qubit που μετριέται δημιουργούν την μεταβλητή i (i=0,…2M-1) της συνολικής κατανομής χρησιμοποιώντας κατάλληλο βάρος ως εξής:

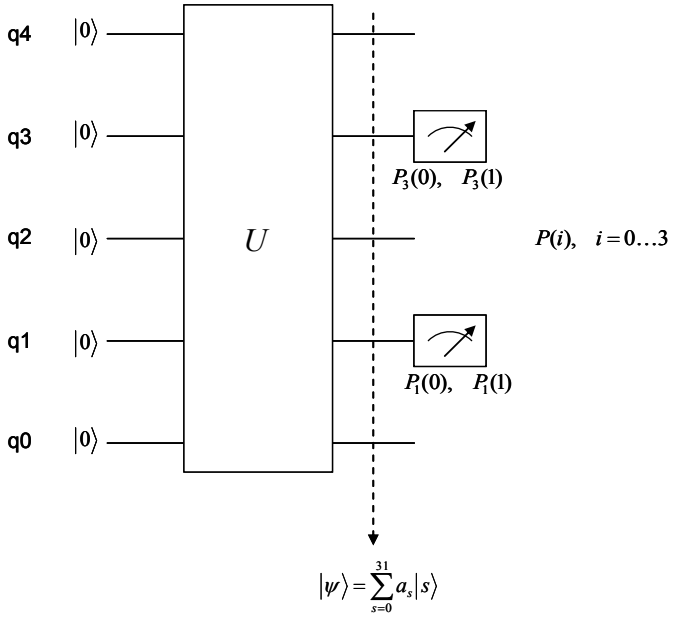
$$i\left(x_{k_0}, \dots, x_{k_{M-1}}\right) = \sum_{j=0}^{M-1} x_{k_j}\, 2^j$$

Η συνολική κατανομή πιθανότητας υπολογίζεται ως εξής

$$P(i) = P\left(x_{k_0}, \dots, x_{k_{M-1}}\right) = \sum_{(s_{N-1}s_{N-2}\dots s_0)=(0x_{k_{M-1}}0\dots 0x_{k_0}0)}^{(1x_{k_{M-1}}1\dots 1x_{k_0}1)} |a_s|^2$$

Όπου α_s είναι το πλάτος πιθανότητας.

Για παράδειγμα έστω ότι έχουμε το κύκλωμα του παρακάτω σχήματος με δύο πύλες μέτρησης στα qubits 1 και 3.



Σχήμα 1.6 - Παράδειγμα πυλών μέτρησης

Σύμφωνα με τα παραπάνω οι κατανομή των πιθανοτήτων θα είναι:

$$P(0)=P(x_{k_0}=0,\ldots,x_{k_0}=0)=\sum_{(s_{N-1}s_{N-2}\ldots s_0)=(0\bar{0}0\bar{0}0)}^{(1\bar{0}1\bar{0}1)}|a_s|^2=|a_0|^2+|a_1|^2+|a_4|^2+|a_5|^2+|a_{16}|^2+|a_{17}|^2+|a_{20}|^2+|a_{21}|^2$$

$$P(1)=P(x_{k_0}=0,\ldots,x_{k_0}=1)=\sum_{(s_{N-1}s_{N-2}\ldots s_0)=(0\bar{0}0\bar{1}0)}^{(1\bar{0}1\bar{1}1)}|a_s|^2=|a_2|^2+|a_3|^2+|a_6|^2+|a_7|^2+|a_{18}|^2+|a_{19}|^2+|a_{22}|^2+|a_{23}|^2$$

$$P(2)=P(x_{k_0}=1,\ldots,x_{k_0}=0)=\sum_{(s_{N-1}s_{N-2}\ldots s_0)=(0\bar{1}0\bar{0}0)}^{(1\bar{1}1\bar{0}1)}|a_s|^2=|a_8|^2+|a_9|^2+|a_{12}|^2+|a_{13}|^2+|a_{24}|^2+|a_{25}|^2+|a_{28}|^2+|a_{29}|^2$$

$$P(3)=P(x_{k_0}=1,\ldots,x_{k_0}=1)=\sum_{(s_{N-1}s_{N-2}\ldots s_0)=(0\bar{1}0\bar{1}0)}^{(1\bar{1}1\bar{1}1)}|a_s|^2=|a_{10}|^2+|a_{11}|^2+|a_{14}|^2+|a_{15}|^2+|a_{26}|^2+|a_{27}|^2+|a_{30}|^2+|a_{31}|^2$$

Φυσικά το άθροισμα όλων αυτών πρέπει να είναι 1.

$$\sum_{i=0}^{3}P(i)=\sum_{s=0}^{31}|a_s|^2=1$$

2. ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ GPU – ΜΟΝΤΕΛΟ CUDA

Για την παραλληλοποίηση του μεγάλου αριθμού πράξεων που απαιτούνται για τον υπολογισμό των κβαντικών κυκλωμάτων, επιλέχθηκε η χρήση αρχιτεκτονικών σε κάρτες γραφικών (GPU). Συγκεκριμένα επιλέχθηκαν οι αρχιτεκτονικές της Nvidia και υλοποίηση με την χρήση του αναπτυξιακού περιβάλλοντος της CUDA. Σε αυτό το κεφάλαιο θα παρουσιαστούν συνοπτικά αυτές οι αρχιτεκτονικές, το μοντέλο προγραμματισμού CUDA, πως γίνεται η παραλληλοποίηση, ποια είναι τα μειονεκτήματα και ποια τα πλεονεκτήματα στα οποία πρέπει να δοθεί σημασία για να επιτευχθεί η καλύτερη δυνατή απόδοση.

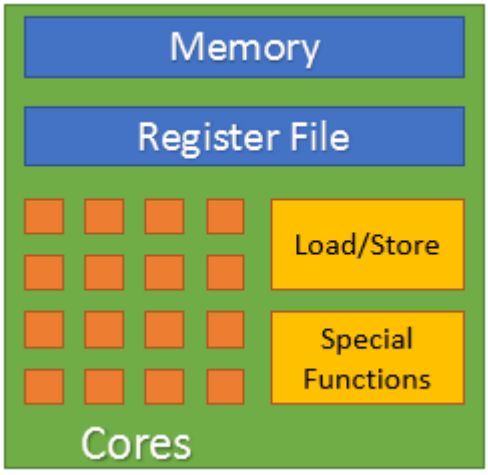
2.1 Αρχιτεκτονική των σύγχρονων GPU της Nvidia

Οι κάρτες γραφικών προορίζονται για εφαρμογές επεξεργασίας εικόνας, οι οποίες περιέχουν συνήθως απλές πράξεις σε μεγάλα μητρώα που είναι ανεξάρτητες μεταξύ τους. Για το λόγο αυτό, οι αρχιτεκτονικές των GPUs είναι σχεδιασμένες να εξυπηρετούν τέτοια προβλήματα. Διαθέτουν την δική τους ιεραρχία μνήμης και ένα σύνολο από επεξεργαστικούς πυρήνες απλού συνόλου εντολών, διατεταγμένους έτσι ώστε να μπορούν να υλοποιήσουν ταυτόχρονα την ίδια πράξη σε πολλά διαφορετικά δεδομένα, με άλλα λόγια παρέχουν παράλληλη επεξεργασία του τύπου SIMD.

Οι CPUs είναι σχεδιασμένες να έχουν μικρό latency ώστε να τελειώνουν γρήγορα τις εργασίες τους, ενώ αντίθετα οι GPUs έχουν υψηλό throughput με παραλληλοποίηση των πράξεων και μπορούν να επικαλύπτουν το latency με τον κατάλληλο προγραμματισμό της επεξεργασίας διαφορετικών κομματιών δεδομένων.

Με την πάροδο του χρόνου και την εξέλιξη της τεχνολογίας η Nvidia ανανεώνει και κυκλοφορεί στην αγορά αρχιτεκτονικές, όπως οι Tesla, Fermi, Kepler, Maxwell, Pascal. Εδώ θα γίνει η περιγραφή μιας γενικής αρχιτεκτονικής και δεν θα γίνει εμβάθυνση σε κάποια συγκεκριμένη έκδοση από τις παραπάνω.

Παρακάτω η κάρτα γραφικών θα αναφέρεται ως device και η GPU ως host. Οι όροι αυτοί χρησιμοποιούνται επίσημα στο περιβάλλον της CUDA.



Σχήμα 2.1 – Streaming Multiprocessor (SM)

2.1.1 Streaming Multiprocessor (SM)

Οι επεξεργαστικοί πυρήνες (CUDA cores) είναι οργανωμένοι σε ομάδες, τους streaming multiprocessors (SM). Κάθε SM περιέχει έναν αριθμό από cores, ειδικές μονάδες όπως μονάδες load/store και μονάδες ειδικών πράξεων (ημίτονο, συνημίτονο, τετραγωνική ρίζα) και ένα σύνολο μνημών που προορίζονται για τοπική αποθήκευση, όπως θα περιγραφθεί παρακάτω.

Κάθε core είναι ένας μικρός επεξεργαστής απλού συνόλου εντολών (RISC), με ικανότητα εκτέλεσης πράξεων κινητής υποδιαστολής.

2.1.2 Ιεραρχία μνήμης

Ειπώθηκε ότι η αρχιτεκτονική των GPUs διαθέτει δική της ιεραρχία μνήμης. Σε αυτό το κομμάτι γίνεται μια σύντομη περιγραφή της ιεραρχίας αυτής.

Αρχικά μπορεί να γίνει ένας διαχωρισμός σε δύο μεγάλες κατηγορίες. Την απομακρυσμένη μνήμη, η οποία βρίσκεται πάνω στην κάρτα αλλά όχι μέσα στο chip που περιέχονται οι επεξεργαστές, και στην on-chip μνήμη που είναι υλοποιημένη μέσα στο chip. Όσων αφορά την απομακρυσμένη μνήμη, στο πιο χαμηλό επίπεδο βρίσκεται η ***Global Memory***, το ανάλογο της RAM σε μια αρχιτεκτονική CPU. Πρόκειται για την μεγαλύτερη αλλά και πιο αργή μνήμη η οποία έχει άμεσα επικοινωνία με τον host. Λόγω της μικρής ταχύτητας που προσφέρει έχουν προστεθεί στο ίδιο επίπεδο μερικές επιπλέον μνήμες οι οποίες αυξάνουν την απόδοση για συγκεκριμένες εφαρμογές. Αυτές είναι οι εξής:

- **Constant memory**: Πρόκειται για μια μεσαίου μεγέθους μνήμη η οποία έχει κι αυτή άμεση επικοινωνία με τον host. Είναι read-only από τα υψηλότερα επίπεδα. Αυτό που προσφέρει είναι υψηλότερη ταχύτητα ανάγνωσης σε δεδομένα που δεν αλλάζουν κατά την εκτέλεση, δηλαδή σταθερές. Συγκεκριμένα, η ανάγνωση της ίδιας θέσης μνήμης από μισό warp(16 threads) είναι το ίδιο γρήγορη με μια ανάγνωση από έναν καταχωρητή. Όμως οι αναγνώσεις διαφορετικών διευθύνσεων μνήμης από μισό warp σειριοποιούνται.
- **Texture memory**: Είναι κι αυτή μια μεσαίου μεγέθους, read-only μνήμη σε άμεση επικοινωνία με τον host. Η διαφορά με την constant είναι ότι έχει την καλύτερη απόδοση εκμεταλλευόμενη την χωρική τοπικότητα (spatial locality), δηλαδή όταν τα κοντινά threads διαβάζουν από κοντινές θέσεις μνήμης.
- **Local memory**: Είναι κομμάτι της Global Memory στο οποίο κάθε thread έχει την δική του θέση και χρησιμοποιείται όταν μια τιμή δεν χωράει σε έναν καταχωρητή.

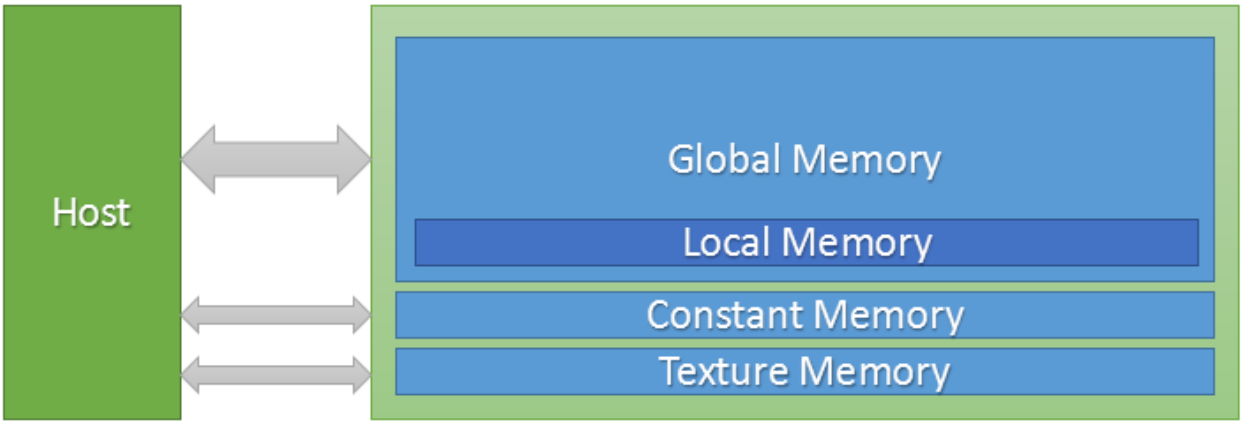
Όλα τα παραπάνω ήδη μνήμης φαίνονται στο Σχήμα 2.2 που ακολουθεί.

Τα επόμενα επίπεδα ιεραρχίας είναι on-chip, επομένως είναι μικρότερες και πιο γρήγορες μνήμες.

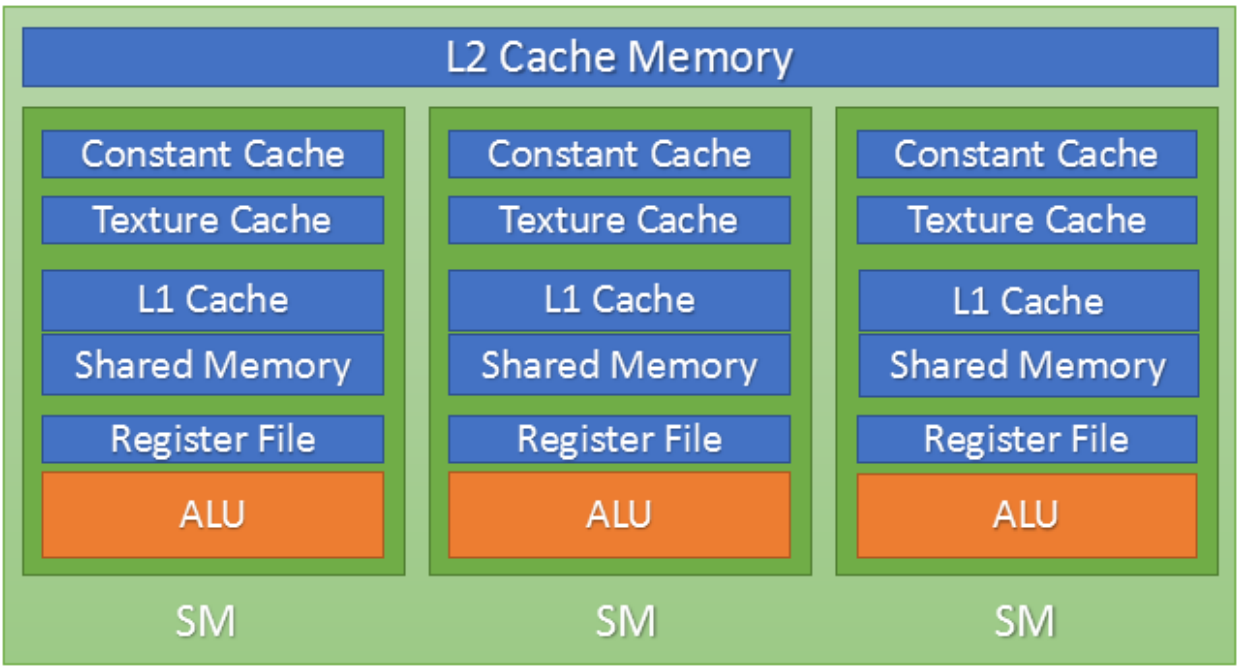
- **L2 cache**: Η global memory διαθέτει κρυφές μνήμες μέσα στο chip για την αποφυγή μεγάλου αριθμού μεταφορών έξω από αυτό. Η L2 είναι κοινή για όλους τους streaming multiprocessors που υπάρχουν μέσα στο device.

- **L1 cache**: Ένα ακόμα επίπεδο κρυφής μνήμης βρίσκεται μέσα σε κάθε multiprocessor.
- **Constant cache και texture cache:** Κρυφές μνήμες διαθέτουν και τα δύο άλλα είδη απομακρυσμένης μνήμης.
- **Shared Memory:** Χρησιμοποιείται για την αποθήκευση ενδιάμεσων αποτελεσμάτων και την επικοινωνία των threads μέσα σε ένα block χωρίς την ανάγκη η πληροφορία να πηγαίνει σε πιο αργά επίπεδα. Η shared memory είναι κομμάτι της L1 cache και υπάρχει η δυνατότητα αύξησης τους μεγέθους της με αντίστοιχη μείωση της δεύτερης μέσα σε κάποια όρια.

Τέλος κάθε multiprocessor διαθέτει ένα δικό του σύνολο καταχωρητών για την αποθήκευση τοπικών τιμών κατά την διάρκεια της εκτέλεσης. Η ιεραρχία της on chip μνήμης φαίνεται στο Σχήμα 2.3.



Σχήμα 2.2 - Ιεραρχία απομακρυσμένης μνήμης



Σχήμα 2.3 - Ιεραρχία on chip μνήμης

2.2 Μοντέλο εκτέλεσης CUDA

Πιο πάνω έγινε περιγραφή της αρχιτεκτονικής και πως είναι υλοποιημένη σε φυσικό επίπεδο. Τώρα θα αναλυθεί πως γίνεται η εκτέλεση χρησιμοποιώντας αυτό το υλικό.

Η παράλληλη εκτέλεση πολλών πράξεων επιτυγχάνεται με την χρήση ***threads***. Τα threads τρέχουν στα cores του device παράλληλα. Καθένα έχει το δικό του ξεχωριστό ID, με το οποίο ο προγραμματιστής μπορεί να του αναθέσει μια εργασία. Οργώνονται σε ομάδες των 32, τα warps τα οποία τρέχουν αλληλένδετα και εκτελούν την ίδια εντολή σε διαφορετικά δεδομένα. Αν μέσα σε ένα warp οριστούν παραπάνω εντολές, όπως για παράδειγμα σε μια διακλάδωση, τότε η εκτέλεση του warp σειριοποιείται. Διαφορετικά warps μπορούν να τρέξουν παράλληλα με διαφορετικές εντολές.

Κατά την εκτέλεση υπάρχει μια ιεραρχία η οποία αποτελείται από τα threads που οργανώνονται σε ***blocks*** και αυτά με την σειρά τους ομαδοποιούνται περαιτέρω σε ένα ***grid***. Τα blocks και grids έχουν διαστάσεις (1,2 ή 3 σε πιο καινούργιες αρχιτεκτονικές) οι οποίες ελέγχονται από τον προγραμματιστή. Η εκτέλεση ενός block γίνεται μέσα σε έναν SM και δεν μπορεί να μοιραστεί σε περισσότερους σε περίπτωση που περιέχει περισσότερα threads απ' ότι επιτρέπει ο SM. Ένας SM όμως μπορεί να περιέχει παραπάνω από ένα block ταυτόχρονα, όσο του επιτρέπουν οι πόροι του. Η εκτέλεση του grid αντιστοιχεί σε όλο το device.

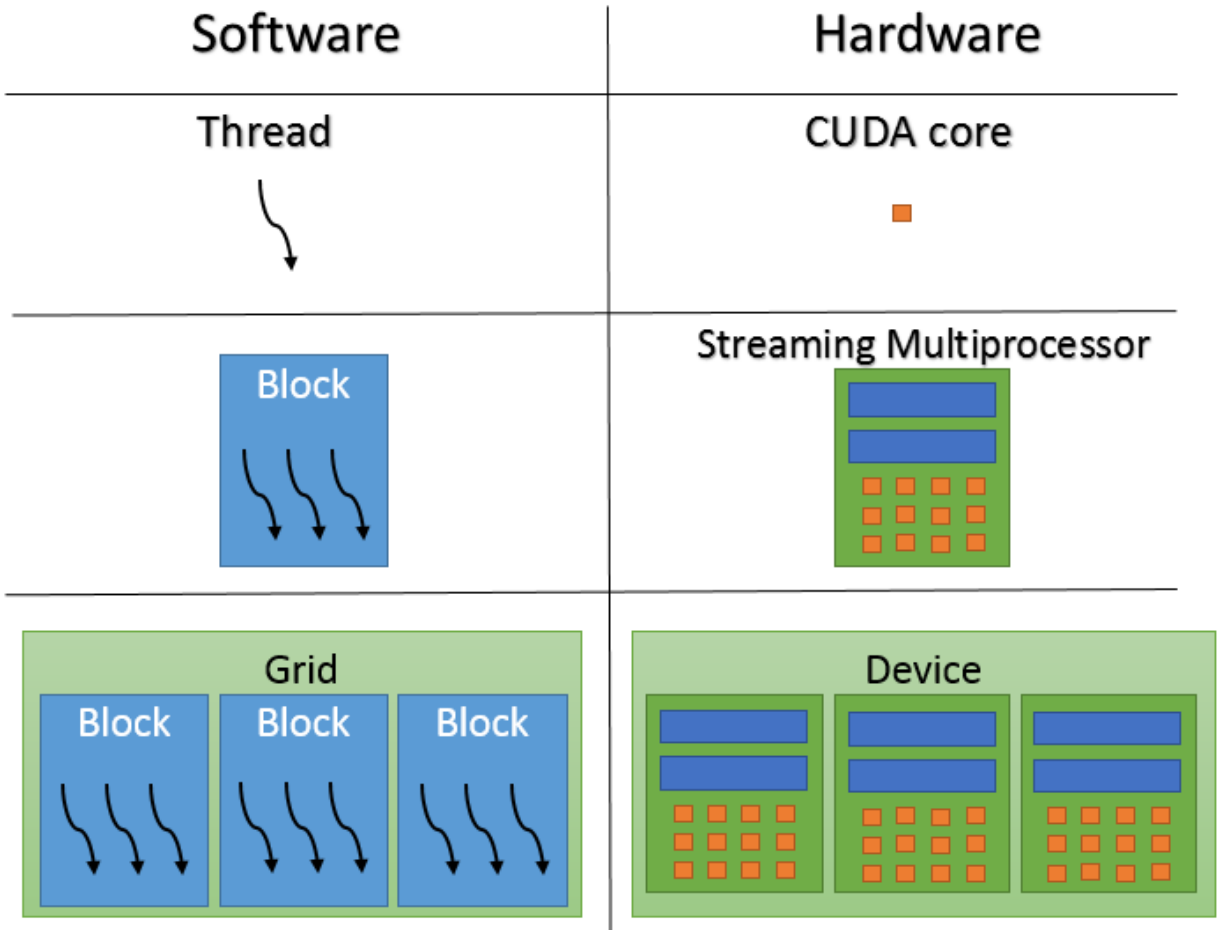
Ο ορισμός του αριθμού των blocks και threads γίνεται από τον προγραμματιστή, ενώ ο καταμερισμός τους στους πόρους γίνεται αυτόματα. Στο Σχήμα 2.4 φαίνεται η αντιστοίχιση των εννοιών μεταξύ software και hardware.

Η παραλληλοποίηση όπως αναφέρθηκε έχει την μορφή SIMD(single instruction on multiple data). Το ίδιο σειριακό κομμάτι κώδικα εκτελείται σε κάθε thread, πάνω σε διαφορετικά δεδομένα. Για παράδειγμα σε μια πρόσθεση δύο διανυσμάτων μεγέθους 40 θα πρέπει να δημιουργηθούν 40 threads καθένα από τα οποία θα εκτελέσει την εντολή *V1[i] + V2[i] = V1[i]* . Ο δείκτης i ορίζεται ως το ID των threads επομένως κάθε thread θα αναλάβει να προσθέσει μια ξεχωριστή θέση διανύσματος.

2.3 Προγραμματισμός σε CUDA

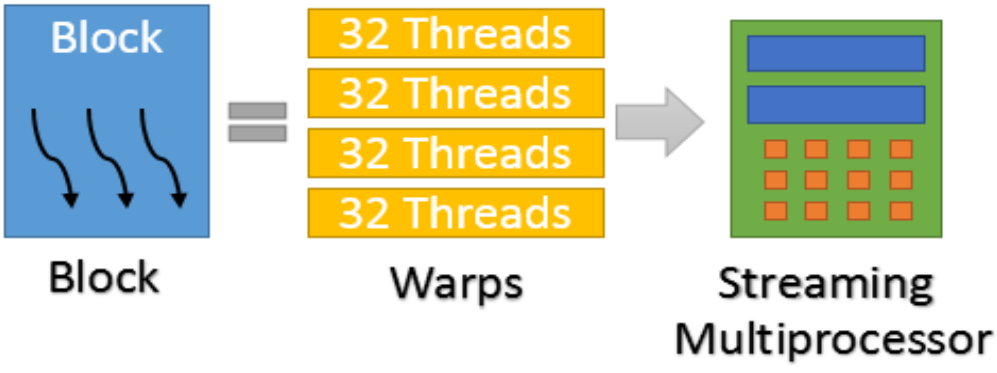
Η γλώσσα προγραμματισμού που χρησιμοποιείται είναι μια επέκταση των C/C++ που περιέχει ειδικές βιβλιοθήκες με συναρτήσεις που εκτελούνται στην κάρτα γραφικών. Επομένως σε ένα αρχείο κώδικα υπάρχει ταυτόχρονα ο προγραμματισμός του host και του device. Παρακάτω περιγράφονται τα βασικά βήματα για τον σωστή χρήση του device.

- **Έλεγχος device:** Ένα προαιρετικό μεν αλλά σημαντικό βήμα δε είναι να γίνει έλεγχος για το αν υπάρχει διαθέσιμη κάρτα γραφικών της Nvidia και αν λειτουργεί σωστά. Αν κάποιο device είναι διαθέσιμο για χρήση, υπάρχουν συναρτήσεις που σου επιστρέφουν μεταβλητές με πληροφορίες που μπορούν να χρησιμοποιηθούν αργότερα όπως το μέγεθος της μνήμης ή τις διαστάσεις των grid και blocks.
- **Δέσμευση μνήμης:** Το νόημα της χρήσης κάρτας γραφικών είναι η επεξεργασία πολλών δεδομένων παράλληλα. Αυτά τα δεδομένα πρέπει να αποθηκευτούν στην global memory του device. Για να γίνει αυτό είναι απαραίτητο να γίνει δέσμευση της μνήμης από πριν. Υπάρχει ειδική συνάρτηση η ***cudaMalloc*** η οποία είναι αντίστοιχη με την *malloc*.



Σχήμα 2.4 – Αντιστοίχιση Software και Hardware

Στο Σχήμα 2.5 φαίνεται πως ένα block αποτελείται από ένα σύνολο από warps τα οποία εκτελούνται σε έναν multiprocessor.



Σχήμα 2.5 – Σχέση μεταξύ warps και blocks

- **Μεταφορά δεδομένων προς το device:** Αφού έχει γίνει δέσμευση της μνήμης και έστω ότι έχουν παραχθεί δεδομένα με κάποιον τρόπο στην μνήμη του host, με την εντολή ***cudaMemcpy*** γίνεται αντιγραφή από τον host στο device. Τα δεδομένα μεταφέρονται στην global memory και είναι έτοιμα για χρήση.
- **Κλήση kernel:** Ο κώδικας που θα τρέξει στο device είναι γραμμένος σε μια συνάρτηση και όχι στην main(). Επομένως όταν έρθει η ώρα για την επεξεργασία, απαιτεί η κλήση αυτής της συνάρτησης που ονομάζεται kernel. Κατά την κλήση ενός kernel πέραν από τυχόν δεδομένα που μπορεί να διαβιβάζονται σαν παράμετροι, γίνεται και ο ορισμός του αριθμού των blocks και των threads ανά block που θα δημιουργηθούν για την συγκεκριμένη εκτέλεση. Αυτό γίνεται με ένα τελεστή που δεν υπάρχει στις C/C++, τα τριπλά εισαγωγικά. Για παράδειγμα:

  kernel_name<<<#_of_blocks, #_of_threads>>> (parameter list);

- **Μεταφορά δεδομένων προς το host:** Μετά την ολοκλήρωση της εκτέλεσης τα δεδομένα που παράχθηκαν πρέπει να επιστρέψουν στον host για να διαβαστούν, να γραφτούν σε αρχείο κτλ. Με την ίδια εντολή *cudaMemcpy* αλλά με μια διαφορετική παράμετρο, τα δεδομένα αντιγράφονται από την global memory στην μνήμη του host.
- **Αποδέσμευση μνήμης:** Ένας σωστός προγραμματιστής πρέπει όταν δεν την χρειάζεται άλλο να αποδεσμεύσει την μνήμη στο device με την χρήση της συνάρτησης *cudaFree*.

Αυτά είναι τα πιο βασικά βήματα για την δημιουργία μιας εφαρμογής που τρέχει σε αρχιτεκτονική κάρτας γραφικών. Φυσικά ο προγραμματισμός μπορεί να γίνει αρκετά πιο πολύπλοκος. Υπάρχουν πολλές συναρτήσεις οι οποίες προσθέτουν λειτουργίες στα παραπάνω. Για παράδειγμα υπάρχουν συναρτήσεις οι οποίες υπάρχουν για να προσφέρουν βέλτιστη απόδοση σε συνηθισμένες πράξεις όπως η αρχικοποίηση ενός διανύσματος ή πράξεις μεταξύ διανυσμάτων.

3. ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΠΡΟΣΟΜΟΙΩΤΗ

Σε αυτό το κεφάλαιο παρουσιάζεται η διαδικασία σχεδίασης και οι μέθοδοι υλοποίησης του επιθυμητού εργαλείου με βάση την θεωρία που περιγράφτηκε στα προηγούμενα. Γίνεται αιτιολόγηση των επιλογών που έγιναν κατά την διάρκεια των δύο διαδικασιών και τονίζονται τόσο τα θετικά όσο και τα αρνητικά του αποτελέσματος.

3.1 Σχεδίαση

Πρώτο βήμα σε κάθε διαδικασία υλοποίησης είναι η θεωρητική σχεδίαση. Αρχικά καθορίζονται οι απαιτήσεις για το τελικό αποτέλεσμα, οι διαθέσιμοι πόροι για την υλοποίηση, τα δεδομένα που προκύπτουν από την θεωρητική ανάλυση και λαμβάνονται αποφάσεις για το μοντέλο που θα χρησιμοποιηθεί, τον αλγόριθμο της υλοποίησης, τις μεθόδους επικοινωνίας με τον χρήστη (δηλαδή την είσοδο και την έξοδο του εργαλείου), σε ποιες απαιτήσεις θα υπάρξει μια ΄΄χαλαρότητα΄΄ προκείμενου να κερδηθεί κάτι άλλο σε περιπτώσεις που υπάρχει κάποιο trade-off.

3.1.1 Καθορισμός απαιτήσεων

Οι απαιτήσεις που καθορίστηκαν εξ αρχής είναι οι ακόλουθες:

- Ένα εργαλείο με εύκολη πρόσβαση και εύκολη χρήση.
- Να είναι πλήρες, δηλαδή να μπορεί να εξομοιώνει οποιοδήποτε κβαντικό κύκλωμα σχεδιάζει και εισάγει ο χρήστης και να μην εξειδικεύεται σε συγκεκριμένες εφαρμογές.
- Να μπορεί να λειτουργεί για μεγάλα κυκλώματα το οποίο σημαίνει να προσομοιώνει όσο το δυνατόν μεγαλύτερο αριθμό qubits αλλά και πυλών.
- Να είναι αποδοτικό σε χρόνο εκτέλεσης ακόμα και για μεγάλα κυκλώματα.

3.1.2 Καθορισμός δεδομένων

Από την κβαντική θεωρία, την αρχιτεκτονική των GPU και το μοντέλο της CUDA προκύπτουν κάποια δεδομένα που δεν μπορούν να αγνοηθούν.

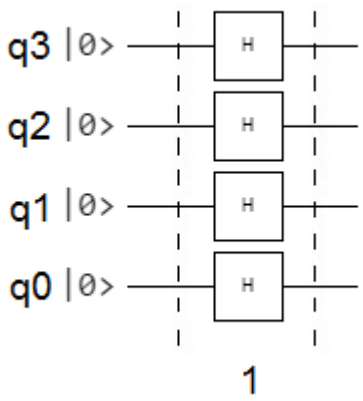
- Όπως ειπώθηκε στο αντίστοιχο κεφάλαιο ο υπολογισμός στα κβαντικά κυκλώματα είναι μια ακολουθία βημάτων που πρέπει να τηρηθεί και επομένως αυτό το κομμάτι δεν μπορεί να παραλληλοποιηθεί με προφανή τρόπο.
- Οι αριθμοί που επεξεργάζονται είναι μιγαδικοί.
- Η αύξηση των qubits αυξάνει εκθετικά την απαίτηση σε μνήμη.
- Η διαθέσιμη μνήμη στις κάρτες γραφικών είναι περιορισμένη και οι μεταφορές από και προς αυτήν είναι αργές.
- Οι κάρτες γραφικών είναι σχεδιασμένες για εκτέλεση πολλών απλών πράξεων παράλληλα σε μορφή SIMD.

- Αν και μπορούν να χρησιμοποιηθούν αριθμοί κινητής υποδιαστολής διπλής ακρίβειας, σε μία τέτοια περίπτωση η απόδοση πέφτει δραματικά.
- Υπάρχουν περιορισμοί στο πόσα threads μπορούν να τρέξουν ανά block και ανά SM.
- Τα threads μέσα σε ένα warp πρέπει να τρέχουν την ίδια εντολή αλλιώς σειριοποιούνται.

3.1.3 Διαχείριση μνήμης

Έγινε αντιληπτό ότι οι απαιτήσεις για μνήμη είναι μεγάλες. Συγκεκριμένα για n qubits χρειάζεται ένα μητρώο 2^nx2^n για τον υπολογισμό ενός βήματος σε ένα κύκλωμα. Με δεδομένο ότι έχουμε μιγαδικούς αριθμούς μονής ακρίβειας, οι οποίοι καταλαμβάνουν 8 bytes, ο συνολικός χώρος ανέρχεται στα 2^nx2^nx8 bytes. Για 5 qubits απαιτούνται 8192 bytes ή 8 Kbytes. Στα 10 qubits απαιτούνται 8.388.608 bytes = 8 Mbytes. Στα 15 qubits απαιτούνται 8.589.934.592 bytes = 8 Gbytes. Στα 20 qubits απαιτούνται 8.796.093.022.208 bytes = 8 Tbytes! Επομένως μετά τα 15 qubits εμφανίζονται τιμές απαγορευτικές για αποθήκευση στην κύρια μνήμη μιας GPU. Επιπλέον είναι πιθανό ένα τέτοιο μητρώο να είναι πάρα πολύ αραιό και να δαπανάται μνήμη χωρίς λόγο. Συγκεκριμένα μπορεί να υπολογιστεί ότι η αραιότητα σε κύκλωμα με n qubits και μια πύλη σε ένα βήμα είναι $\frac{1}{2^n}$. Σε 10 qubits σημαίνει ότι 1 στοιχείο στο εκατομμύριο θα είναι ωφέλιμο και ο υπόλοιπος χώρος θα καταλαμβάνεται από μηδενικά. Πέρα από την κατανάλωση της μνήμης αυτό μπορεί να έχει αρνητικό αποτέλεσμα και στην απόδοση, αν γίνονται εκατομμύρια πράξεις πολλαπλασιασμού με το μηδέν.

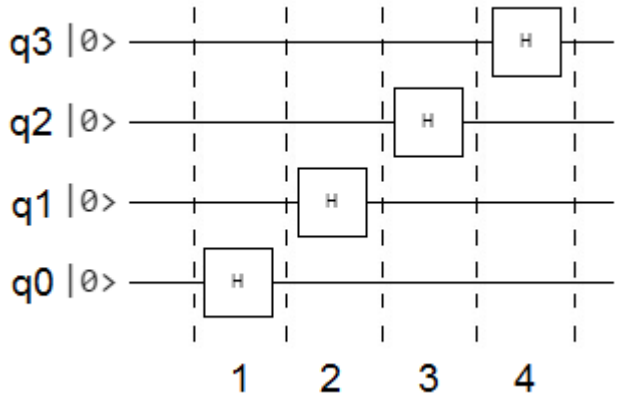
Για την βελτίωση του παραπάνω προβλήματος χρησιμοποιείται μία ιδέα για την εκμετάλλευση μιας ιδιότητας που εμφανίζεται στους υπολογισμούς και περιγράφεται στην συνέχεια. Αυτή είναι και η βασική ιδέα του αλγόριθμου της υλοποίησης. Έστω ότι έχουμε λοιπόν το κύκλωμα του Σχήματος 3.1.



Σχήμα 3.1 – Παράδειγμα κυκλώματος ενός βήματος

Σε αυτό φαίνεται ότι το κύκλωμα αποτελείται από ένα βήμα με 4 πύλες του ενός qubit. Ο χώρος που χρειάζεται σύμφωνα με τα προηγούμενα είναι 2^4x2^4x8 bytes = 2 Kbytes. Στο Σχήμα 3.2 φαίνεται ένα ισοδύναμο κύκλωμα. Όπως γίνεται αντιληπτό οποιοδήποτε βήμα σε ένα κβαντικό κύκλωμα μπορεί να διαχωριστεί σε επιμέρους βήματα το καθένα από τα οποία θα περιέχει ακριβώς μια πύλη. Αρχικά αυτό φαίνεται να μην εξυπηρετεί τον σκοπό

μας αφού η απαίτηση σε χώρο μένει ίδια και τα βήματα εκτέλεσης αυξάνονται. Όμως μια παρατήρηση αλλάζει τα δεδομένα.



Σχήμα 3.2 – Παράδειγμα ισοδύναμου κυκλώματος 4 βημάτων

Αφού σε κάθε βήμα υπάρχει ακριβώς μια πύλη, ο υπολογισμός του μητρώου του βήματος είναι μια πράξη γινομένου Kronecker, μεταξύ του μητρώου της πύλης και τόσων πυλών αδράνειας όσα είναι τα qubits μείον 1. Στο παράδειγμα του Σχήματος 3.2 ισχύει:

$$\boldsymbol{Βήμα\ 1}\colon\ \boldsymbol{I \otimes I \otimes I \otimes H}$$

$$\boldsymbol{Βήμα\ 2}\colon\ \boldsymbol{I \otimes I \otimes H \otimes I}$$

$$\boldsymbol{Βήμα\ 3}\colon\ \boldsymbol{I \otimes H \otimes I \otimes I}$$

$$\boldsymbol{Βήμα\ 4}\colon\ \boldsymbol{H \otimes I \otimes I \otimes I}$$

Οι παραπάνω πράξεις είναι απλές καθώς τα μητρώα I είναι ταυτοτικά δηλαδή περιέχουν 1 στην διαγώνιο τους. Το μητρώο του κάθε βήματος μπορεί να δημιουργηθεί χωρίς να εκτελεστούν ρητά οι πράξεις, εκμεταλλευόμενοι το γεγονός ότι μπορούμε να προβλέψουμε την θέση των μη τετριμμένων στοιχείων 0 και 1 της κάθε πύλης ανάλογα σε ποιο qubit επιδρά αυτή. Πιο αναλυτικά έχουμε τα εξής δεδομένα:

- Ένα κβαντικό κύκλωμα ***n*** qubits.
- Μια πύλη ***A*** με μητρώο $\begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{bmatrix}$
- ***q*** είναι η θέση της πύλης, δηλαδή σε πιο qubit επιδρά.
- Οι συντεταγμένες των στοιχείων του μητρώου είναι $a \to (0,0)$, $b \to (0,1)$, $c \to (1,0)$, $d \to (1,1)$
- Το μέγεθος του μητρώου αποτελέσματος είναι 2^nx2^n, κάθε διάσταση δηλαδή έχει 2^n στοιχεία που αριθμούνται από το 0 ως το 2^{n-1}
- Για την αρίθμηση κάθε διάστασης σε δυαδικό σύστημα απαιτούνται ***n*** bits $\mathbf{b_{n-1}b_{n-2}\ldots b_1b_0}$.

Οι συντεταγμένες του πίνακα στις οποίες θα βρίσκεται το στοιχείο *α*, είναι οι αριθμοί κάθε διάστασης που στο q-ιοστό bit της δυαδικής τους αναπαράστασης έχουν τις συντεταγμένες του *α*, δηλαδή 0,0. Αν ***i*** είναι η γραμμή και ***j*** η στήλη σημαίνει ότι

$$\boldsymbol{i = b_{n-1}b_{n-2}\dots b_{q+1}\,0\,b_{q-1}\dots b_1b_0} \text{ και } \boldsymbol{j = b_{n-1}b_{n-2}\dots b_{q+1}\,0\,b_{q-1}\dots b_1b_0}$$

Δηλαδή το b_q = 0 και στις δύο διαστάσεις. Αντίστοιχα ισχύουν για τα υπόλοιπα στοιχεία με τις ανάλογες συντεταγμένες.

Το *b* βρίσκεται στις θέσεις που καθορίζουν τα:

$$\boldsymbol{i = b_{n-1}b_{n-2}\dots b_{q+1}\,0\,b_{q-1}\dots b_1b_0} \text{ και } \boldsymbol{j = b_{n-1}b_{n-2}\dots b_{q+1}\,1\,b_{q-1}\dots b_1b_0}$$

Το *c* βρίσκεται στις θέσεις που καθορίζουν τα:

$$\boldsymbol{i = b_{n-1}b_{n-2}\dots b_{q+1}\,1\,b_{q-1}\dots b_1b_0} \text{ και } \boldsymbol{j = b_{n-1}b_{n-2}\dots b_{q+1}\,0\,b_{q-1}\dots b_1b_0}$$

Το *d* βρίσκεται στις θέσεις που καθορίζουν τα:

$$\boldsymbol{i = b_{n-1}b_{n-2}\dots b_{q+1}\,1\,b_{q-1}\dots b_1b_0} \text{ και } \boldsymbol{j = b_{n-1}b_{n-2}\dots b_{q+1}\,1\,b_{q-1}\dots b_1b_0}$$

Για να γίνει πιο κατανοητό παρουσιάζονται παραδείγματα για n = 3 και την πύλη A με το γενικό μητρώο που χρησιμοποιήθηκε πιο πάνω.

Για q = 0. Η πύλη βρίσκεται στο πρώτο qubit και ισχύει

$$\boldsymbol{I \otimes I \otimes A} = \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\end{bmatrix} \otimes \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\end{bmatrix} \otimes \begin{bmatrix}\boldsymbol{a} & \boldsymbol{b}\\ \boldsymbol{c} & \boldsymbol{d}\end{bmatrix} = \begin{bmatrix}\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}\end{bmatrix} \otimes \begin{bmatrix}\boldsymbol{a} & \boldsymbol{b}\\ \boldsymbol{c} & \boldsymbol{d}\end{bmatrix}$$

$$= \begin{bmatrix}\boldsymbol{a} & \boldsymbol{b} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \boldsymbol{c} & \boldsymbol{d} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \boldsymbol{b} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \boldsymbol{d} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \boldsymbol{b} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \boldsymbol{d} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \boldsymbol{b}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \boldsymbol{d}\end{bmatrix}$$

Το *α* βρίσκεται στις συντεταγμένες (0,0), (2,2), (4,4), (6,6) ή σε δυαδικό (000,000), (010,010), (100,100), (110,110). Αφού q=0 βρίσκεται σωστά στις θέσεις με i,b_0=0 και j, b_0=0.

Το *b* βρίσκεται στις συντεταγμένες (0,1), (2,3), (4,5), (6,7) ή σε δυαδικό (000,001), (010,011), (100,101), (110,111). Αφού q=0 βρίσκεται σωστά στις θέσεις με i,b_0=0 και j, b_0=1.

Το *c* βρίσκεται στις συντεταγμένες (1,0), (3,2), (5,4), (7,6) ή σε δυαδικό (001,000), (011,010), (101,100), (111,110). Αφού q=0 βρίσκεται σωστά στις θέσεις με i,b_0=1 και j, b_0=0.

Το *d* βρίσκεται στις συντεταγμένες (1,1), (3,3), (5,5), (7,7) ή σε δυαδικό (001,001), (011,011), (101,101), (111,111). Αφού q=0 βρίσκεται σωστά στις θέσεις με i,b_0=1 και j, b_0=1.

Στον Πίνακα 3.1 φαίνονται αναλυτικά όλοι οι αριθμοί σε δυαδικό και που αντιστοιχεί κάθε στοιχείο.

Πίνακας 3.1 – Συντεταγμένες της πύλης A για n=3 και q=0

a		b		c		d	
i	j	i	j	i	j	i	j
00**0**	00**0**	00**0**	00**1**	00**1**	00**0**	00**1**	00**1**
01**0**	01**0**	01**0**	01**1**	01**1**	01**0**	01**1**	01**1**
10**0**	10**0**	10**0**	10**1**	10**1**	10**0**	10**1**	10**1**
11**0**	11**0**	11**0**	11**1**	11**1**	11**0**	11**1**	11**1**

Για q = 1. Η πύλη βρίσκεται στο δεύτερο qubit και ισχύει

$$\boldsymbol{I \otimes A \otimes I} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \otimes \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{c} & \boldsymbol{d} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \boldsymbol{b} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \boldsymbol{d} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{a} & \mathbf{0} & \boldsymbol{b} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{a} & \mathbf{0} & \boldsymbol{b} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{c} & \mathbf{0} & \boldsymbol{d} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{c} & \mathbf{0} & \boldsymbol{d} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \mathbf{0} & \boldsymbol{b} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a} & \mathbf{0} & \boldsymbol{b} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \mathbf{0} & \boldsymbol{d} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{c} & \mathbf{0} & \boldsymbol{d} \end{bmatrix}$$

Το *a* βρίσκεται στις συντεταγμένες (0,0), (1,1), (4,4), (5,5) ή σε δυαδικό (000,000), (001,001), (100,100), (101,101). Αφού q=1 βρίσκεται σωστά στις θέσεις με i,b_1=0 και j, b_1=0.

Το *b* βρίσκεται στις συντεταγμένες (0,2), (1,3), (4,6), (5,7) ή σε δυαδικό (000,010), (001,011), (100,110), (101,111). Αφού q=1 βρίσκεται σωστά στις θέσεις με i,b_1=0 και j, b_1=1.

Το *c* βρίσκεται στις συντεταγμένες (2,0), (3,1), (6,4), (7,5) ή σε δυαδικό (010,000), (011,001), (110,100), (111,101). Αφού q=1 βρίσκεται σωστά στις θέσεις με i,b_1=1 και j, b_1=0.

Το *d* βρίσκεται στις συντεταγμένες (2,2), (3,3), (6,6), (7,7) ή σε δυαδικό (010,010), (011,011), (110,110), (111,111). Αφού q=1 βρίσκεται σωστά στις θέσεις με i,b_1=1 και j, b_1=1.

Στον Πίνακα 3.2 φαίνονται αναλυτικά όλοι οι αριθμοί σε δυαδικό και που αντιστοιχεί κάθε στοιχείο.

Πίνακας 3.2 – Συντεταγμένες της πύλης A για n=3 και q=1

a		b		c		d	
i	j	i	j	i	j	i	j
0**0**0	0**0**0	0**0**0	0**1**0	0**1**0	0**0**0	0**1**0	0**1**0
0**0**1	0**0**1	0**0**1	0**1**1	0**1**1	0**0**1	0**1**1	0**1**1
1**0**0	1**0**0	1**0**0	1**1**0	1**1**0	1**0**0	1**1**0	1**1**0
1**0**1	1**0**1	1**0**1	1**1**1	1**1**1	1**0**1	1**1**1	1**1**1

Βασίζοντας την υλοποίηση σε αυτήν την παρατήρηση μπορεί να γίνει ο υπολογισμός αυτών των θέσεων κατά την εκτέλεση και να μην χρειαστεί να υπολογιστεί καθόλου το μητρώο του κάθε βήματος. Αυτό θα γίνει πιο κατανοητό παρακάτω. Εδώ αυτό που πρέπει να κρατηθεί είναι ο τεράστιος χώρος που κερδίζεται αφού τα 2^nx2^nx8 bytes δεν χρειάζεται να αποθηκευτούν στην μνήμη.

Παραμένει όμως το ερώτημα πόσο μνήμη απαιτείται. Η απάντηση δίνεται από τον τρόπο της υλοποίησης. Όπως έχει ήδη δειχθεί ο υπολογισμός ενός βήματος εκτέλεσης είναι ο πολλαπλασιασμός του μητρώου του γινομένου Kronecker επί το διάνυσμα κατάστασης και αποτέλεσμα είναι το νέο διάνυσμα κατάστασης. Για παράδειγμα :

$$\begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Αφού δεν χρειάζεται να γίνει αποθήκευση του μητρώου, μένουν προς αποθήκευση δύο διανύσματα διάστασης 2^n, όπου n ο αριθμός των qubits. Επομένως ο χώρος που απαιτείται είναι 2^{n+1} x 8 bytes.

Οι ελεγχόμενες πύλες των 2 qubits αποτελούνται, όπως είδαμε, από τα qubits ελέγχου και στόχου (control και target). Ο στόχος υλοποιεί μια από τις πύλες ενός qubit εφόσον ο έλεγχος είναι στην βασική κατάσταση 1, διαφορετικά δεν γίνεται κάποια αλλαγή. Τα μητρώα αυτών των πυλών έχουν την μορφή

$$\begin{vmatrix} I & 0 \\ 0 & U \end{vmatrix}$$

Όπου *I* το ταυτοτικό μητρώο και *U* το μητρώο της πύλης ενός qubit. Επομένως κατά την υλοποίηση ελέγχεται το qubit ελέγχου και αναλόγως εφαρμόζεται η πύλη του qubit στόχου όπως περιγράφτηκε προηγουμένως. Αντίστοιχα γίνεται επέκταση και στις πύλες των τριών qubits.

Λίγο διαφορετική υλοποίηση γίνεται στην περίπτωση της πύλης SWAP η οποία δεν είναι ελεγχόμενη. Σε αυτή την περίπτωση γίνεται έλεγχος αν τα δύο qubits της πύλης

βρίσκονται στην ίδια κατάσταση και αν αυτό δεν ισχύει γίνεται εναλλαγή των καταστάσεων τους.

3.1.4 Καταμερισμός εργασιών

Η σωστή χρήση μιας κάρτας γραφικών για μέγιστη απόδοση περιλαμβάνει τον σωστό καταμερισμό των εργασιών στους διαθέσιμους πόρους. Με άλλα λόγια είναι εξαιρετικά σημαντικό πως θα γίνει η ανάθεση των πράξεων σε threads και πως θα παραλληλοποιηθεί ο κώδικας της υλοποίησης.

Για ακόμα μια φορά αναφέρεται ότι ένα βήμα εκτέλεσης μια κβαντικής πύλης ανάγεται υπολογιστικά σε γινόμενο μητρώου επί διάνυσμα. Το μητρώο σε αυτήν την περίπτωση δεν είναι αποθηκευμένο αλλά υπάρχει θεωρητικά γνωρίζοντας το μητρώο της πύλης που εκτελείται και σε πιο qubit ενεργεί. Επομένως η υλοποίηση τόσων threads όσα είναι τα στοιχεία του πίνακα δεν έχει νόημα καθώς τα περισσότερα από αυτά τα στοιχεία θα είναι μηδέν. Όμως μπορούμε να είμαστε σίγουροι ότι κάθε γραμμή θα περιέχει ακριβώς δύο στοιχεία λόγω του γινομένου Kronecker με της πύλες αδράνειας, τα μητρώα των οποίων έχουν 1 μόνο στην κύρια διαγώνιο. Επομένως κάθε γραμμή περιέχει δύο πράξεις πολλαπλασιασμού και μιας πρόσθεσης. Για το παράδειγμα που αντιστοιχεί στην πύλη ενός qubit A στο μεσαίο από τα τρία qubits ($\boldsymbol{I} \otimes \boldsymbol{A} \otimes \boldsymbol{I}$) όταν αυτή επιδρά στην κατάσταση $|000\rangle$:

$$\begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Στην πρώτη γραμμή γίνονται οι πράξεις **a*1+b*0 = a** , ενώ στην τρίτη γραμμή γίνονται οι πράξεις **c*1+d*0 = c**. Επιλέγεται λοιπόν η δημιουργία 2^n threads καθένα από τα οποία εκτελεί τις πράξεις που απαιτούνται για μια γραμμή. Θα μπορούσε να γίνει δημιουργία thread για κάθε έναν πολλαπλασιασμό. Σε αυτήν την περίπτωση όμως θα έπρεπε να υπάρχει επικοινωνία μεταξύ threads για την πράξη της πρόσθεσης πράγμα που σημαίνει αποθήκευση ενδιάμεσου αποτελέσματος και αύξηση του χρόνου εκτέλεσης. Στο παραπάνω παράδειγμα θα έχουμε:

$$\begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} thread\_0 \rightarrow a * 1 + b * 0 = a \\ thread\_1 \rightarrow a * 0 + b * 0 = 0 \\ thread\_2 \rightarrow c * 1 + d * 0 = c \\ thread\_3 \rightarrow c * 0 + d * 0 = 0 \\ thread\_4 \rightarrow a * 0 + b * 0 = 0 \\ thread\_5 \rightarrow a * 0 + b * 0 = 0 \\ thread\_6 \rightarrow c * 0 + d * 0 = 0 \\ thread\_7 \rightarrow c * 0 + d * 0 = 0 \end{array}$$

3.1.5 Εισαγωγή δεδομένων από χρήστη

Για να παρουσιαστεί αναλυτικά η υλοποίηση θα πρέπει να είναι γνωστή η μέθοδος που εισάγει ο χρήστης τα δεδομένα της εξομοίωσης. Για τον σκοπό αυτό δημιουργήθηκε ένα πρότυπο περιγραφής κβαντικών κυκλωμάτων βασισμένο στην QASM, μια υπάρχουσα απλή γλώσσα περιγραφής που μοιάζει με assembly. Η βασική ιδέα είναι ότι με αυτή την γλώσσα χτίζεται το κύκλωμα πύλη προς πύλη. Σε μια γραμμή του αρχείου ορίζεται μια πύλη μαζί με το qubit (ή qubits) που ενεργεί και έναν αριθμό αν πρόκειται για πύλη μετατόπισης φάσης. Οι πύλες πρέπει να εισάγονται με την σειρά που εμφανίζονται στο κύκλωμα γιατί με αυτήν την σειρά θα εκτελεστούν. Ακόμα ο χρήστης ορίζει πόσα qubits έχει το κύκλωμα και το αρχικό διάνυσμα κατάστασης.

Ακολουθούν οι συντακτικοί κανόνες της γλώσσας περιγραφής:

1. Στην πρώτη γραμμή του κώδικα πρέπει να γίνεται πάντα η δήλωση του αριθμού των qubits με την μορφή '**qubits: n**', όπου n ακέραιος.
2. Στην δεύτερη γραμμή ακολουθεί η αρχικοποίηση του διανύσματος κατάστασης εισόδου με τιμές 0 και 1 με την μορφή '**init_vector: b**', όπου b μια ακολουθία από 0,1 με το πιο αριστερά bit να αντιστοιχεί στο πιο σημαντικό qubit.
3. Στην συνέχεια σε κάθε γραμμή ορίζεται μια πύλη με τον συντακτικό της όνομα, τα qubits που ενεργεί και έναν επιπλέον αριθμό στις πύλες μετατόπισης φάσης. Η σύνταξη των πυλών φαίνεται παρακάτω:
    a. Οι πύλες ενός qubit που συντάσσονται με τη μορφή '**name: qn**', όπου name το όνομα της πύλης, το γράμμα q ως έχει και n ένας φυσικός αριθμός. Π.χ. 'H: q3', για μια πύλη hadamard στο qubit 3. Στον πίνακα 3.3 φαίνονται τα συντακτικά ονόματα.

Συντακτικό όνομα	**Πύλη**
H	Hadamard
X	Pauli-X
Y	Pauli-Y
Z	Pauli-Z
S	Phase Shift Gate S
T	Phase Shift Gate T
Rz	Phase Shift Gate
Rd	Phase Shift Gate
Rk	Phase Shift Gate

Πίνακας 3.3 – Συντακτικά ονόματα πυλών ενός qubit

Οι πύλες μετατόπισης φάσης είναι ειδική περίπτωση καθώς απαιτείται ένας ακόμα αριθμός για τον καθορισμό της γωνίας μετατόπισης. Η γενική μορφή είναι '**name: qn,km**', αλλά υπάρχουν 3 τρόποι εισαγωγής της γωνίας. Ο πρώτος είναι ο '**Rz: qn,km**' όπου n φυσικός και m είναι ένας πραγματικός αριθμός από το 0 ως το 1. Η γωνία υπολογίζεται ως $2 * \pi * m$ επομένως αν χρειάζεται μετατόπιση 45ο το m θα είναι 0.125. Ο δεύτερος τρόπος είναι '**Rd: qn,km**' όπου n φυσικός και m είναι ένας πραγματικός που δηλώνει ακριβώς την γωνία σε μοίρες. Ο τρίτος τρόπος είναι '**Rk: qn,km**' όπου n φυσικός και m είναι ένας ακέραιος που υπολογίζει την γωνία από τον τύπο $\frac{2\pi}{2^m}$. Επομένως για γωνία 45ο το m θα είναι 3. Σε όλες τις περιπτώσεις μπορεί να γίνει ορισμός αρνητικής γωνίας με την προσθήκη του συμβόλου μείον '-' πριν από το m.

b. Οι πύλες των δύο qubits συντάσσονται με τη μορφή '**name: qn,qm**', όπου name το όνομα της πύλης, το γράμμα q ως έχει και n,m φυσικοί αριθμοί. Στον πίνακα 3.4 φαίνονται τα συντακτικά ονόματα.

Συντακτικό όνομα	**Πύλη**
CNOT	Controlled NOT
CZ	Controlled Z
SWAP	Swap
CRz	Controlled Phase Shift Gate
CRd	Controlled Phase Shift Gate
CRk	Controlled Phase Shift Gate

Οι πύλες CRz, CRd και CRk είναι οι ελεγχόμενες περιπτώσεις των Rz, Rd και Rk αντίστοιχα. Απαιτούν έναν ακόμα αριθμό για την γωνία περιστροφής όπως και προηγουμένως. Έχουν την μορφή '**name: qn,qm,kp**', όπου n,m φυσικοί, q,k ως έχουν και το p ακολουθεί την μορφή του m των αντίστοιχων πυλών μετατόπισης φάσης του ενός qubit.

c. Οι πύλες των τριών qubits που συντάσσονται με τη μορφή '**name: qn,qm,qp**', όπου name το όνομα της πύλης, το γράμμα q ως έχει και n,m,p φυσικοί αριθμοί. Ακολουθεί πίνακας με τα συντακτικά ονόματα των πυλών,

Συντακτικό όνομα	**Πύλη**
Tof	Toffoli - CCNOT
Fred	Fredkin - CSWAP
CCRz	Controlled Controlled Phase Shift Gate
CCRd	Controlled Controlled Phase Shift Gate
CCRk	Controlled Controlled Phase Shift Gate

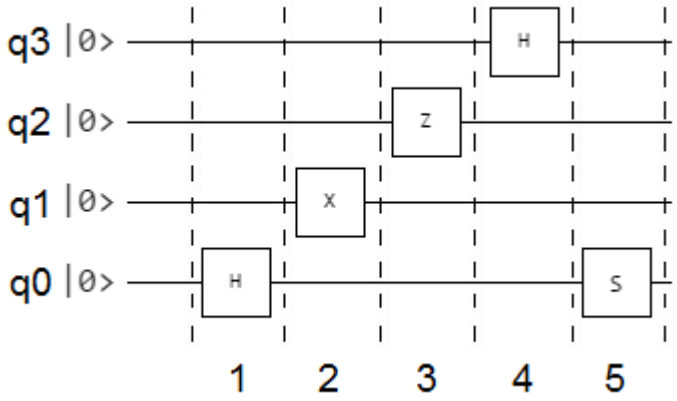
Οι πύλες CCRz, CCRd και CCRk είναι οι διπλά ελεγχόμενες περιπτώσεις των Rz, Rd και Rk αντίστοιχα. Απαιτούν έναν ακόμα αριθμό για την γωνία περιστροφής και έχουν την μορφή **'name: qn,qm,qp,ko**', όπου n,m,p φυσικοί, q,κ ως έχουν και το ο ακολουθεί τους κανόνες των m και p των προηγούμενων περιπτώσεων.

4. Οι γραμμές σχόλιων ξεκινούν με το χαρακτήρα '**#**' και αγνοούνται, όπως επίσης αγνοούνται και οι κενές γραμμές.
5. Τα κενά ή tabs στις γραμμές δεν έχουν σημασία.

Επιπλέον χαρακτηριστικό που εισάγει ο χρήστης είναι οι πύλες μέτρησης. Αυτές εισάγονται ξεχωριστά από τις υπόλοιπες κι έχουν κάποιους επιπλέον δικούς τους κανόνες.

1. Ο ορισμός τους γίνεται στο τέλος του αρχείου, αφού έχουν εισαχθεί όλες οι κανονικές πύλες.
2. Ο ορισμός ξεκινάει με την γραμμή ***Measurement: n*** , όπου n ο αριθμός των πυλών μέτρησης που θα εισαχθούν.
3. Ακολουθούν οι πύλες που εισάγονται μία προς μία με την μορφή ***Measure: gm*** , όπου g όπως είναι και m ο αριθμός του qubit στο οποίο μπαίνει η πύλη μέτρησης.
4. Οι πύλες πρέπει να ορίζονται με την σειρά από το λιγότερο στο περισσότερο σημαντικό qubit.
5. Ο ορισμός τελειώνει με ***End_measurement***.
6. Αν δεν εισάγονται πύλες μέτρησης δεν εισάγεται τίποτα από τα παραπάνω.

Ακολουθεί ένα παράδειγμα για το ακόλουθο κβαντικό κύκλωμα:

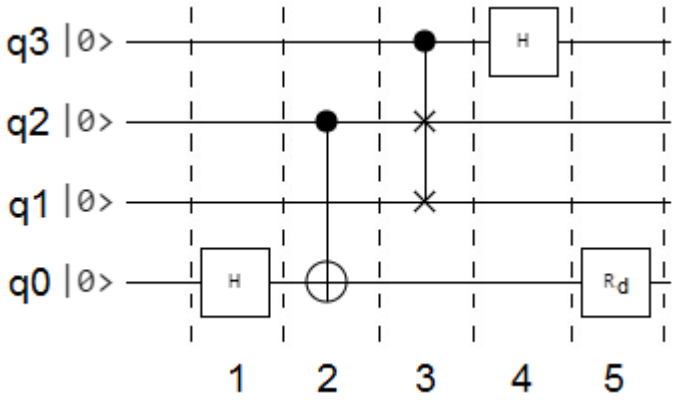


Ο κώδικας που αποθηκεύεται σε ένα αρχείο κειμένου:

```
qubits: 4
init_vector: 0000

H: q0
X: q1
Z: q2
H: q3
S: q0
```

Και ένα πιο σύνθετο παράδειγμα για το κύκλωμα:



Ο κώδικας που αποθηκεύεται σε ένα αρχείο κειμένου:

```
qubits: 4
init_vector: 0000

H: q0
CNOT: q2, q0
Fred: q3, q2, q1
H: q3
Rd: q0, k45.0
```

Αν σε αυτό το παράδειγμα θέλουμε να εισάγουμε δύο πύλες μέτρησης στα qubit 1 και 3 τότε προστίθεται το παρακάτω κομμάτι κώδικα.

```
Measurement: 2
Measure: g1
Measure: g3
End_measurement
```

3.2 Υλοποίηση

Σε αυτήν την ενότητα θα γίνει πιο αναλυτική περιγραφή της υλοποίησης με βάση τις προηγούμενες ιδέες και επιλογές.

Αρχικοποίηση και έλεγχος: Σε πρώτη φάση στο πρόγραμμα γίνεται ο ορισμός global μεταβλητών που θα χρησιμοποιηθούν σε όλη την έκταση του κώδικα. Επίσης γίνονται κάποιοι έλεγχοι όπως για παράδειγμα αν υπάρχει διαθέσιμη κάρτα γραφικών, ποιος είναι ο αριθμός computation capability της κάρτας και υπολογίζεται μέχρι πόσα qubits μπορεί να γίνει προσομοίωση με βάση την μνήμη που υπάρχει διαθέσιμη.

Είσοδος αρχείου από τον χρήστη και διάβασμα πρώτων γραμμών: Στην εντολή εκτέλεσης του προγράμματος ο χρήστης βάζει σαν όρισμα το αρχείο περιγραφής του κυκλώματος, με το διακριτικό **-i**. Για παράδειγμα αν το εκτελέσιμο έχει το όνομα a.out και το αρχείο το όνομα qft.txt, η εκτέλεση γίνεται με την εντολή ***./a.out –i qft.txt***. Το αρχείο ανοίγει και διαβάζεται από μια συνάρτηση. Το πρώτο πράγμα που θα διαβαστεί είναι τα qubits του κυκλώματος. Αν ο αριθμός ξεπερνάει το όριο που υπολογίστηκε προηγουμένως η διαδικασία τελειώνει με σχετικό μήνυμα. Διαφορετικά συνεχίζει με την ανάγνωση της ακολουθίας των bits του διανύσματος κατάστασης εισόδου και μετατροπή σε ακέραιο αριθμό. Έχοντας γνώση πλέον του αριθμού των qubits και συνεπώς της μνήμης που απαιτείται γίνεται δέσμευση ενός διανύσματος 2^n στην μεριά του host και δύο ισομεγεθών διανυσμάτων στο device. Επίσης δημιουργείται το διάνυσμα κατάστασης που αρχικά είναι μηδενικό εκτός από μια μονάδα που βρίσκεται στην θέση του ακεραίου που βρέθηκε πριν. Το διάνυσμα αντιγράφεται σε ένα από τα δύο του device.

Εκτέλεση: Η εκτέλεση κώδικα στην κάρτα γραφικών γίνεται με την κλήση του kernel, που είναι μια ειδική συνάρτηση. Εδώ επιλέγεται η συνάρτηση αυτή να εκτελέσει ένα βήμα όπως περιγράφτηκε προηγουμένως, δηλαδή το γινόμενο του μητρώου επί το διάνυσμα κατάστασης. Η κλήση του kernel καθορίζει με άλλα λόγια την επίδραση μιας πύλης στην κατάσταση του κυκλώματος. Επομένως η συνολική εκτέλεση είναι μια επαναληπτική διαδικασία που καλεί τον kernel τόσες φορές όσες είναι και οι πύλες του κυκλώματος. Στο πρόγραμμα γίνεται η ανάγνωση μιας γραμμής του αρχείου εισόδου και με βάση αυτή περνάνε οι απαραίτητες πληροφορίες σαν παράμετροι στην κλήση του kernel. Τα δύο διανύσματα που έχουν δεσμευτεί χρησιμοποιούνται εναλλάξ, ως διάνυσμα πολλαπλασιασμού και διάνυσμα αποθήκευσης του αποτελέσματος της πράξης.

Οι πράξεις που εκτελούνται κατά την εκτέλεση όπως περιγράφτηκε σε προηγούμενο κεφάλαιο είναι δύο πολλαπλασιασμοί και μια πρόσθεση ανά thread. Όμως τα μητρώα των κβαντικών πυλών περιέχουν κυρίως 0 και 1 με τα οποία οι πράξεις πολλαπλασιασμού δεν έχουν νόημα. Επομένως για αποφυγή αυτών των πράξεων και βελτίωση της απόδοσης, ο κώδικας εξειδικεύεται για κάθε πύλη. Για παράδειγμα στην πύλη Pauli-X στην πραγματικότητα δεν γίνονται πράξεις παρά μόνο μετακινήσεις στην μνήμη, ενώ η πύλη Hadamard είναι η μόνη για την οποία γίνονται όλες οι πράξεις. Στις πύλες S,T και Rz γίνονται μόνο οι πολλαπλασιασμοί που αφορούν το στοιχείο του πίνακα που περιγράφει την γωνία.

Κάθε thread έχει ένα ξεχωριστό ID με βάση το οποίο κατανέμονται οι πράξεις. Όταν δημιουργούνται η αρίθμηση ξεκινάει από το 0 διευκολύνοντας την συσχέτισή τους με την αρίθμηση των διανυσμάτων που ξεκινάει επίσης από το 0. Όπως Σε συνέχεια του

προηγούμενου παραδείγματος, παρακάτω δείχνεται το ID του thread αντιστοιχεί στην γραμμή του μητρώου.

$$\begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} thread\_0 \\ thread\_1 \\ thread\_2 \\ thread\_3 \\ thread\_4 \\ thread\_5 \\ thread\_6 \\ thread\_7 \end{matrix}$$

Επίσης είδαμε ότι οι θέσεις των a, b, c, d καθορίζονται από το αν το q-ιοστό bit της δυαδικής αναπαράστασης του αριθμού της γραμμής είναι 0 ή 1 όπου q το qubit που ενεργεί η πύλη. Γι΄αυτό τον λόγο μια διακλάδωση ***if*** χωρίζει την εκτέλεση για την περίπτωση του 0 και του 1, μοιράζοντας τα thread μισά στην κατεύθυνση του 0 και τα άλλα μισά στο 1, όπως φαίνεται παρακάτω.

Για 0 στο qubit 1:

$$\begin{matrix} 0\mathbf{0}0 \\ 0\mathbf{0}1 \\ 010 \\ 011 \\ 1\mathbf{0}0 \\ 1\mathbf{0}1 \\ 110 \\ 111 \end{matrix} \begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \boldsymbol{thread\_0} \\ \boldsymbol{thread\_1} \\ thread\_2 \\ thread\_3 \\ \boldsymbol{thread\_4} \\ \boldsymbol{thread\_5} \\ thread\_6 \\ thread\_7 \end{matrix}$$

Για 1 στο qubit 1:

$$\begin{matrix} 000 \\ 001 \\ 0\mathbf{1}0 \\ 0\mathbf{1}1 \\ 100 \\ 101 \\ 1\mathbf{1}0 \\ 1\mathbf{1}1 \end{matrix} \begin{bmatrix} a & 0 & b & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 & 0 & 0 \\ c & 0 & d & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & c & 0 & d & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ c \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} thread\_0 \\ thread\_1 \\ \boldsymbol{thread\_2} \\ \boldsymbol{thread\_3} \\ thread\_4 \\ thread\_5 \\ \boldsymbol{thread\_6} \\ \boldsymbol{thread\_7} \end{matrix}$$

Επομένως τα μισά thread κάνουν τις πράξεις με τα στοιχεία a, b και τα υπόλοιπα με τα c, d. Ένα παράδειγμα για τον κώδικα εκτέλεσης των πυλών Hadamard και Pauli-X φαίνεται παρακάτω:

```
if (q-bit_of_ID == 0)
{
    if (gate == 'Hadamard')
    {
        vector2[ i ] = (1/√2)*vector1[ i ] + (1/√2)*vector1[i + 2q];
    }
    if (gate == 'X')
{
        vector2[ i ] = vector1[i + 2q];
    }
}
else (q-bit_of_ID == 1)

{
    if (gate == 'Hadamard')
    {
        vector2[ i ] = (1/√2)*vector1[i - 2q] - (1/√2)*vector1[ i ];
    }
    if (gate == 'X')
{
        vector2[ i ] = vector1[i - 2q];
    }
}
```

Η μεταβλητή q-bit_of_ID αναφέρεται στο q-ιστό bit της δυαδικής αναπαράστασης του ID του thread. Η gate αναφέρεται προφανώς στην πύλη. Τα vector2 και vector1 είναι τα δύο διανύσματα που έχουν δεσμευτεί στο device. Το i είναι το ID των thread και q είναι το qubit που βρίσκεται η πύλη.

Από την αρίθμηση των διανυσμάτων φαίνεται ότι από το ID και το qubit υπολογίζονται οι θέσεις των στοιχείων χωρίς να χρειάζεται το μητρώο να είναι αποθηκευμένο, όπως φαίνεται στο παράδειγμα.

Στις πύλες των 2 και 3 qubits, ελέγχονται επιπλέον οι θέσεις των bits στην δυαδική αναπαράσταση του ID που αντιστοιχούν στα επιπλέον ένα ή δύο qubits και με βάση αυτά θα γίνουν οι πράξεις. Για παράδειγμα στην πύλη CNOT θα ελεγχθεί το δεύτερο qubit και αν αυτό είναι 1 θα γίνει η μεταφορά στην μνήμη που γίνεται και στην πύλη Pauli-X.

Μεταφορά αποτελέσματος στον host: Σε όλη την διάρκεια της εκτέλεσης όλα τα δεδομένα παραμένουν στην κάρτα. Μετά την ολοκλήρωση της, το τελικό διάνυσμα κατάστασης αντιγράφεται στην μνήμη του host που έχει δεσμευτεί γι' αυτόν τον σκοπό.

Εγγραφή αποτελεσμάτων σε αρχείο: Τώρα που τα δεδομένα βρίσκονται στον host το μόνο που απομένει είναι η εγγραφή τους σε αρχείο ώστε να μπορούν να διαβαστούν από τον χρήστη. Η εγγραφή μπορεί να ενεργοποιηθεί βάζοντας σαν παράμετρο στην εντολή εκτέλεσης **–v** και το όνομα του αρχείου εξόδου. Για παράδειγμα ***./a.out –i qft.txt –v output*** είναι η εντολή που θα τρέξει με αρχείο εισόδου το qft.txt και θα αποθηκεύσει το διάνυσμα κατάστασης στο αρχείο output.txt. Το αρχείο εισόδου πρέπει να μπαίνει πάντα πριν από αυτό της εξόδου. Αν δεν γραφτεί όρισμα –v τότε δεν γίνεται εγγραφή στην έξοδο.

3.3 Υλοποίηση πυλών μέτρησης

Η υλοποίηση των πυλών μέτρησης είναι ανεξάρτητη και παραβλέπεται αν δεν εισάγονται τέτοιες πύλες στο αρχείο του χρήστη.

Από την μέχρι τώρα υλοποίηση έχουμε ένα διάνυσμα που περιέχει τα πλάτη πιθανότητας όλων των δυνατών καταστάσεων του καταχωρητή. Αυτό που θέλουμε ουσιαστικά με τις πύλες μέτρησης είναι να μετατρέψουμε τα πλάτη σε πιθανότητες και να τις προσθέσουμε με βάση των παραπάνω τύπο δημιουργώντας ένα μικρότερο διάνυσμα.

Αρχικά όταν διαβάζεται το αρχείο εισόδου αποθηκεύεται ένα διάνυσμα με τις θέσεις των πυλών. Αυτό μεταφέρεται στο device γιατί θα χρησιμοποιείται από τα threads. Ακόμα δεσμεύονται δύο διανύσματα 2^M θέσεων όπου M ο αριθμός των πυλών μέτρησης για την αποθήκευση στον host και το device των επιθυμητών πιθανοτήτων.

Μετά την ολοκλήρωση της δημιουργίας του αρχικού διανύσματος, σειρά έχει η δημιουργία του νέου που περιέχει τις πιθανότητες. Αφού πρόκειται για διαφορετική λειτουργία δημιουργείται ένας διαφορετικός kernel, ο οποίος καλείται μόνο όταν υπάρχουν πύλες μέτρησης. Σε αυτή τη νέα συνάρτηση kernel δημιουργούνται τόσα threads όσα είχαμε και πριν, δηλαδή ένα thread για κάθε στοιχείο του διανύσματος. Κάθε thread συγκρίνει τις θέσεις που αποθηκεύτηκαν πριν με τα αντίστοιχα bit της δεκαδικής αναπαράστασης του ID του και παράγει την θέση που θα αποθηκεύσει την πιθανότητα. Για παράδειγμα αν έχουμε 5 qubits και πύλες στα 1 και 3 όπως στο προηγούμενο παράδειγμα, τότε το thread με ID 0 παράγει τον αριθμό 0 (0**0**0**0**0), ενώ το thread με ID 8 παράγει τον αριθμό 2 (0**1**0**0**0). Είναι δηλαδή η δυαδική αναπαράσταση αν αφαιρεθούν τα bit που δεν έχουν πύλες. Αυτήν την τιμή χρησιμοποιούν τα threads ως δείκτη για την αποθήκευση τις πιθανότητας που υπολογίζεται στο δεσμευμένο διάνυσμα.

Στο σχήμα 3.3 φαίνεται ένα παράδειγμα για 5 qubits με πύλες μέτρησης στα qubits 0 και 2. Οι γραμμές από το ένα διάνυσμα στο άλλο εκτός από μεταφορά κρύβουν και μετατροπή του πλάτους πιθανότητας σε πιθανότητα με ύψωση του μέτρο στο τετράγωνο. Ένα θέμα που προκύπτει όπως φαίνεται και στο σχήμα είναι ότι διαφορετικά threads γράφουν στην ίδια θέση μνήμης. Συγκεκριμένα προσθέτουν μια τιμή στην υπάρχουσα. Αυτό δημιουργεί race conditions στα threads καθώς μπορεί κάποιο να διαβάσει την τιμή πριν την αλλάξει κάποιο άλλο, να προσθέσει την δική του και να υπάρξει λάθος. Επομένως απαιτείται συγχρονισμός. Στο μοντέλο της CUDA δεν υπάρχει global συγχρονισμός των threads. Για την υλοποίηση επιλέγεται η χρήση ατομικής πράξης πρόσθεσης που σημαίνει ότι ένα thread διαβάζει-προσθέτει-γράφει σε μια θέση μνήμης χωρίς να επιτρέπεται σε κάποιο άλλο να κάνει χρήση αυτής. Αυτό δημιουργεί πρόβλημα σειριοποίησης και ειδικά όσο πιο λίγες πύλες μέτρησης τόσο πιο μεγάλη καθυστέρηση, αλλά διασφαλίζει την σωστή λειτουργία.

Για να γίνει εγγραφή αρχείου με τις πιθανότητες που υπολογίζονται από τις πύλες μέτρησης πρέπει να περιληφθεί όρισμα **–m** και το όνομα του αρχείου εξόδου στην εντολή εκτέλεσης. Τα ορίσματα –v και –m μπορούν να μπουν με όποια σειρά, αλλά μετά από το –i.

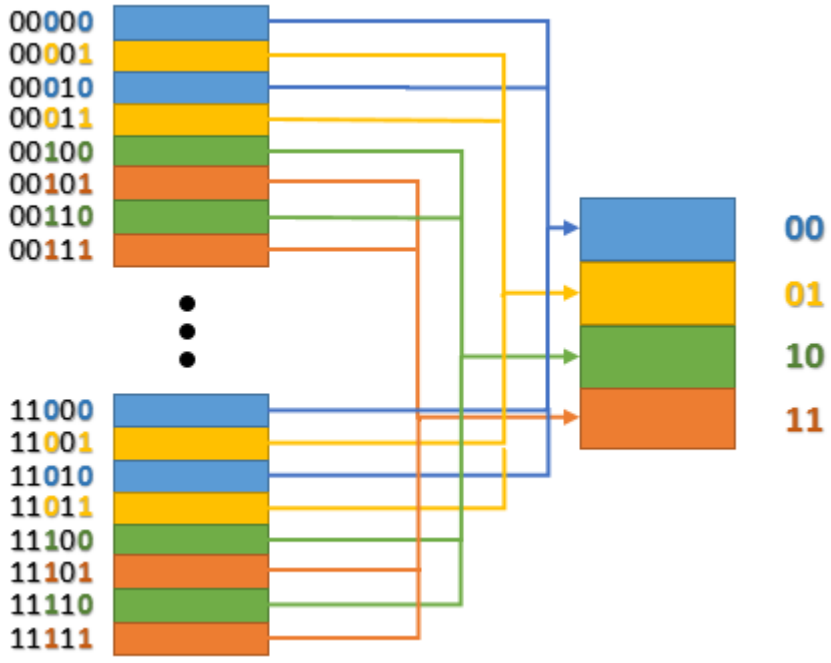
3.4 Ανάλυση Υλοποίησης

Σε αυτό το κομμάτι γίνεται θεωρητική ανάλυση των θετικών και αρνητικών στοιχείων αυτής της υλοποίησης.

Στα θετικά συγκαταλέγεται η πολύ καλή διαχείριση μνήμης που όπως είδαμε με την μη αποθήκευση του μητρώου του γινομένου Kronecker κερδίζεται τουλάχιστον $2^n x 2^n x 8$ bytes. Πιθανά να υπήρχε η περίπτωση να αποθηκευτούν ενδιάμεσα αποτελέσματα κατά τον υπολογισμό του μητρώου στην global memory αν αυτά ήταν μεγάλα για να χωρέσουν σε πιο γρήγορες μνήμες τις ιεραρχίας, και συνεπώς να αυξανόταν ο απαιτούμενος χώρος. Ένα ακόμα θετικό στοιχείο είναι η αποφυγή περιττών πράξεων. Αποφεύγονται οι πράξεις δημιουργίας του μητρώου και επίσης οι πράξεις πολλαπλασιασμού με 0 και 1 κατά τον υπολογισμό του διανύσματος κατάστασης.

Το μεγάλο αρνητικό αυτής της υλοποίησης είναι η ύπαρξη διακλάδωσης που χωρίζει τα threads σε δύο μονοπάτια. Όπως έχει ήδη ειπωθεί αν μέσα σε ένα warp υπάρχουν threads που εκτελούν διαφορετικές πράξεις τότε η εκτέλεση τους σειριοποιείται. Αυτό συμβαίνει και σε αυτήν την περίπτωση. Όμως το warp αποτελείται από 32 threads που σημαίνει ότι από 64 threads και πάνω δηλαδή από 6 qubits και πάνω ο διαχωρισμός γεμίζει τα warps που σημαίνει ότι δεν υπάρχει σειριοποίηση. Επομένως μέχρι τα 5 qubits μπορεί να μην υπάρχει η καλύτερη δυνατή απόδοση, αλλά πρόκειται για μικρές προσομοιώσεις με πολύ μικρούς χρόνους εκτέλεσης. Σε μεγάλες προσομοιώσεις δεν προκύπτει πρόβλημα. Υπάρχουν και οι περιπτώσεις των πυλών με 2 και 3 qubits που γίνεται περαιτέρω διαχωρισμός σε μονοπάτια εκτέλεσης. Έτσι για πύλες των 2 δεν υπάρχει πρόβλημα για 128 threads ή 7 qubits και πάνω, ενώ για πύλες των 3 για 256 threads ή 8 qubits και πάνω. Επίσης η διακλάδωση που επιλέγει την εκτέλεση συγκεκριμένης πύλης δεν αποτελεί πρόβλημα επειδή όλα τα threads ακολουθούν το ίδιο μονοπάτι εκτέλεσης.

Μερικά ακόμα αρνητικά είναι μερικές πράξεις που γίνονται για την εύρεση του bit της δυαδικής μορφής του ID, ο υπολογισμός του δείκτη των διανυσμάτων που περιλαμβάνει ύψωση σε δύναμη του 2 πράγμα που σημαίνει ότι μπορεί να γίνει με ολίσθηση και τέλος οι αριθμοί που επεξεργαζόμαστε είναι μιγαδικοί που σημαίνει ότι καλούνται συναρτήσεις για την επεξεργασία τους και ότι απαιτούνται διπλές πράξεις, καθώς υπάρχει το πραγματικό και το φανταστικό μέρος.



Σχήμα 3.3 - Παράδειγμα 5 qubits με πύλες μέτρησης στα 0 και 2

4. ΜΕΤΡΗΣΕΙΣ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα μετρήσεων απόδοσης για την εκτέλεση διαφόρων κβαντικών κυκλωμάτων στον προσομοιωτή. Σκοπός είναι να γίνει φανερό αν η χρήση της κάρτας γραφικών παρέχει αποτελέσματα που να επιβεβαιώνουν την επιλογή αυτή σαν αποδοτική, με βάση τα απαιτούμενα που ορίστηκαν πριν την σχεδίαση, και σε σύγκριση με σειριακή εκτέλεση. Επομένως για την σύγκριση αυτή δημιουργήθηκε επιπλέον μια άλλη έκδοση του προσομοιωτή που λειτουργεί σειριακά σε CPU χρησιμοποιώντας μόνο ένα thread που είναι και η χειρότερη περίπτωση. Για την προσομοίωση επιλέγονται κυκλώματα που είναι γνωστά και χρήσιμα στους κβαντικούς υπολογιστές. Αυτά είναι κυκλώματα κβαντικού μετασχηματισμού Fourier (QFT), διάφοροι αθροιστές και κυκλώματα για την εφαρμογή του αλγόριθμου του Shor.

Τα κυκλώματα που προσομοιώνονται είναι γραμμένα με την γλώσσα περιγραφής που παρουσιάστηκε. Κάποια από αυτά όμως είναι πολύ μεγάλα σε έκταση και διαθέτουν χιλιάδες πύλες, πράγμα που κάνει την δημιουργία τους με το χέρι υπερβολικά δύσκολη αν όχι αδύνατη. Για τον λόγο αυτό δημιουργήθηκαν γεννήτριες στην Matlab, οι οποίες παράγουν κυκλώματα όπως για παράδειγμα QFT και Shor για οποιοδήποτε αριθμό από qubits. Αυτό είναι δυνατό γιατί τέτοια κυκλώματα παρουσιάζουν μια κανονικότητα και επαναληψιμότητα που επιτρέπουν την αυτοματοποίηση της παραγωγής τους.

4.1 Το σύστημα που χρησιμοποιείται για την προσομοίωση

Η διαθέσιμη κάρτα γραφικών που χρησιμοποιήθηκε τόσο κατά την ανάπτυξη όσο και τις προσομοιώσεις είναι η ***'Tesla K20c'***. Τα τεχνικά της χαρακτηριστικά περιγράφονται παρακάτω.

- Architecture: Kepler
- CUDA Driver Version / Runtime Version 6.5 / 6.5
- CUDA Capability Major/Minor version number: 3.5
- Total amount of global memory: 4800 MBytes (5032706048 bytes)
- (13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
- GPU Clock rate: 706 MHz (0.71 GHz)
- Memory Clock rate: 2600 MHz
- Memory Bus Width: 320-bit
- L2 Cache Size: 1310720 bytes
- Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
- Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
- Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
- Total amount of constant memory: 65536 bytes

- Total amount of shared memory per block: 49152 bytes
- Total number of registers available per block: 65536
- Warp size: 32
- Maximum number of threads per multiprocessor: 2048
- Maximum number of threads per block: 1024
- Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
- Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
- Maximum memory pitch: 2147483647 bytes
- Texture alignment: 512 bytes
- Concurrent copy and kernel execution: Yes with 2 copy engine(s)

Ο έλεγχος για το bandwidth της μνήμης της κάρτας δίνει τα αποτελέσματα:

- Host to Device Bandwidth

Transfer Size: 33554432 Bytes

Bandwidth: 5858.9 MB/s

- Device to Host Bandwidth

Transfer Size: 33554432 Bytes

Bandwidth: 6558.1 MB/s

- Device to Device Bandwidth

Transfer Size: 33554432 Bytes

Bandwidth: 146949.1 MB/s

Το σύστημα πάνω στο οποίο βρίσκεται η κάρτα και στο οποίο γίνεται η σειριακή εκτέλεση διαθέτει ως κεντρική μονάδα επεξεργασίας τον ***Intel Core i7-3970X*** ο οποίος είναι χρονισμένος στα 3.5 GHz και διαθέτει κρυφή μνήμη μεγέθους 15 MB.

4.2 Προσομοίωση κυκλωμάτων κβαντικού μετασχηματισμού Fourier QFT

Ο QFT είναι πολύ χρήσιμος καθώς αποτελεί βάση για πολλούς κβαντικούς αλγόριθμους. Επομένως είναι ένα καλό παράδειγμα για την δοκιμή του προσομοιωτή. Μπορεί να εφαρμοστεί για οποιοδήποτε αριθμό qubits και έχει μικρό αριθμό πυλών της τάξης $O(n^2)$, όπου n ο αριθμός των qubits. Στην συνέχεια θα παρουσιαστούν μετρήσεις για κυκλώματα QFT διαφόρων qubits.

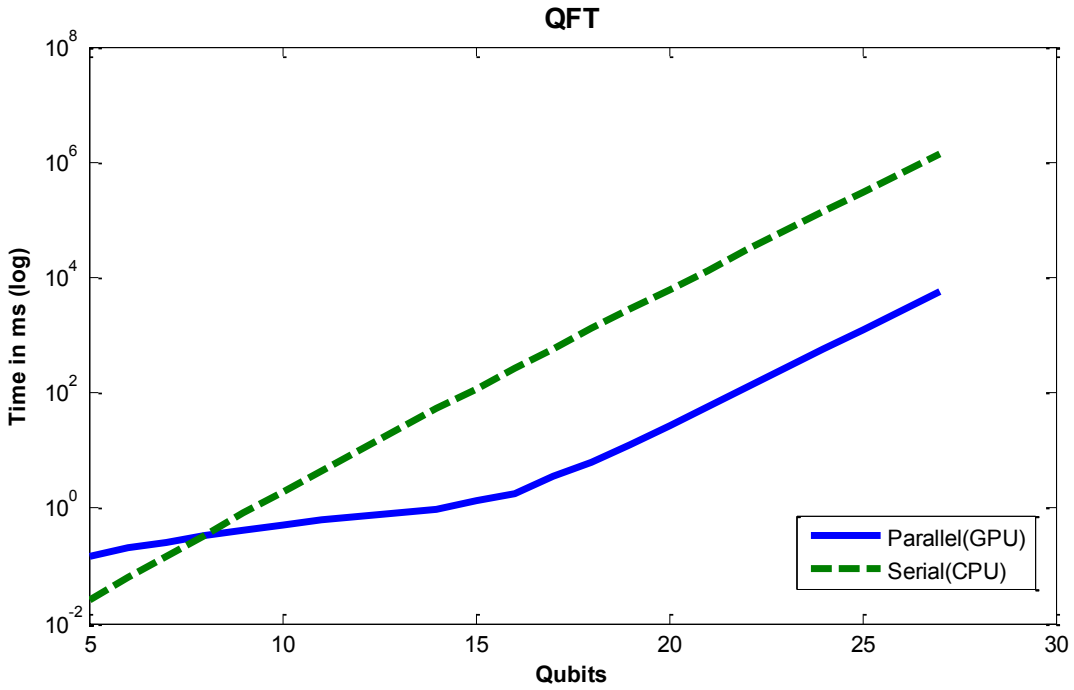
Για την μέτρηση του χρόνου εκτέλεσης χρησιμοποιούνται συναρτήσεις της CUDA στην παράλληλη εκτέλεση και στην σειριακή η συνάρτηση *clock()*. Το κομμάτι που

χρονομετρείται είναι αυτό που περιέχει αποκλειστικά τις πράξεις και τις μετακινήσεις των τιμών από και προς την μνήμη. Αυτό σημαίνει ότι στην παράλληλη εκτέλεση χρονομετρούνται οι εκτελέσεις του kernel και στην σειριακή η αντίστοιχη συνάρτηση που παίρνει την θέση του. Τέλος οι μετρήσεις αποτελούν μέσο όρο για εκτέλεση 100 φορών.

Πίνακας 4.1 - Χρόνοι μετρήσεων κυκλωμάτων QFT

Κύκλωμα	GPU	CPU
QFT 5 qubits, 15 gates	0.148614 ms	0.02502 ms
QFT 6 qubits, 21 gates	0.210001 ms	0.06152 ms
QFT 7 qubits, 28 gates	0.261975 ms	0.14404 ms
QFT 8 qubits, 36 gates	0.332199 ms	0.33961 ms
QFT 9 qubits, 45 gates	0.412197 ms	0.80095 ms
QFT 10 qubits, 55 gates	0.52396 ms	1.8907 ms
QFT 11 qubits, 66 gates	0.628766 ms	4.37627 ms
QFT 12 qubits, 78 gates	0.719677 ms	10.0782 ms
QFT 13 qubits, 91 gates	0.818057 ms	23.0666 ms
QFT 14 qubits, 105 gates	0.971191 ms	52.2405 ms
QFT 15 qubits, 120 gates	1.38493 ms	117.145 ms
QFT 16 qubits, 136 gates	1.80536 ms	261.489 ms
QFT 17 qubits, 153 gates	3.54479 ms	580.461 ms
QFT 18 qubits, 171 gates	6.41769 ms	1282.88 ms
QFT 19 qubits, 190 gates	12.7143 ms	2818.36 ms
QFT 20 qubits, 210 gates	26.3353 ms	6188.41 ms
QFT 21 qubits, 231 gates	56.2257 ms	13527.1 ms
QFT 22 qubits, 253 gates	121.367 ms	29411.8 ms
QFT 23 qubits, 276 gates	262.759 ms	63669.6 ms
QFT 24 qubits, 300 gates	568.337 ms	137273 ms
QFT 25 qubits, 325 gates	1228.24 ms	295303 ms

QFT 26 qubits, 351 gates	2644.56 ms	634206 ms
QFT 27 qubits, 378 gates	5688.36 ms	1357180 ms



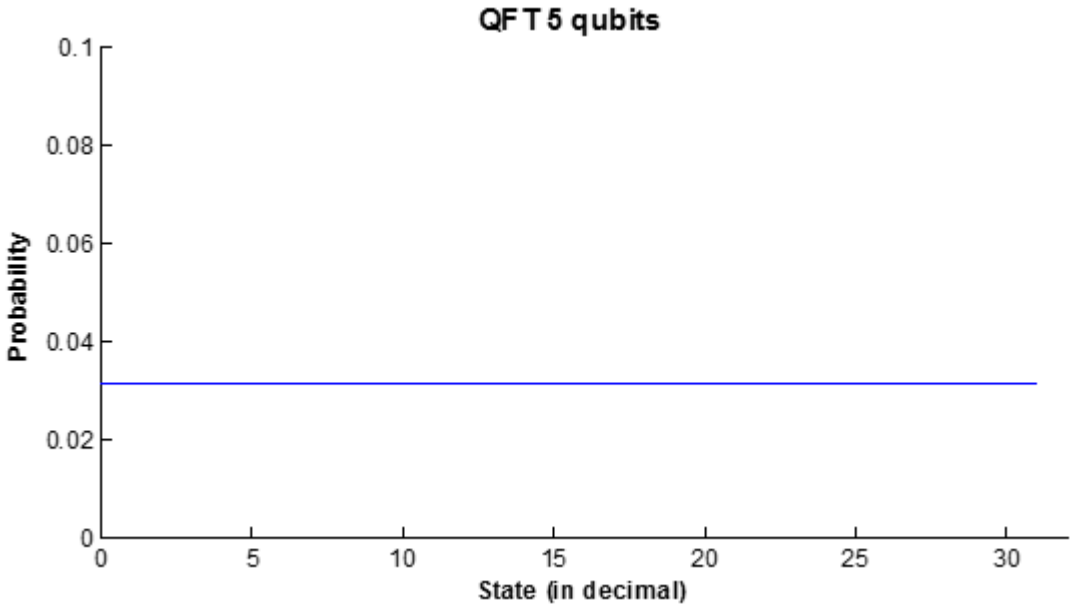
Σχήμα 4.1 - Γραφική απεικόνιση των χρόνων εκτέλεσης του QFT

Οι μετρήσεις πραγματοποιήθηκαν για όλα τα κυκλώματα QFT από 5 ως 27 qubits. Λιγότερα από 5 qubits παράγουν πολύ μικρά κυκλώματα χωρίς ενδιαφέρον στις μετρήσεις και τα 27 είναι το όριο που χωράει στην συγκεκριμένη κάρτα.

Βλέποντας τα δεδομένα που παρέχονται στον Πίνακα 4.1 και το Σχήμα 4.1, μπορούμε να καταλάβουμε ότι για μικρά κυκλώματα η εκτέλεση είναι πιο γρήγορη στην CPU. Αυτό ισχύει γενικά και στην συγκεκριμένη περίπτωση φαίνεται ότι μέχρι τα 7 qubits υπερέχει η CPU, στα 8 qubits οι χρόνοι είναι παραπλήσιοι και από κει και πέρα αυξάνεται μια μεγάλη διαφορά υπέρ της GPU που φτάνει τις 2 με 3 τάξεις μεγέθους. Επίσης παρατηρείται ότι μέχρι τα 15 qubits οι χρόνοι στην GPU αυξάνονται με μικρό ρυθμό και στην συνέχεια ο ρυθμός αλλάζει παρόμοια με αυτόν τις CPU, πράγμα που σημαίνει ότι σε εκείνο το σημείο το μέγεθος των κυκλωμάτων επηρεάζει πολύ την απόδοση. Από άποψη απόλυτων αριθμών, σε ένα κύκλωμα 27 qubits με περίπου 380 πύλες, η παράλληλη υλοποίηση χρειάζεται 5688.36 ms ≈ 5,7 δευτερόλεπτα έναντι 1357180 ms ≈ 22,6 λεπτά.

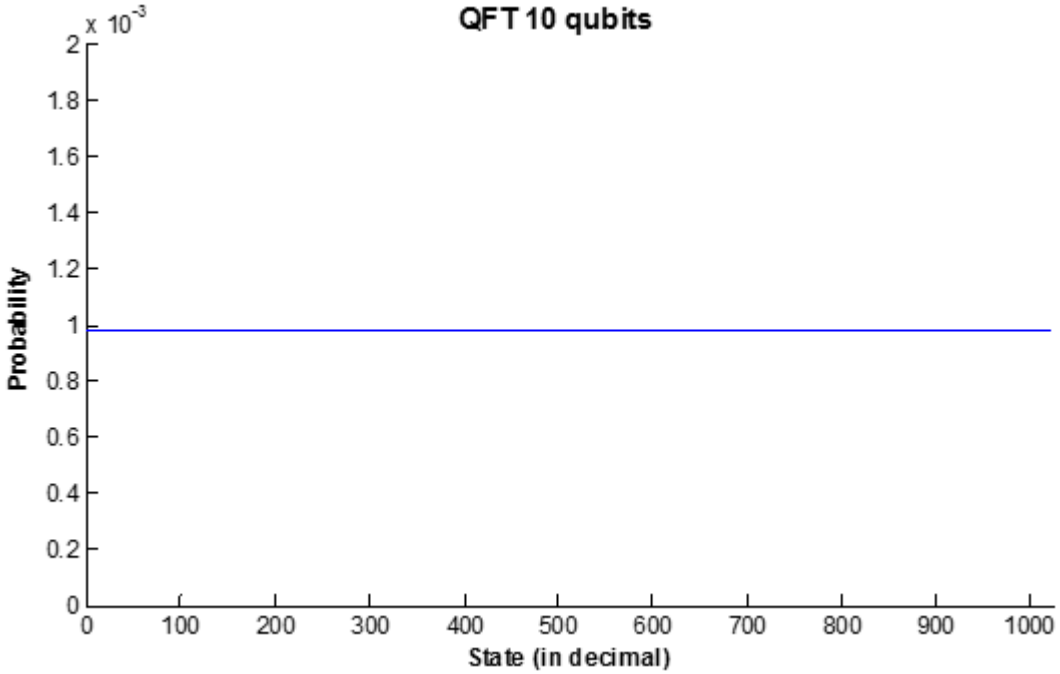
Η πιστοποίηση ορθής λειτουργίας της προσομοίωσης για τον QFT γίνεται εφαρμόζοντάς τον σε ένα αρχικό διάνυσμα κατάστασης της μορφής π.χ. 00001, δηλαδή χωρίς υπερθέσεις . Επειδή σε μία τέτοια περίπτωση ο QFT λειτουργεί σαν μια σειρά από πύλες Hadamard σε κάθε qubit, το αποτέλεσμα είναι ο ισοπίθανος διαμοιρασμός σε όλες τις πιθανές καταστάσεις.

Δοκιμάζοντας για 3 από τα παραπάνω κυκλώματα για 5, 10 και 20 qubits προκύπτουν τα παρακάτω αποτελέσματα.



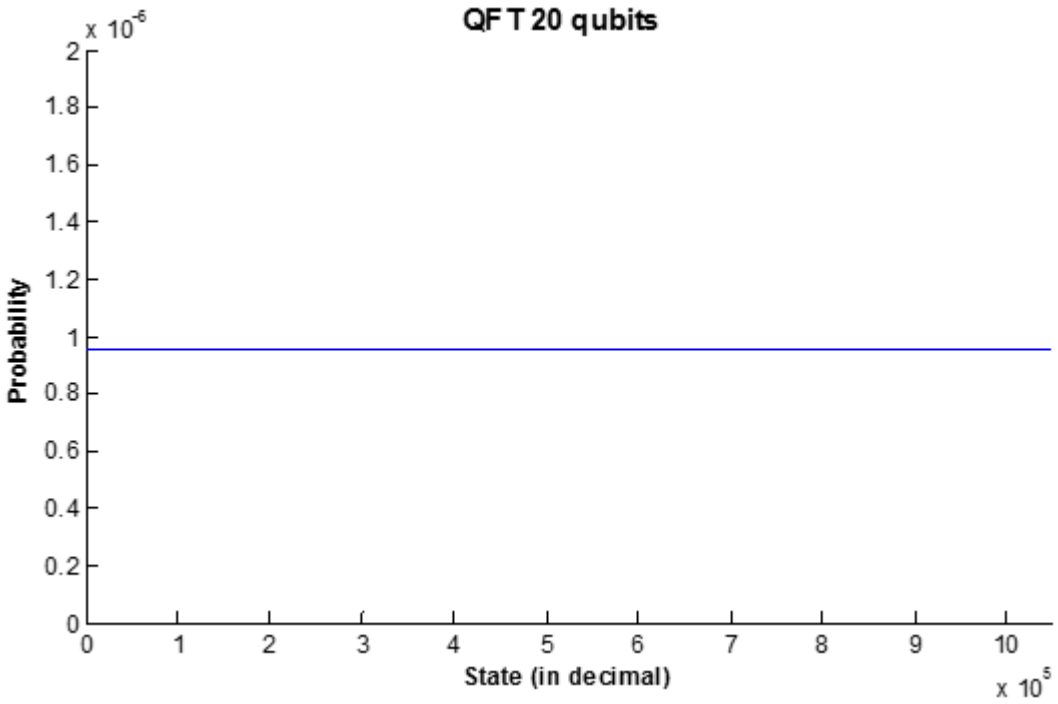
Σχήμα 4.2 – Πιθανότητες καταστάσεων για QFT 5 qubits

Για κύκλωμα QFT 5 qubits υπάρχουν 32 διαφορετικές καταστάσεις. Αν οι πιθανότητες μοιραστούν όμοια σε αυτές τότε καθεμία έχει πιθανότητα 1/32 = 0,03125.



Σχήμα 4.3 – Πιθανότητες καταστάσεων για QFT 10 qubits

Για κύκλωμα QFT 10 qubits υπάρχουν 1024 διαφορετικές καταστάσεις. Αν οι πιθανότητες μοιραστούν όμοια σε αυτές τότε καθεμία έχει πιθανότητα 1/1024 = $0{,}9765625*10^{3}$.



Σχήμα 4.4 – Πιθανότητες καταστάσεων για QFT 20 qubits

Για κύκλωμα QFT 20 qubits υπάρχουν 1048576 διαφορετικές καταστάσεις. Αν οι πιθανότητες μοιραστούν όμοια σε αυτές τότε καθεμία έχει πιθανότητα 1/1048576 = $0{,}9536743*10^{6}$.

Πέρα από τους χρόνους εκτέλεσης ενδιαφέρον έχει πως κατανέμονται στους πόρους της κάρτας οι εργασίες και ποια απόδοση επιτυγχάνεται. Η NVidia παρέχει ειδικό εργαλείο, τον NVProfiler που μεταξύ άλλων επιστρέφει διάφορες μετρικές από την εκτέλεση ενός προγράμματος. Στον Πίνακα 4.2 παρουσιάζονται ποιες μετρικές χρησιμοποιούνται και μια σύντομη περιγραφή τους.

Πίνακας 4.2 – Περιγραφή μετρικών

Όνομα Μετρικής	Περιγραφή
achieved_occupancy	Η αναλογία των ενεργών warps σε κάθε κύκλο προς τον μέγιστο αριθμό warps που υποστηρίζονται σε έναν multiprocessor(SM)
ipc	Εντολές που εκτελούνται ανά κύκλο
sm_efficiency	Το ποσοστό του χρόνου στον οποίο τουλάχιστον ένα warp είναι ενεργό σε έναν SM, υπολογισμένο κατά μέσο όρο όλων των SM της GPU

flop_count_sp	Ο αριθμός των πράξεων κινητής υποδιαστολής μονής ακρίβειας που εκτελούνται
warp_execution_efficiency	Η αναλογία του μέσου αριθμού threads ανά warp προς το μέγιστο αριθμό των threads ανά warp
stall_exec_dependency	Ποσοστό καθυστερήσεων που συμβαίνουν λόγω του ότι κάποια είσοδος μιας εντολής δεν είναι ακόμα διαθέσιμη
stall_memory_dependency	Ποσοστό καθυστερήσεων που συμβαίνουν όταν δεν μπορεί να γίνει μια λειτουργία μνήμης λόγω μη διαθεσιμότητας των πόρων ή επειδή υπάρχουν πολλά αιτήματα.
gld_throughput	Το throughput για φόρτωση από την Global Memory
gld_transactions	Ο αριθμός των συναλλαγών φόρτωσης με την Global Memory
gst_throughput	Το throughput για αποθήκευση στην Global Memory
gst_transactions	Ο αριθμός των συναλλαγών αποθήκευσης με την Global Memory
l2_read_throughput	Το throughput της cache μνήμης L2 για όλες τις αιτήσεις ανάγνωσης
l2_read_transactions	Ο αριθμός των συναλλαγών με την cache μνήμη L2 για όλες τις αιτήσεις ανάγνωσης
l2_write_throughput	Το throughput της cache μνήμης L2 για όλες τις αιτήσεις εγγραφής
l2_write_transactions	Ο αριθμός των συναλλαγών με την cache μνήμη L2 για όλες τις αιτήσεις εγγραφής

Οι μετρικές αυτές επιλέχτηκαν για να δώσουν μια εικόνα για το πόσο αξιοποιούνται οι πόροι της κάρτας, που υπάρχουν μεγάλες καθυστερήσεις και την ταχύτητα που γίνονται οι μετακινήσεις στις μνήμες της κάρτας κατά την διάρκεια της εκτέλεσης.

Στην συνέχεια παρουσιάζονται τα αποτελέσματα των μετρικών για τα κυκλώματα των 10,15,20 και 27 qubits.

Πίνακας 4.3 – Μετρικές για QFT 10 qubits

Όνομα Μετρικής	Μέση τιμή
achieved_occupancy	0.059239
ipc	0.158808
sm_efficiency	22.29%
flop_count_sp	14336
warp_execution_efficiency	69.38%
stall_exec_dependency	39.91%
stall_memory_dependency	12.35%
gld_throughput	2.1845GB/s
gld_transactions	256
gst_throughput	1.8615GB/s
gst_transactions	192
l2_read_throughput	3.9756GB/s
l2_read_transactions	1247
l2_write_throughput	1.9000GB/s
l2_write_transactions	771

Πίνακας 4.4 - Μετρικές για QFT 15 qubits

Όνομα Μετρικής	Μέση τιμή
achieved_occupancy	0.670544
ipc	1.083391
sm_efficiency	58.52%
flop_count_sp	458752
warp_execution_efficiency	77.59%
stall_exec_dependency	19.10%

stall_memory_dependency	18.21%
gld_throughput	47.031GB/s
gld_transactions	8192
gst_throughput	7.1265GB/s
gst_transactions	6144
l2_read_throughput	48.515GB/s
l2_read_transactions	33011
l2_write_throughput	42.083GB/s
l2_write_transactions	24582

Πίνακας 4.5 - Μετρικές για QFT 20 qubits

Όνομα Μετρικής	**Μέση τιμή**
achieved_occupancy	0.841414
ipc	2.196482
sm_efficiency	95.87%
flop_count_sp	196608
warp_execution_efficiency	82.37%
stall_exec_dependency	30.17%
stall_memory_dependency	47.58%
gld_throughput	80.809GB/s
gld_transactions	262144
gst_throughput	87.449GB/s
gst_transactions	196608
l2_read_throughput	87.541GB/s
l2_read_transactions	1048771
l2_write_throughput	81.197GB/s
l2_write_transactions	808066

Πίνακας 4.6 - Μετρικές για QFT 27 qubits

Όνομα Μετρικής	Μέση τιμή
achieved_occupancy	0.829604
ipc	2.138896
sm_efficiency	99.96%
flop_count_sp	1879048192
warp_execution_efficiency	86.53%
stall_exec_dependency	29.24%
stall_memory_dependency	51.28%
gld_throughput	86.172GB/s
gld_transactions	33554432
gst_throughput	79.339GB/s
gst_transactions	25165824
l2_read_throughput	86.191GB/s
l2_read_transactions	134221339
l2_write_throughput	81.672GB/s
l2_write_transactions	103243396

Από τις μετρήσεις φαίνεται ότι όσο μεγαλώνει το κύκλωμα βελτιώνονται οι μετρικές όσον αφορά εκμετάλλευση των πόρων αλλά όπως είναι λογικό εμφανίζονται περισσότερες καθυστερήσεις που αφορούν την αναμονή στην μνήμη ή στην εκτέλεση μιας εντολής. Ακόμα φαίνεται ότι σε μεγάλα κυκλώματα υπάρχει σχεδόν πάντα τουλάχιστον ένα warp σε λειτουργία όπως φαίνεται από το sm_efficiency, και σύμφωνα με το achieved_occupancy η κάρτα γεμίζει σε ικανοποιητικό βαθμό από warps. Τα warps δεν γεμίζουν πλήρως από threads που εκτελούν την ίδια εντολή λόγω των if όπως περιγράφτηκε σε προηγούμενο κεφάλαιο. Τέλος στην μετακινήσεις της μνήμης φαίνεται ότι η cache L2 είναι πιο συχνά χρησιμοποιούμενη το οποίο είναι θετικό επειδή συνήθως είναι και πιο γρήγορη.

4.3 Προσομοίωση κυκλωμάτων του αλγόριθμου Shor

Ο κβαντικός αλγόριθμος του Shor χρησιμοποιείται στην ανάλυση σύνθετων ακέραιων αριθμών σε γινόμενο πρώτων παραγόντων, με σπουδαία εφαρμογή στην περιοχή της

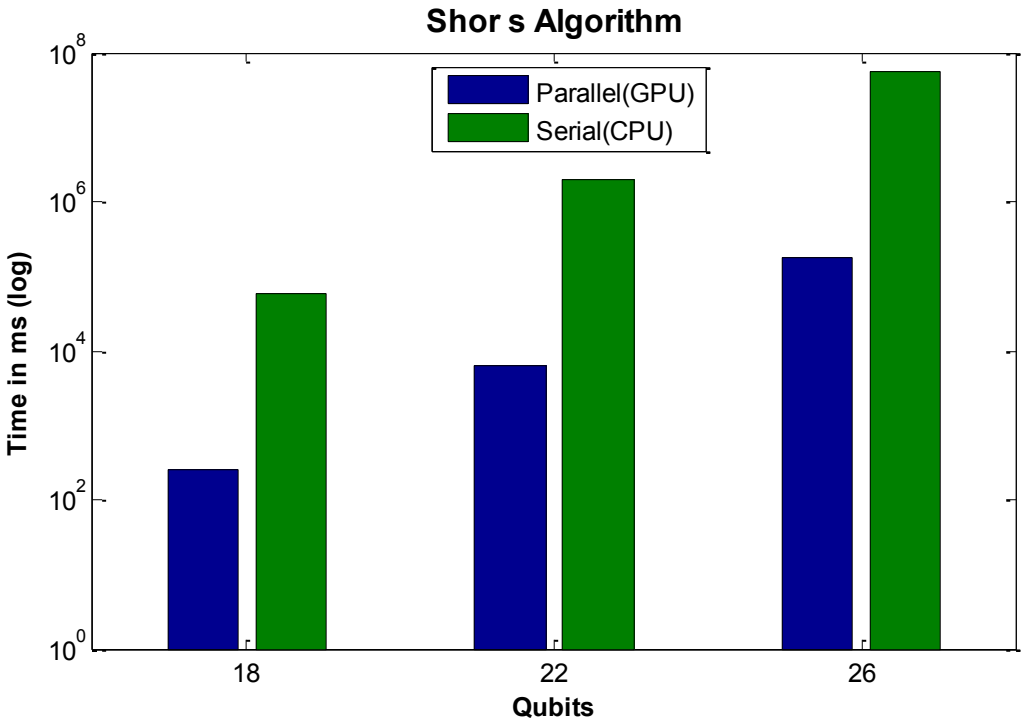
κρυπτανάλυσης και στην εύρεση της περιόδου περιοδικών συναρτήσεων. Ανήκει στην οικογένεια των αλγορίθμων κβαντικής εκτίμησης φάσης (Quantum Phase Estimation) που χρησιμοποιούνται μεταξύ άλλων και στην προσομοίωση κβαντικών φυσικών συστημάτων από κβαντικούς υπολογιστές. Τα κυκλώματα που δημιουργούνται για την εφαρμογή του αλγόριθμου είναι πολύ μεγάλα από άποψη αριθμού πυλών, της τάξης n^4 σε πολλές αρχιτεκτονικές, όπου n ο αριθμός των bits του ακεραίου που πρόκειται να παραγοντοποιηθεί, και περιέχουν άλλα υποκυκλώματα όπως αθροιστές και QFT.

Στη βιβλιογραφία υπάρχουν αρκετές αρχιτεκτονικές κυκλωμάτων για τον αλγόριθμο του Shor. Στις προσομοιώσεις χρησιμοποιήθηκε η αρχιτεκτονική του S.Beauregard για το μέρος του modular exponentiation που βασίζεται με τη σειρά της σε αθροιστές τύπου Draper και γίνεται εκτεταμένη χρήση ευθέων και αντίστροφων QFT για τον υπολογισμό αθροισμάτων και γινομένων. Η αρχιτεκτονική αυτή επιλέχθηκε γιατί η πλειοψηφία των χρησιμοποιούμενων πυλών είναι πύλες μετατόπισης φάσης (ελεγχόμενες και μη) οι οποίες απαιτούν πραγματικά υπολογισμούς αριθμών, σε αντίθεση με τις πύλες CNOT και Toffoli που χρησιμοποιούνται σε άλλες αρχιτεκτονικές. Οι πύλες CNOT και Toffoli δεν απαιτούν πραγματικές αριθμητικές πράξεις αφού τα στοιχεία των μητρώων τους είναι 0 ή 1 και επομένως προσομοιώνονται με απλή ανακατανομή του διανύσματος κατάστασης σε διαφορετικές θέσεις μνήμης.

Για τις μετρήσεις επιλέγονται κυκλώματα για παραγοντοποίηση των αριθμών 15,21,35. Αυτοί οι αριθμοί αντιπροσωπεύουν τα 3 διαφορετικά από άποψη qubits και πυλών κυκλώματα που μπορούν να δημιουργηθούν σε αυτή την GPU. Οποιοσδήποτε άλλος πρώτος αριθμός μέχρι το 63, το οποίο είναι το όριο, δημιουργεί κύκλωμα παρόμοιο με κάποιο από αυτά. Οι απαιτήσεις της αρχιτεκτονικής του Beauregard είναι 4n+2 qubits (όπου n ο αριθμός των bits του αριθμού προς παραγοντοποίησης). Δεν έχει χρησιμοποιηθεί η semiclassical υλοποίηση του τελικού QFT όπου σε αυτήν την περίπτωση θα χρειάζονταν 2n+3 qubits, και αυτό γιατί ο προσομοιωτής στην παρούσα φάση χειρίζεται τελικές και όχι ενδιάμεσες κβαντικές πύλες μέτρησης. Στην περίπτωση του semiclassical QFT όμως θα χρειαζόταν ένας μεγάλος αριθμός επαναλήψεων της κάθε προσομοίωσης ώστε να εξαχθεί η τελική στατιστική. Αντίθετα, στην παρούσα υλοποίηση εξάγεται απ' ευθείας η τελική κατανομή πιθανότητας.

Πίνακας 4.7 - Χρόνοι μετρήσεων κυκλωμάτων Shor

Κύκλωμα	GPU	CPU
Shor, N=15, 18 qubits, 6832 gates	245.526 ms	59973.2 ms
Shor, N=21, 22 qubits, 14080 gates	6278.2 ms	1914820 ms
Shor, N=35, 26 qubits, 25128 gates	177673 ms	>15 hours



Σχήμα 4.5 - Γραφική απεικόνιση των χρόνων εκτέλεσης του αλγόριθμου του Shor

Τα αποτελέσματα δείχνουν πάλι διαφορά στην απόδοση υπέρ της GPU. Επειδή τα κυκλώματα είναι μεγάλα ακόμα και για μικρούς αριθμούς δεν παρατηρείται στην αρχή η υπεροχή της CPU, όπως γινόταν στα μικρά κυκλώματα του QFT. Η διαφορά στην ταχύτητα φτάνει πάλι τις 2 με 3 τάξεις μεγέθους, όπως φαίνεται από την λογαριθμική κλίμακα της γραφικής παράστασης.

Στην συνέχεια γίνεται επαλήθευση της εξαγωγής σωστών αποτελεσμάτων. Ο αλγόριθμος του Shor, όπως ειπώθηκε, μπορεί να αναλύσει έναν αριθμό στο γινόμενο πρώτων παραγόντων του. Σαν είσοδο στον αλγόριθμο δίνεται ο αριθμός προς παραγοντοποίηση **N** και ένας τυχαίος ακέραιος 1<**a**<N. Αν ο a είναι σχετικά πρώτος με τον N, δηλαδή, ΜΚΔ{α, N} = 1, τότε ο αλγόριθμος παράγει την περίοδο **r** της συνάρτησης $f_{N,a}(x)=a^x \ (mod\ N)$, όπου x=0,1,2,... (Αν ο α δεν ήταν πρώτος σε σχέση με τον N τότε το ΜΚΔ{α, N} είναι παράγοντας του N και δε χρειάζεται να προχωρήσουμε). Οι παράγοντες του αριθμού δίνονται από P_1 = ΜΚΔ{$a^{r/2}$ -1,N} και P_2 = ΜΚΔ{$a^{r/2}$ +1,N}, με την προϋπόθεση ότι r είναι άρτιος και ισχύει ($a^{r/2}$+1 mod N) ≠ 0. Στον προσομοιωτή μπορεί να γίνει εξαγωγή της περιόδου r που μας ενδιαφέρει βάζοντας πύλες μέτρησης στα κατάλληλα qubits. Αυτό παράγει ένα αρχείο με την κατανομή πιθανοτήτων. Ο αριθμός των πιθανοτήτων που είναι διάφορες του 0 (ή πολύ κοντά στο 0) δίνει το r Τα σημεία της κατανομής πιθανότητας στα οποία παρατηρούνται κορυφές (υψηλές τιμές πιθανότητας) δίνουν την περίοδο r. Για παράδειγμα έστω ότι θέλουμε να παραγοντοποιήσουμε τον αριθμό N=55 με δύο διαφορετικούς a, τους 13 και 43. Το 55 αναπαρίσταται από 6 bits, πράγμα που σημαίνει η μέτρηση θα γίνει σε 2x6=12 qubits και η κατανομή πιθανότητας θα είναι σε 2^{12}=4096 σημεία.

Για α= 43:

$$43^0 \bmod 55 = 1$$

$$43^1 \bmod 55 = 43$$

$$43^2 \bmod 55 = 34$$

$$43^3 \bmod 55 = 32$$

$$43^4 \bmod 55 = 1$$

Επομένως r = 4. Τότε στην κατανομή πιθανότητας οι κορυφές θα έχουν απόσταση 4096/4 = 1024.

$$P_1 = \text{ΜΚΔ}\{a^{r/2}-1,N\} = \text{ΜΚΔ}\{43^2-1,55\} = \text{ΜΚΔ}\{1848,55\} = 11$$

$$P_2 = \text{ΜΚΔ}\{a^{r/2}-1,N\} = \text{ΜΚΔ}\{43^2+1,55\} = \text{ΜΚΔ}\{1850,55\} = 5$$

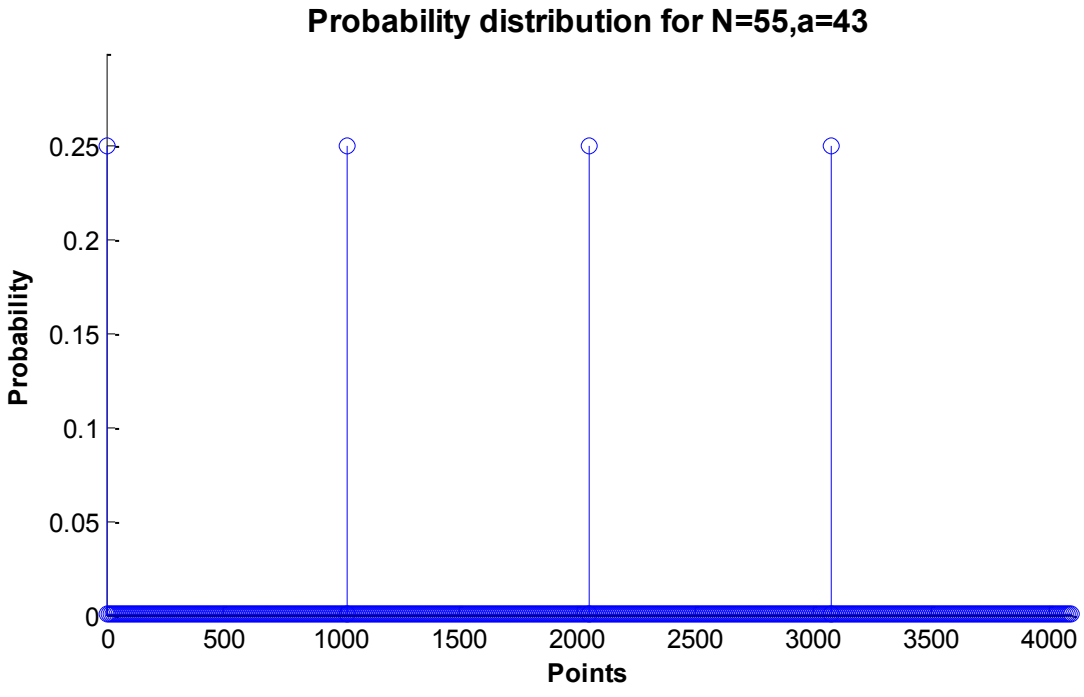
Όπως φαίνεται στο Σχήμα 4.6 το αποτέλεσμα της προσομοίωσης μας δίνει το επιθυμητό αποτέλεσμα.

Για α= 13, αν γίνουν αναλυτικά οι πράξεις προκύπτει ότι r = 20. Επομένως 4096/20 = 204,8 το οποίο δεν είναι ακέραιος.

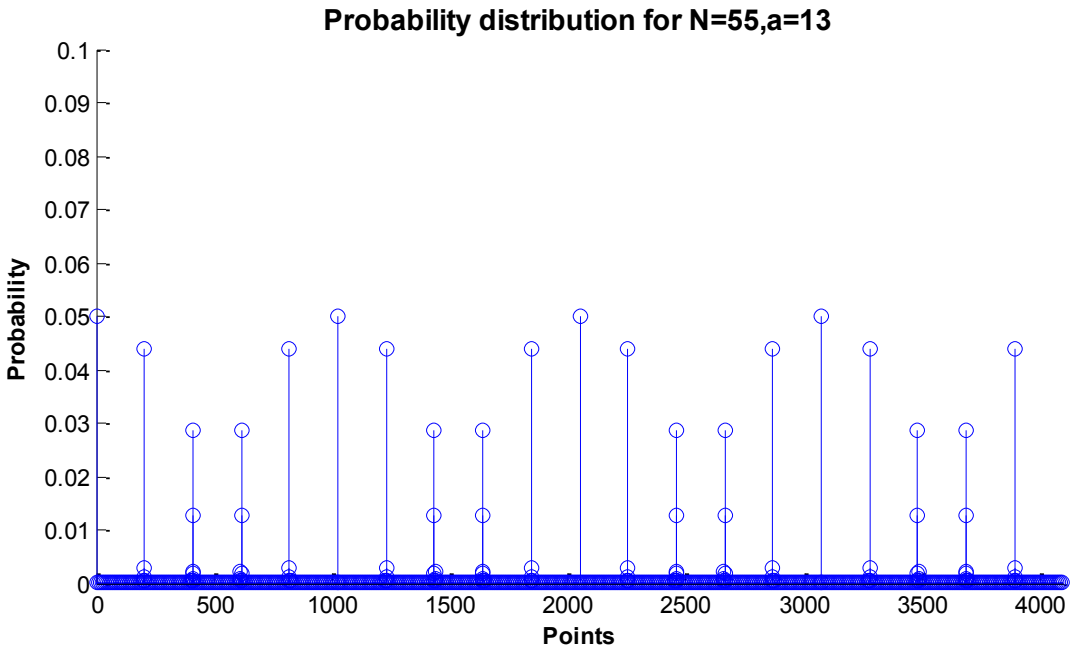
$$P_1 = \text{ΜΚΔ}\{a^{r/2}-1,N\} = \text{ΜΚΔ}\{13^{10}-1,55\} = \text{ΜΚΔ}\{\ 137858491848,55\} = 11$$

$$P_2 = \text{ΜΚΔ}\{a^{r/2}-1,N\} = \text{ΜΚΔ}\{13^{10}+1,55\} = \text{ΜΚΔ}\{\ 137858491850,55\} = 5$$

Στο Σχήμα 4.7 φαίνονται σωστά 20 κορυφές, αλλά λόγω του ότι το r δεν είναι δύναμη του 2 και προκύπτει μη ακέραια απόσταση ανάμεσα στις κορυφές, υπάρχει μια 'διαρροή' τιμών γύρω απ' αυτές.



Σχήμα 4.6 – Κατανομή πιθανοτήτων για N=55, a=43



Σχήμα 4.7 – Κατανομή πιθανοτήτων για N=55, a=13

Ακολουθούν στην συνέχεια πίνακες με μετρικές από τις εκτελέσεις κυκλωμάτων του αλγόριθμου Shor.

Πίνακας 4.8 - Μετρικές για Shor, N=17

Όνομα Μετρικής	Μέση τιμή
achieved_occupancy	0.810865
ipc	1.925457
sm_efficiency	84.83%
flop_count_sp	5767168
warp_execution_efficiency	86.65%
stall_exec_dependency	27.82%
stall_memory_dependency	47.57%
gld_throughput	78.448GB/s
gld_transactions	32768

gst_throughput	10.985GB/s
gst_transactions	24576
l2_read_throughput	79.376GB/s
l2_read_transactions	131388
l2_write_throughput	65.445GB/s
l2_write_transactions	100459

Πίνακας 4.9 - Μετρικές για Shor, N=55

Όνομα Μετρικής	**Μέση τιμή**
achieved_occupancy	0.785096
ipc	1.807391
sm_efficiency	99.92%
flop_count_sp	1476395008
warp_execution_efficiency	98.21%
stall_exec_dependency	27.12%
stall_memory_dependency	49.32%
gld_throughput	90.889GB/s
gld_transactions	8388608
gst_throughput	14.474GB/s
gst_transactions	6291456
l2_read_throughput	90.895GB/s
l2_read_transactions	33556624
l2_write_throughput	77.535GB/s
l2_write_transactions	25349237

4.4 Προσομοίωση αθροιστών βασισμένων σε QFT

Εδώ θα προσομοιωθούν μερικοί αθροιστές που σχεδιάστηκαν από τον T. G. Draper και βασίζονται σαν κυκλώματα σε αυτά του QFT που έχει ήδη περιγραφθεί. Αυτοί οι αθροιστές μπορούν να προσθέσουν δύο κβαντικούς ακεραίους ή έναν κβαντικό με μία σταθερά. Στην πρώτη περίπτωση οι αθροιστές έχουν 8 qubits, από τα οποία στα 4 λιγότερα σημαντικά εισάγεται ο ένας αριθμός, στα 4 περισσότερα σημαντικά ο άλλος και το αποτέλεσμα βγαίνει στα 4 λιγότερα σημαντικά χωρίς να γίνει αλλαγή στα υπόλοιπα. Για παράδειγμα για να γίνει η πρόσθεση $|6\rangle + |1\rangle$ το διάνυσμα κατάστασης εισόδου έχει την μορφή 0110 0001 και το αποτέλεσμα είναι 0110 0111. Για πρόσθεση υπέρθεσης, όπως για παράδειγμα την υπέρθεση $(|4\rangle + |6\rangle)$ (για λόγους απλότητας αγνοούμε τους συντελεστές $1/\sqrt{2}$ της υπέρθεσης) με το $|1\rangle$, στα πιο σημαντικά qubits δίνεται η τιμή $|4\rangle$, και με μια πύλη Hadamard στο δεύτερο πιο σημαντικό qubit δημιουργείται η υπέρθεση $|4\rangle + |6\rangle$. Αντίστοιχα προετοιμάζεται το διάνυσμα εισόδου για την πρόσθεση δύο υπερθέσεων. Αυτοί οι αθροιστές με 8 qubits μπορούν να δώσουν μέγιστο άθροισμα 15, αλλιώς γίνεται υπερχείλιση και το αποτέλεσμα είναι άθροισμα mod 16. Ένας αθροιστής με σταθερά δεν χρειάζεται την εισαγωγή δύο αριθμών και επομένως με 5 qubits μπορεί να έχει όριο το 31.

Σε αυτές τις προσομοιώσεις επιλεχθήκαν αθροιστές μικρού εύρους για να δειχθεί ξανά η ορθή λειτουργία του προσομοιωτή και επομένως δεν έχει ιδιαίτερη αξία η ανάλυση των χρόνων εκτέλεσης και οι μετρικές, εξ' άλλου συνήθως χρησιμοποιούνται ως υποκυκλώματα σε άλλα μεγαλύτερα. Η επίδειξη της ορθής λειτουργίας θα γίνει με την ανάγνωση τον αρχείων που αποθηκεύουν το διάνυσμα κατάστασης.

- Για πρόσθεση $|6\rangle + |1\rangle$ το διάνυσμα που επιστρέφεται είναι το:

```
 1.32094e-08 + -7.48613e-09i   |01100001>   2.30531e-14%
 7.13508e-08 + -2.30098e-08i   |01100011>   5.62039e-13%
 2.89263e-08 + -1.23535e-08i   |01100101>   9.8934e-14%
           1 +   1.0344e-07i   |0110 0111>   100%
-3.42828e-08 + -1.43696e-08i   |01101001>   1.3818e-13%
 1.29429e-08 + -4.72031e-08i   |01101011>   2.39565e-13%
 -7.8529e-09 + -9.50224e-09i   |01101101>   1.51961e-14%
           0 +  1.04843e-08i   |01101111>   1.0992e-14%
```

Παραπάνω φαίνονται τα στοιχεία του διανύσματος που δεν είναι μηδενικά. Στην πρώτη στήλη είναι η μιγαδική αναπαράσταση των πλατών πιθανότητας, στη δεύτερη η κατάσταση του διανύσματος και στην τρίτη οι πιθανότητες εμφάνισης αυτής της κατάστασης.

Μας ενδιαφέρουν τα 4 λιγότερα σημαντικά bits που είναι έντονα. Η κατάσταση 0111 έχει πιθανότητα εμφάνισης 100%, το όποιο είναι το σωστό αποτέλεσμα αφού 6+1=7. Οι υπόλοιπες καταστάσεις που εμφανίζονται έχουν πολύ μικρές τιμές κοντά στο 0 και οφείλονται σε λάθη κατά την διάρκεια των αριθμητικών πράξεων και μπορούν να αγνοηθούν.

- Για πρόσθεση $(|4\rangle + |6\rangle) + |1\rangle$:

```
 3.72354e-08 +  1.48011e-08i   |01000001>   1.60555e-13%
 3.19127e-16 +  6.88344e-09i   |01000011>   4.73817e-15%
    0.707107 +  2.96835e-08i   |0100 0101>   50%
 4.35771e-16 +  2.85121e-09i   |01000111>   8.12942e-16%
-1.61619e-08 + -3.15101e-08i   |01001001>   1.2541e-13%
-3.19127e-16 + -6.88344e-09i   |01001011>   4.73817e-15%
 5.96046e-08 + -1.29746e-08i   |01001101>   3.72105e-13%
-4.35771e-16 + -2.85121e-09i   |01001111>   8.12942e-16%
 5.70243e-09 + -7.72715e-09i   |01100001>   9.22266e-15%
 3.57084e-08 +  1.83944e-08i   |01100011>   1.61345e-13%
 1.37669e-08 + -7.72715e-09i   |01100101>   2.49236e-14%
    0.707107 +  4.80189e-08i   |0110 0111>   50%
-5.70243e-09 + -7.72715e-09i   |01101001>   9.22266e-15%
 -1.4635e-08 + -2.52682e-08i   |01101011>   8.52667e-14%
-1.37669e-08 + -7.72715e-09i   |01101101>   2.49236e-14%
 5.96046e-08 + -1.02365e-08i   |01101111>   3.6575e-13%
```

Σε αυτήν την περίπτωση που υπάρχει υπέρθεση δύο τιμών και το αποτέλεσμα θα είναι υπέρθεση δύο τιμών. Συγκεκριμένα $(|4\rangle + |6\rangle) + |1\rangle$ = $(|5\rangle + |7\rangle)$. Στο διάνυσμα φαίνεται ότι οι καταστάσεις 0101 (5) και 0111(7) μοιράζονται τις πιθανότητες 50%-50%, επομένως το αποτέλεσμα είναι σωστό.

- Για πρόσθεση $(|4\rangle + |6\rangle) + (|1\rangle + |5\rangle)$:

```
 1.49012e-08 + -2.55717e-08i   |01000001>   8.75956e-14%
-3.44172e-09 + -1.42561e-09i   |01000011>   1.38778e-15%
         0.5 +  4.10713e-09i   |0100 0101>   25%
-1.42561e-09 +  3.44172e-09i   |01000111>   1.38778e-15%
         0.5 +  2.55717e-08i   |0100 1001>   25%
 3.44172e-09 +  1.42561e-09i   |01001011>   1.38778e-15%
 1.49012e-08 + -4.10713e-09i   |01001101>   2.38913e-14%
 1.42561e-09 + -3.44172e-09i   |01001111>   1.38778e-15%
-2.32569e-08 + -1.43696e-08i   |01100001>   7.47369e-14%
 4.47035e-08 + -3.06895e-08i   |01100011>   2.94025e-13%
   2.173e-08 + -9.50224e-09i   |01100101>   5.62486e-14%
         0.5 +  5.64277e-09i   |0110 0111>   25%
 2.32569e-08 + -7.48613e-09i   |01101001>   5.96927e-14%
         0.5 +  5.25452e-08i   |0110 1011>   25%
  -2.173e-08 + -1.23535e-08i   |01101101>   6.24801e-14%
 4.47035e-08 +  1.62129e-08i   |01101111>   2.26126e-13%
```

Η πρόσθεση δύο υπερθέσεων θα έχει ως αποτέλεσμα 4 πιθανές τιμές. $(|4\rangle + |6\rangle) + (|1\rangle + |5\rangle)$ =$|5\rangle + |9\rangle + |7\rangle + |11\rangle$. Σωστά στο διάνυσμα οι τιμές 0101(5), 1001(9), 0111(7) και 1011(11) μοιράζονται τις πιθανότητες με 25% η καθεμία.

- Για πρόσθεση $|12\rangle + |14\rangle$ με σταθερά 13 :

```
-9.70585e-09 +  -4.7949e-09i   |00000>   1.17194e-14%
 -3.8023e-08 + -2.54175e-08i   |00001>   2.0918e-13%
-2.36481e-09 + -9.89758e-09i   |00010>   1.03554e-14%
 8.23434e-09 + -3.55051e-08i   |00011>   1.32842e-13%
-3.90613e-10 + -5.01603e-09i   |00100>   2.53131e-15%
 9.79284e-09 + -1.14673e-08i   |00101>   2.27398e-14%
 2.76102e-09 + -9.68472e-10i   |00110>   8.56115e-16%
 3.60467e-09 + -1.82031e-08i   |00111>   3.44348e-14%
  3.5655e-09 +  2.88851e-09i   |01000>   2.10563e-15%
 1.19209e-07 + -4.60736e-08i   |01001>   1.63336e-12%
 4.92448e-09 + -1.00448e-09i   |01010>   2.52595e-15%
 2.98023e-08 + -4.62515e-09i   |01011>   9.0957e-14%
 9.06468e-09 + -7.65021e-09i   |01100>   1.40694e-14%
-1.42998e-08 +  1.01912e-08i   |01101>   3.08346e-14%
-1.28846e-09 + -1.34097e-09i   |01110>   3.45834e-16%
-4.97819e-09 +  6.69008e-09i   |01111>   6.95395e-15%
-5.51489e-09 + -1.06961e-09i   |10000>   3.15581e-15%
 5.90965e-08 + -2.32495e-08i   |10001>   4.03293e-13%
-4.22063e-09 + -6.27766e-09i   |10010>   5.72228e-15%
 5.49859e-08 + -4.13987e-08i   |10011>   4.73731e-13%
 2.57284e-09 + -2.71113e-09i   |10100>   1.39697e-15%
 -1.4361e-08 +  1.64591e-08i   |10101>   4.77138e-14%
  1.1727e-08 +  -8.8473e-09i   |10110>   2.15796e-14%
-1.04881e-08 +  2.34164e-08i   |10111>   6.58328e-14%
 2.59172e-08 + -1.24783e-08i   |11000>   8.27412e-14%
    0.707107 +  9.95367e-08i   |11001>   50%
 1.66096e-09 + -1.99987e-09i   |11010>   6.75829e-16%
    0.707107 +  9.63087e-08i   |11011>   50%
-2.55089e-08 + -7.69456e-11i   |11100>   6.5071e-14%
```

Αυτή είναι η περίπτωση του αθροιστή 5 qubits με σταθερά, στην συγκεκριμένη περίπτωση την 13. Μας ενδιαφέρει η τιμή και των 5 qubits.$(|12\rangle + |14\rangle)$+ 13 = $|25\rangle + |27\rangle$. Όπως φαίνεται στο διάνυσμα το αποτέλεσμα δίνει σωστά ισοπίθανες τις καταστάσεις 11001(25) και 11011(27).

4.5 Προσομοίωση VBE αθροιστών

Τέλος θα παρουσιαστούν τα αποτελέσματα της προσομοίωσης ενός άλλου είδους αθροιστών, των VBE (το όνομα προέρχεται από τα αρχικά των σχεδιαστών του Vedral, Barenco και Ekert), οι οποίοι αντιστοιχούν σε ripple carry αθροιστές σε κβαντικό κύκλωμα. Το ενδιαφέρον και ο λόγος παρουσιάζονται είναι ότι υλοποιούνται διαφορετικά και έτσι επαληθεύεται η σωστή λειτουργία περισσότερων κβαντικών πυλών, συγκεκριμένα των CNOT και Toffoli.

Αυτά τα κυκλώματα προσθέτουν δύο αριθμούς των n qubits, a και b. Το αποτέλεσμα εξάγεται στα qubits του b, ενώ το a παραμένει αναλλοίωτο. Ο αριθμός b έχει ένα qubit περισσότερο το οποίο χρησιμοποιείται σαν carry για την αποφυγή των υπερχειλίσεων. Επιπλέον των προηγούμενων χρησιμοποιούνται n βοηθητικά (ancilla) qubits για τον υπολογισμό ενδιάμεσων κρατούμενων. Αυτά τα qubits ξεκινάν πάντα ως 0 και καταλήγουν 0, δεν παράγουν επομένως ‘σκουπίδια’ στην έξοδο. Συνολικά για την πρόσθεση δύο n qubits αριθμών χρειάζονται 3n+1 qubits.

Για τα παρακάτω αποτελέσματα δημιουργήθηκαν κυκλώματα για πρόσθεση αριθμών των 4 qubits, επομένως έχουν εύρος 3*4+1 = 13 qubits. Για την δημιουργία υπερθέσεων χρησιμοποιήθηκαν επιπλέον πύλες Hadamard στις κατάλληλες θέσεις.

- Για πρόσθεση $|3\rangle + |7\rangle$:

Η σειρά των qubits είναι **anc3 anc2 anc1 anc0 a3 a2 a1 a0 b4 b3 b2 b1 b0,** επομένως για την πράξη αυτή το διάνυσμα κατάστασης εισόδου είναι 0000 0011 00111. Το αποτέλεσμα που προκύπτει διαβάζοντας το αρχείο του διανύσματος κατάστασης εξόδου είναι:

```
1 + 0i   |000000110101010>   100%
```

Το αποτέλεσμα 10(1010) βρίσκονται σωστά στα b, το a δεν έχει αλλάξει και τα ancilla παραμένουν 0.

- Για πρόσθεση $|7\rangle + |10\rangle$:

Το διάνυσμα κατάστασης εισόδου είναι 0000 0111 01010. Το αποτέλεσμα που προκύπτει διαβάζοντας το αρχείο του διανύσματος κατάστασης εξόδου είναι:

```
1 + 0i   |000001111010001>   100%
```

Το αποτέλεσμα 17(10001) βρίσκονται σωστά στα b και χρησιμοποιείται το επιπλέον qubit λόγω της αδυναμίας αναπαράστασης του σε 4 bits.

- Για πρόσθεση $(|3\rangle + |11\rangle) + (|7\rangle + |15\rangle)$:

Το διάνυσμα κατάστασης εισόδου είναι 0000 0011 00111 και για την δημιουργία των υπερθέσεων χρησιμοποιούνται πύλες Hadamard στα qubits 3 και 8.Το αποτέλεσμα που προκύπτει διαβάζοντας το αρχείο του διανύσματος κατάστασης εξόδου είναι:

```
0.5 + 0i   |000000110101010>   25%
0.5 + 0i   |000000111010010>   25%
0.5 + 0i   |000010111010010>   25%
0.5 + 0i   |000010111011010>   25%
```

Το αποτέλεσμα 10(10001), 18(10010), 18(10010), 26(11010) βρίσκονται σωστά στα b και χρησιμοποιείται το επιπλέον qubit στις 3 περιπτώσεις λόγω της αδυναμίας αναπαράστασης σε 4 bits.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Μετά την θεωρία, την σχεδίαση, την υλοποίηση, τις μετρήσεις και τα αποτελέσματα, έρχεται η ώρα της εξαγωγής των συμπερασμάτων για την όλη διαδικασία της εργασίας αυτής.

Αρχικά γίνεται αναφορά στις απαιτήσεις για μνήμη. Όπως φάνηκε στο τρίτο κεφάλαιο οι τεράστιες θεωρητικές απαιτήσεις μειώνονται σε μεγάλο βαθμό με αυτήν την υλοποίηση, αν και παραμένουν ακόμα υψηλές. Το αποτέλεσμα είναι να μπορεί να προσομοιωθεί ένας ικανοποιητικός αριθμός από qubits. Στον Πίνακα 5.1 παρουσιάζεται η αντιστοιχία του μεγέθους της μνήμης με τα qubits που μπορεί να προσομοιώσει.

Πίνακας 0.1 – Αντιστοιχία μνήμης / qubits

Μέγεθος Μνήμης	Qubits
512 MB	24
1 GB	25
2 GB	26
4 GB	27
8 GB	28

Όπως παρατηρείται μπορεί να μην υπάρχει μεγάλη αύξηση στον αριθμό των qubits αλλά είναι σημαντικό ότι και σε μικρές κάρτες προσομοιώνεται ένας ικανοποιητικός αριθμός. Επίσης σαν θετικό μπορεί να υπολογιστεί το γεγονός ότι με αυτόν τον τρόπο δεν γεμίζει η μνήμη του συστήματος στο οποίο βρίσκεται η κάρτα και μπορεί να χρησιμοποιείται από άλλες διεργασίες.

Όσων αφορά την απόδοση και τους χρόνους εκτέλεσης, από το τέταρτο κεφάλαιο και την σύγκριση παράλληλης και σειριακής εκτέλεσης είναι φανερό ότι η χρήση κάρτας γραφικών μπορεί να έχει πολύ καλύτερα αποτελέσματα. Για πολύ μικρά κυκλώματα μπορεί η CPU να ήταν ανώτερη, αλλά οι χρόνοι είναι αμελητέοι σε τέτοια επίπεδα. Όσο μεγαλύτερο το κβαντικό κύκλωμα τόσο περισσότερο κερδίζει η GPU. Οι δοκιμές στην CPU έγιναν με χρήση ενός μόνο thread, αλλά η διαφορά είναι τόσο μεγάλη που λογικά και με καλύτερη υλοποίηση που να εκμεταλλεύεται όλους τους πυρήνες και τα threads μιας σύγχρονης CPU, η διαφορά θα παραμείνει υπέρ της κάρτας. Για παράδειγμα στο μεγαλύτερο κύκλωμα που δοκιμάστηκε με 26 qubits και πάνω από 25000 πύλες η εκτέλεση έκανε 3 λεπτά έναντι >15 ώρες. Βέβαια όλα αυτά είναι σχετικά με το τι GPU και CPU θα χρησιμοποιηθούν.

Οι βελτιώσεις που μπορούν να υπάρξουν πιθανόν να μην είναι πολλές. Από τις μετρικές φαίνεται να γίνεται αρκετά καλή χρήση της κάρτας γραφικών. Το είδος του προβλήματος είναι καλό για παραλληλοποίηση αλλά έχει το πρόβλημα της μνήμης. Στην υλοποίηση χρησιμοποιείται η Global memory η οποία είναι η πιο αργή στην ιεραρχία αλλά δεν μπορούν να χρησιμοποιηθούν πιο γρήγορες γιατί είναι αρκετά πιο μικρές. Ακόμα η διάσπαση σε κομμάτια και τμηματική επεξεργασία δεν είναι εύκολη υπόθεση γιατί σε

πολλές περιπτώσεις απαιτούνται τμήματα της μνήμης που βρίσκονται σε απόσταση. Αν υπάρχει περίπτωση να υπάρξει βελτίωση θα είναι επανασχεδιασμό από την αρχή του αλγόριθμου.

Ένα σημείο που μπορεί να υπάρξει εύκολα βελτίωση είναι να γίνεται χρήση πολλών καρτών που λειτουργούν παράλληλα. Αυτό θα επιφέρει αύξηση στον αριθμό των qubits αλλά πιθανή μείωση στην απόδοση λόγω μετακινήσεων μεταξύ των καρτών.

Συνολικά οι αρχικοί στόχοι έχουν επιτευχθεί. Το πρόγραμμα που δημιουργήθηκε είναι απλό στην χρήση, είναι γενικής χρήσης σε θέματα κβαντικών κυκλωμάτων, έχει την ικανότητα να προσομοιώνει μεγάλο αριθμό qubits και οι χρόνοι εκτέλεσης είναι εμφανώς πολύ ικανοποιητικοί.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Host	Δέκτης, εξυπηρετητής (αναφέρεται στο σύστημα στο οποίο 'φιλοξενείται' η κάρτα
Device	Συσκευή (αναφέρεται στην κάρτα)
Multiprocessor	Πολυεπεξεργαστής
Kernel	Πυρήνας
Thread	Νήμα
Register	Καταχωρητής
Cache memory	Κρυφή μνήμη
Shared memory	Διαμοιραζόμενη μνήμη
Constant memory	Σταθερή μνήμη
Local memory	Τοπική μνήμη
Architecture	Αρχιτεκτονική
Clock rate	Ρυθμός ρολογιού
Bus Width	Πλάτος διαύλου
Dimension	Διάσταση
Size	Μέγεθος
Bandwidth	Εύρος ζώνης
Transfer	Μεταφορά
Execution	Εκτέλεση
Efficiency	Αποδοτικότητα
Throughput	Διαμεταγωγή, παροχέτευση
Transactions	Συναλλαγές
Dependency	Εξάρτηση

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
CPU	Central Processing Unit
SM	Streaming Multiprocessor
SIMD	Single Instruction Multiple Data
FLOP	Floating point
QFT	Quantum Fourier Transform
RISC	Reduced Instruction Set Computer

ΠΑΡΑΡΤΗΜΑ Ι

Σε αυτό το παράρτημα παρουσιάζεται ολόκληρος ο κώδικας του εξομοιωτή.

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cuda.h>
#include <cuda runtime.h>
#include <device launch parameters.h>
#include <cuComplex.h>
#include <math.h>
#include <vector>
#include <cuda profiler api.h>
#include <bitset>
#include <ctime>

using namespace std;

#define PI 3.14159265358979323846264338327795

//global variables
string init_vector;
unsigned int qubits;
unsigned int size;
int gate_pos, gate_pos_2, gate_pos_3 = 0;
unsigned int input_value;
char gate;
string temp_gate;
unsigned int step = 0;
float angle = 0.0;
int int angle = 0;
cudaEvent_t start, stop, start_m, stop_m;
int max_qubits;
int maxThreadsPerBlock;
int multiProcessorCount;
float timing, measure_time;
float computation_time = 0.0;
int threads;
int blocks;
int major, minor;
bool measure = false;
int measure_gates;
int measure_size = 0;
cudaError_t err;
bool m_argument = false;
bool v_argument = false;
string v_results;
string m_results;

//create index's to vectors
cuFloatComplex *host_vector;
cuFloatComplex *device_vector_1, *device_vector_2;

//create index's for measurement vectors
int *measure_positions;
int *device_measure_positions;
float *host_measure_vector;
float *device_measure_vector;

//function: find if the bit in position is 1 or 0
__device__ int convert(int dec, int gate)
{

    if (dec & (1 << gate))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
//kernel function
__global__ void computation(cuFloatComplex *device_vector_2, cuFloatComplex *device_vector_1,
int size, int gate, int gate2, int gate3, char g, float angle)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;

    if (i < size)
    {
        if (convert(i, gate) == 0)
        {
            if (g == 'H')
            {
                device_vector_2[i] = cuCaddf(cuCmulf(make_cuFloatComplex(1 / sqrtf(2), 0.0),
device_vector_1[i]), cuCmulf(make_cuFloatComplex(1 / sqrtf(2), 0.0), device_vector_1[i + (1
<< gate)]));
            }
            else if (g == 'X')
            {
                device_vector_2[i] = device_vector_1[i + (1 << gate)];
            }
            else if (g == 'Y')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i + (1 << gate)],
make_cuFloatComplex(0.0, -1.0));
            }
            else if (g == 'Z')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'S')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'T')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'r')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'n')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'w')
            {
                if (convert(i, gate2) == 1)
                {
                    device_vector_2[i] = device_vector_1[i + (1 << gate) - (1 << gate2)];
                }
                else if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
            }
            else if (g == 'v')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'u')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'o')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'f')
            {
                device_vector_2[i] = device_vector_1[i];
            }
            else if (g == 'k')
            {
                device_vector_2[i] = device_vector_1[i];
            }
        }
```

```
        else if (convert(i, gate) == 1)
        {
            if (g == 'H')
            {
                device_vector_2[i] = cuCaddf(cuCmulf(make_cuFloatComplex(1 / sqrtf(2), 0.0),
device_vector_1[i - (1 << gate)]), cuCmulf(make_cuFloatComplex(-(1 / sqrtf(2)), 0.0),
device_vector_1[i]));
            }
            else if (g == 'X')
            {
                device_vector_2[i] = device_vector_1[i - (1 << gate)];
            }
            else if (g == 'Y')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i - (1 << gate)],
make_cuFloatComplex(0.0, 1.0));
            }
            else if (g == 'Z')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i], make_cuFloatComplex(-1.0,
0.0));
            }
            else if (g == 'S')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i], make_cuFloatComplex(0.0,
1.0));
            }
            else if (g == 'T')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i], make_cuFloatComplex(1 /
sqrtf(2), 1 / sqrtf(2)));
            }
            else if (g == 'r')
            {
                device_vector_2[i] = cuCmulf(device_vector_1[i],
make_cuFloatComplex(cosf(angle), sinf(angle)));
            }
            else if (g == 'n')
            {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i + (1 << gate2)];
                }
                else if (convert(i, gate2) == 1)
                {
                    device vector 2[i] = device vector 1[i - (1 << gate2)];
                }
            }
            else if (g == 'w')
            {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i - ((1 << gate) - (1 << gate2))];
                }
                else if (convert(i, gate2) == 1)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
            }
            else if (g == 'v')
            {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
                else if (convert(i, gate2) == 1)
                {
                    device_vector_2[i] = cuCmulf(device_vector_1[i], make_cuFloatComplex(-
1.0, 0.0));
                }
            }
```

```
            else if (g == 'u')

            {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
                else if (convert(i, gate2) == 1)
                {
                    device_vector_2[i] = cuCmulf(device_vector_1[i],
make_cuFloatComplex(cosf(angle), sinf(angle)));
                }
            }
            else if (g == 'o')
            {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
                else if (convert(i, gate2) == 1)
                {
                    if (convert(i, gate3) == 0)
                    {
                        device vector 2[i] = device vector 1[i + (1 << gate3)];
                    }
                    else if (convert(i, gate3) == 1)
                    {
                        device_vector_2[i] = device_vector_1[i - (1 << gate3)];
                    }
                }
            }
            else if (g == 'f')
            {
                if (convert(i, gate2) == 0)
                {
                    if (convert(i, gate3) == 0)
                    {
                        device_vector_2[i] = device_vector_1[i];
                    }
                    else if (convert(i, gate3) == 1)
                    {
                        device_vector_2[i] = device_vector_1[i + ((1 << gate2) - (1 <<
gate3))];
                    }
                }
                else if (convert(i, gate2) == 1)
                {
                    if (convert(i, gate3) == 0)
                    {
                        device_vector_2[i] = device_vector_1[i - ((1 << gate2) - (1 <<
gate3))];
                    }
                    else if (convert(i, gate3) == 1)
                    {
                        device_vector_2[i] = device_vector_1[i];
                    }
                }
            }
            else if (g == 'k')
               {
                if (convert(i, gate2) == 0)
                {
                    device_vector_2[i] = device_vector_1[i];
                }
                else if (convert(i, gate2) == 1)
                {
                    if (convert(i, gate3) == 0)
                    {
                        device_vector_2[i] = device_vector_1[i];
                    }
                    else if (convert(i, gate3) == 1)
                    {
                        device_vector_2[i] = cuCmulf(device_vector_1[i],
make_cuFloatComplex(cosf(angle), sinf(angle)));
                    }
                }
            }
```

```
        }
    }
}

//function: find measure gate position
__device__ int measure_position(int measure_gates, int dec, int *device_measure_positions)
{
    int position = 0;

    for (int i = 0; i<measure_gates; i++)
    {
        if (dec & (1 << device_measure_positions[i]))
        {
            position = position + (1 << i);
        }
    }

    return position;
}

//measurement kernel function
__global__ void measurement(float *device_measure_vector, int *device_measure_positions,
cuFloatComplex *device_vector, int measure_gates, int size)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    int position;

    if (i < size)
    {
        position = measure_position(measure_gates, i, device_measure_positions);
        atomicAdd(&device_measure_vector[position], powf(cuCrealf(device_vector[i]), 2) +
powf(cuCimagf(device_vector[i]), 2));
    }
}
//function: checks if a gate's qubit exceeds the declared value
void chech_number_of_qubits(int qubits, int g, int line_cnt)
{
    if (g >= qubits)
    {
        cout << "Error in line: " << line_cnt << endl;
        cout << "You can use qubits from 0 to " << qubits - 1 << endl;
        exit(EXIT_FAILURE);
    }

}

//function: free allocated memory both on host and device
void free_mem()
{
    //free host memory
    free(host_vector);

    //free device memory
    cudaFree(device_vector_1);
    cudaFree(device_vector_2);
}

//function: free allocated memory with measurement
void free_mem_all()
{
    //free host memory
    free(host_vector);
    free(measure_positions);
    free(host_measure_vector);

    //free device memory
    cudaFree(device_vector_1);
    cudaFree(device_vector_2);
    cudaFree(device_measure_positions);
    cudaFree(device_measure_vector);
}
```

```
//function: prints message and exits when a syntax error is found
void syntax_error(int line_cnt)
{
    cout << "Syntax error in line: " << line_cnt << endl;
    if(measure)
    {
        free_mem_all();
    }
    else
    {
        free_mem();
    }
    exit(EXIT_FAILURE);
}

//function: read gate position of one qubit gate
void one_qubit_gate(string line, int line_cnt, int *gate_pos, int qubits)
{
    int g;
    if (line[2] == 'q')
    {
        line.erase(0, 3);
        if (!(istringstream(line) >> g)) g = 0;
        chech_number_of_qubits(qubits, g, line_cnt);
        *gate pos = g;
    }
    else
    {
        syntax_error(line_cnt);
    }
}

//function: read gate positions of two qubit gate
void two_qubit_gate(string line, int line_cnt, int *gate_pos, int *gate_pos_2, int qubits)
{
    int g;
    int gate_ptr = 0;
    string gate;

    gate_ptr = line.find_first_of('q');
    if (gate_ptr <= line.length())
    {
        line.erase(0, gate_ptr + 1);
        gate_ptr = line.find_first_of('q');
        if (gate_ptr <= line.length())
        {
            gate = line.substr(0, gate ptr);
            if (!(istringstream(gate) >> g)) g = 0;
            chech_number_of_qubits(qubits, g, line_cnt);
            *gate_pos = g;
            line.erase(0, gate_ptr + 1);
            if (!(istringstream(line) >> g)) g = 0;
            chech_number_of_qubits(qubits, g, line_cnt);
            *gate_pos_2 = g;
            gate_ptr = 0;
            gate.clear();
        }
        else
        {
            syntax_error(line_cnt);
        }
    }
    else
    {
        syntax_error(line_cnt);
    }

}
```

```
//function: read gate positions of three qubit gate
void three_qubit_gate(string line, int line_cnt, int *gate_pos, int *gate_pos_2, int
*gate_pos_3, int qubits)
{
    int g;
    int gate_ptr = 0;
    string gate;

    gate_ptr = line.find_first_of('q');
    if (gate_ptr <= line.length())
    {
        line.erase(0, gate_ptr + 1);
        gate_ptr = line.find_first_of('q');
        if (gate_ptr <= line.length())
        {
            gate = line.substr(0, gate_ptr);
            if (!(istringstream(gate) >> g)) g = 0;
            chech_number_of_qubits(qubits, g, line_cnt);
            *gate_pos = g;
            line.erase(0, gate_ptr + 1);
            gate_ptr = line.find_first_of('q');
            if (gate_ptr <= line.length())
            {
                gate = line.substr(0, gate_ptr);
                if (!(istringstream(gate) >> g)) g = 0;
                chech_number_of_qubits(qubits, g, line_cnt);
                *gate pos 2 = g;
                line.erase(0, gate_ptr + 1);
                if (!(istringstream(line) >> g)) g = 0;
                chech_number_of_qubits(qubits, g, line_cnt);
                *gate_pos_3 = g;
            }
            else
            {
                syntax_error(line_cnt);
            }
            gate_ptr = 0;
            gate.clear();
        }
        else
        {
            syntax_error(line_cnt);
        }
    }
    else
    {
        syntax_error(line_cnt);
    }
}

//function read input file
void parser(string path)
{
    //local variables
    string line;
    ifstream myfile(path);
    int gate_ptr = 0;
    int g;
    int line_cnt = 0;
    bool init = false;
    bool neg = false;
    string s;
    qubits = 0;
    measure_gates = 0;
    int measure_index = 0;

    //open file
    if (myfile.is_open())
    {
        //read line
        while (getline(myfile, line))
        {
            line_cnt++;
            //if line isn't empty and not a comment
            if (!(line.empty()) && (line[0] != '#') && (line[0] != ' ') && (line[0] != '\t')
&& (line[0] != '\n') && (line[0] != '\r'))
            {
```

```
                for (size_t i = 0; i < line.length(); i++)
                {
                    //delete white space and commas
                    if (line[i] == ' ' || line[i] == '\n' || line[i] == '\t' || line[i] ==
','')
                    {
                        line.erase(i, 1);
                        i--;
                    }
                }
                //get qubits number and init vector with 0
                if (qubits == 0)
                {
                    if (line.substr(0, 6) == "qubits")
                    {
                        line.erase(0, 7);
                        if (!(istringstream(line) >> qubits)) qubits = 0;
                        if (qubits > max_qubits)
                        {
                            cout << "Too many qubits. You can use up to " << max_qubits << "
on this GPU" << endl;
                            exit(EXIT_FAILURE);
                        }
//compute block size according computation capability
                        size = 1 << qubits;

                        if ((major == 1) || (major == 2))
                        {
                            if (size < 32)
                            {
                                threads = size;
                            }
                            else
                            {
                                threads = 32;
                            }
                        }
                        else if ((3) || (5))
                        {
                            if (size < 128)
                            {
                                threads = size;
                            }
                            else
                            {
                                threads = 128;
                            }
                        }
                        blocks = size / threads;
                    }
                    else
                    {
                        cout << "Syntax error: In the first line you must define the number
of qubits." << endl;
                        exit(EXIT_FAILURE);
                    }
                }
                //get initial values
                else if (init == false)
                {
                    if (line.substr(0, 11) == "init_vector")
                    {
                        line.erase(0, 12);
                        init_vector = line;
                        init = true;
                        input_value = stoi(init_vector, nullptr, 2);
//memory allocation
                        //allocate host memory
                        host_vector = (cuFloatComplex *)malloc(size*sizeof(cuFloatComplex));
```

```
                        //allocate device memory
                        err = cudaMalloc((void **)&device_vector_1,
size*sizeof(cuFloatComplex));
                        err = cudaMalloc((void **)&device_vector_2,
size*sizeof(cuFloatComplex));
                        if (err != cudaSuccess) printf("%s\n", cudaGetErrorString(err));

                        //init input vector
                        err = cudaMemset(device_vector_1, 0, size * sizeof(cuFloatComplex));
                        host_vector[input_value] = make_cuFloatComplex(1.0, 0.0);
                        err = cudaMemcpy(&device_vector_1[input_value],
&host_vector[input_value], sizeof(cuFloatComplex), cudaMemcpyHostToDevice);
                        if (err != cudaSuccess) printf("%s\n", cudaGetErrorString(err));
                    }
                    else
                    {
                        cout << "Syntax error: In the second line you must define the value
of the initial state vector." << endl;
                        free_mem();
                        exit(EXIT_FAILURE);
                    }
                }
                else if (measure)
                {
                    if (line.substr(0, 7) == "Measure")
                    {
                        if (measure_index < measure_gates)
                        {
                            line.erase(0, 8);
                            if (line[0] == 'g')
                            {
                                line.erase(0, 1);
                                if (!(istringstream(line) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                measure_positions[measure_index] = g;
                                measure_index++;
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            cout << "More measurement gates than you have defined" << endl;
                            free_mem();
                            exit(EXIT_FAILURE);
                        }
                    }
else if (line.substr(0, 15) == "End_measurement")
                     {
//call measurement kernel
                        cudaMemcpy(device_measure_positions, measure_positions, measure_gates
* sizeof(int), cudaMemcpyHostToDevice);
                        cudaEventCreate(&start_m);
                        cudaEventCreate(&stop_m);
                        cudaEventRecord(start_m, 0);
                        if (step % 2 == 0)
                        {
                            measurement << <blocks, threads >> >(device_measure_vector,
device_measure_positions, device_vector_1, measure_gates, size);

                        }
                        else
                        {
                            measurement << <blocks, threads >> >(device_measure_vector,
device_measure_positions, device_vector_2, measure_gates, size);
                        }
                        cudaEventRecord(stop_m, 0);
                        cudaEventSynchronize(stop_m);
                              cudaEventElapsedTime(&measure_time, start_m,
stop_m);
                        cudaMemcpy(host_measure_vector, device_measure_vector, measure_size *
sizeof(float), cudaMemcpyDeviceToHost);
                    }

                    }
```

```
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                else
                {
                    //get gate string
                    gate_ptr = line.find_first_of(':');
                    temp_gate = line.substr(0, gate_ptr);
                    gate_ptr = 0;

                    //find the correct gate
                    if (temp_gate.length() == 1)
                    {
                        if ((line[0] == 'H') || (line[0] == 'X') || (line[0] == 'Y') ||
(line[0] == 'Z') || (line[0] == 'S') || (line[0] == 'T'))
                        {
                            one_qubit_gate(line, line_cnt, &gate_pos, qubits);
                            gate = line[0];
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else if (temp gate == "Rz")
                    {
                        if (line[3] == 'q')
                        {
                            line.erase(0, 4);
                            gate_ptr = line.find_first_of('k');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos = g;
                                gate = 'r';
                                line.erase(0, gate_ptr + 1);
                                if (line[0] == '-')
                                {
                                    neg = true;
                                    line.erase(0, 1);
                                }
                                if (!(istringstream(line) >> angle)) angle = 0.0;
                                angle = 2 * PI * angle;
                                if (neg)
                                {
                                    angle = -angle;
                                }
                                gate_ptr = 0;
                                temp_gate.clear();
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
```

```
                else if (temp_gate == "Rd")
                {
                    if (line[3] == 'q')
                    {
                        line.erase(0, 4);
                        gate_ptr = line.find_first_of('k');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            gate = 'r';
                            line.erase(0, gate_ptr + 1);
                            if (line[0] == '-')
                            {
                                neg = true;
                                line.erase(0, 1);
                            }
                            if (!(istringstream(line) >> angle)) angle = 0.0;
                            angle = (angle * PI) / 180.0;
                            if (neg)
                            {
                                angle = -angle;
                            }
                            gate_ptr = 0;
                            temp_gate.clear();
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                else if (temp_gate == "Rk")
                {
                    if (line[3] == 'q')
                    {
                        line.erase(0, 4);
                        gate_ptr = line.find_first_of('k');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            gate = 'r';
                            line.erase(0, gate_ptr + 1);
                            if (line[0] == '-')
                            {
                                neg = true;
                                line.erase(0, 1);
                            }
                            if (!(istringstream(line) >> int_angle)) int_angle = 0;
                            angle = (2 * PI) / (1 << int_angle);
                            if (neg)
                            {
                                angle = -angle;
                            }
                            gate_ptr = 0;
                            temp_gate.clear();
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
```

```
                else if (temp_gate == "CNOT")
                {
                    two_qubit_gate(line, line_cnt, &gate_pos, &gate_pos_2, qubits);
                    gate = 'n';
                }
                else if (temp_gate == "CZ")
                {
                    two_qubit_gate(line, line_cnt, &gate_pos, &gate_pos_2, qubits);
                    gate = 'v';
                }
                else if (temp_gate == "SWAP")
                {
                    two_qubit_gate(line, line_cnt, &gate_pos, &gate_pos_2, qubits);
                    gate = 'w';
                }
                else if (temp_gate == "CRz")
                {
                    if (line[4] == 'q')
                    {
                        line.erase(0, 5);
                        gate_ptr = line.find_first_of('q');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate pos = g;
                            line.erase(0, gate_ptr + 1);
                            gate_ptr = line.find_first_of('k');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos_2 = g;
                                line.erase(0, gate_ptr + 1);
                                if (line[0] == '-')
                                {
                                    neg = true;
                                    line.erase(0, 1);
                                }
                                if (!(istringstream(line) >> angle)) angle = 0.0;
                                angle = 2 * PI * angle;
                                if (neg)
                                {
                                    angle = -angle;
                                }
                                gate = 'u';
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                            gate_ptr = 0;
                            temp_gate.clear();
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
```

```
                else if (temp_gate == "CRd")
                {
                    if (line[4] == 'q')
                    {
                        line.erase(0, 5);
                        gate_ptr = line.find_first_of('q');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            line.erase(0, gate_ptr + 1);
                            gate_ptr = line.find_first_of('k');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos_2 = g;
                                line.erase(0, gate_ptr + 1);
                                if (line[0] == '-')
                                {
                                    neg = true;
                                    line.erase(0, 1);
                                }
                                if (!(istringstream(line) >> angle)) angle = 0.0;
                                angle = (angle * PI) / 180.0;
                                if (neg)
                                {
                                    angle = -angle;
                                }
                                gate = 'u';
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                            gate_ptr = 0;
                            temp_gate.clear();
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                 else if (temp_gate == "CRk")
                {
                    if (line[4] == 'q')
                    {
                        line.erase(0, 5);
                        gate_ptr = line.find_first_of('q');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            line.erase(0, gate_ptr + 1);
                            gate_ptr = line.find_first_of('k');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                        gate_pos_2 = g;
                                line.erase(0, gate_ptr + 1);
                                if (line[0] == '-')
                                {
                                    neg = true;
                                    line.erase(0, 1);
                                }
```

```
                                    if (!(istringstream(line) >> int_angle)) int_angle = 0;
                                    gate = 'u';
                                    angle = (2 * PI) / (1 << int_angle);
                                    if (neg)
                                    {
                                        angle = -angle;
                                    }
                                }
                                else
                                {
                                    syntax_error(line_cnt);
                                }
                                gate_ptr = 0;
                                temp_gate.clear();
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else if (temp_gate == "Tof")
                    {
                        three_qubit_gate(line, line_cnt, &gate_pos, &gate_pos_2, &gate_pos_3,
qubits);
                        gate = 'o';
                    }
                    else if (temp_gate == "Fred")
                    {
                        three_qubit_gate(line, line_cnt, &gate_pos, &gate_pos_2, &gate_pos_3,
qubits);
                        gate = 'f';
                    }
                    else if (temp_gate == "CCRz")
                    {
                        if (line[5] == 'q')
                        {
                            line.erase(0, 6);
                            gate_ptr = line.find_first_of('q');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos = g;
                                line.erase(0, gate_ptr + 1);
                                gate_ptr = line.find_first_of('q');
                                if (gate_ptr <= line.length())
                                {
                                    temp_gate = line.substr(0, gate_ptr);
                                    if (!(istringstream(temp_gate) >> g)) g = 0;
                                    chech_number_of_qubits(qubits, g, line_cnt);
                                    gate_pos_2 = g;
                                    line.erase(0, gate_ptr + 1);
                                    gate_ptr = line.find_first_of('k');
                                    if (gate_ptr <= line.length())
                                    {
                                        temp_gate = line.substr(0, gate_ptr);
                                        if (!(istringstream(temp_gate) >> g)) g = 0;
                                        chech_number_of_qubits(qubits, g, line_cnt);
                                        gate_pos_3 = g;
                                        line.erase(0, gate_ptr + 1);
                                        if (line[0] == '-')
                                        {
                                            neg = true;
                                            line.erase(0, 1);
                                        }
                                        if (!(istringstream(line) >> angle)) angle = 0.0;
                                        angle = 2 * PI * angle;
                                        if (neg)
                                        {
                                            angle = -angle;
                                        }
                                        gate = 'k';
```

```
                                    }
                                else
                                {
                                    syntax_error(line_cnt);
                                }
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                else if (temp_gate == "CCRd")
                {
                    if (line[5] == 'q')
                    {
                        line.erase(0, 6);
                        gate_ptr = line.find_first_of('q');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            line.erase(0, gate_ptr + 1);
                            gate_ptr = line.find_first_of('q');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos_2 = g;
                                line.erase(0, gate_ptr + 1);
                                gate_ptr = line.find_first_of('k');
                                if (gate_ptr <= line.length())
                                {
                                    temp_gate = line.substr(0, gate_ptr);
                                    if (!(istringstream(temp_gate) >> g)) g = 0;
                                    chech_number_of_qubits(qubits, g, line_cnt);
                                    gate_pos_3 = g;
                                    line.erase(0, gate_ptr + 1);
                                    if (line[0] == '-')
                                    {
                                        neg = true;
                                        line.erase(0, 1);
                                    }
                                    if (!(istringstream(line) >> angle)) angle = 0.0;
                                    angle = (angle * PI) / 180.0;
                                    if (neg)
                                    {
                                        angle = -angle;
                                    }
                                    gate = 'k';
                                }
                                else
                                {
                                    syntax_error(line_cnt);
                                }
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
```

```
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                else if (temp_gate == "CCRk")
                {
                    if (line[5] == 'q')
                    {
                        line.erase(0, 6);
                        gate_ptr = line.find_first_of('q');
                        if (gate_ptr <= line.length())
                        {
                            temp_gate = line.substr(0, gate_ptr);
                            if (!(istringstream(temp_gate) >> g)) g = 0;
                            chech_number_of_qubits(qubits, g, line_cnt);
                            gate_pos = g;
                            line.erase(0, gate_ptr + 1);
                            gate_ptr = line.find_first_of('q');
                            if (gate_ptr <= line.length())
                            {
                                temp_gate = line.substr(0, gate_ptr);
                                if (!(istringstream(temp_gate) >> g)) g = 0;
                                chech_number_of_qubits(qubits, g, line_cnt);
                                gate_pos_2 = g;
                                line.erase(0, gate ptr + 1);
                                gate_ptr = line.find_first_of('k');
                                if (gate_ptr <= line.length())
                                {
                                    temp_gate = line.substr(0, gate_ptr);
                                    if (!(istringstream(temp_gate) >> g)) g = 0;
                                    if (g > qubits)
                                        chech_number_of_qubits(qubits, g, line_cnt);
                                    gate_pos_3 = g;
                                    line.erase(0, gate_ptr + 1);
                                    if (line[0] == '-')
                                    {
                                        neg = true;
                                        line.erase(0, 1);
                                    }
                                    if (!(istringstream(line) >> angle)) int_angle = 0;
                                    angle = (2 * PI) / (1 << int_angle);
                                    if (neg)
                                    {
                                        angle = -angle;
                                    }
                                    gate = 'k';
                                }
                                else
                                {
                                    syntax_error(line_cnt);
                                }
                            }
                            else
                            {
                                syntax_error(line_cnt);
                            }
                        }
                        else
                        {
                            syntax_error(line_cnt);
                        }
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                }
                else if (temp_gate == "Measurement")
                {
                    measure = true;
                    line.erase(0, 12);
                    if (!(istringstream(line) >> measure_gates)) measure_gates = 0;
```

```
                        if (measure_gates > max_qubits - 1)
                        {
                            cout << "Too many measurement gates. You can use up to " <<
max_qubits - 1 << " on this GPU" << endl;
                            exit(EXIT_FAILURE);
                        }

                        measure_size = 1 << measure_gates;

                        measure_positions = (int *)malloc(measure_gates * sizeof(int));
                        host_measure_vector = (float *)malloc(measure_size * sizeof(float));

                        err = cudaMalloc((void **)&device_measure_positions, measure_gates *
sizeof(int));
                        err = cudaMalloc((void **)&device_measure_vector, measure_size *
sizeof(float));
                        err = cudaMemset(device_measure_vector, 0.0, measure_size *
sizeof(float));
                        if (err != cudaSuccess) printf("%s\n", cudaGetErrorString(err));
                    }
                    else
                    {
                        syntax_error(line_cnt);
                    }
                    gate_ptr = 0;
                    temp_gate.clear();
                    neg = false;
//call kernel

                    if (measure == false)
                    {
                        cudaEventCreate(&start);
                        cudaEventCreate(&stop);
                        cudaEventRecord(start, 0);
                        cudaProfilerStart();
                        if (step % 2 == 0)
                        {
                            computation << <blocks, threads >> >(device_vector_2,
device_vector_1, size, gate_pos, gate_pos_2, gate_pos_3, gate, angle);
                        }
                        else
                        {
                            computation << <blocks, threads >> >(device_vector_1,
device_vector_2, size, gate_pos, gate_pos_2, gate_pos_3, gate, angle);
                        }
                        cudaProfilerStop();
                        cudaEventRecord(stop, 0);
                        cudaEventSynchronize(stop);
                        cudaEventElapsedTime(&timing, start, stop);
                        computation_time += timing;
                        timing = 0.0;

                        cudaError_t errSync = cudaGetLastError();
                        cudaError_t errAsync = cudaDeviceSynchronize();
                        if (errSync != cudaSuccess)
                            printf("Sync kernel error: %s\n", cudaGetErrorString(errSync));
                        if (errAsync != cudaSuccess)
                            printf("Async kernel error: %s\n", cudaGetErrorString(errAsync));

                        step++;
                    }
                }
            }
        }
        myfile.close();
    }
    else
    {
        cout << "Unable to open file" << endl;
        free_mem();
        exit(EXIT_FAILURE);
    }
}
```

```
//main function
int main(int argc, char *argv[])
{
    int nDevices;
    cudaDeviceProp prop;
    vector<int> v;
    clock_t start_timer, start_write_results, start_write_measure;
    double duration, time_write_r, time_write_m;
    string path;

    //command line arguments
    if( argc == 3 )
    {
        if(argv[1] == string("-i"))
        {
            path = argv[2];
        }
        else
        {
            cout << "The first argument should be -i." << endl;
            exit(EXIT_FAILURE);
        }
    }
    else if( argc == 5 )
    {
        if(argv[1] == string("-i"))
        {
            path = argv[2];
            if(argv[3] == string("-m"))
            {
                m_argument = true;
                m_results = argv[4];
            }
            else if(argv[3] == string("-v"))
            {
                v_argument = true;
                v_results = argv[4];
            }
            else
            {
                cout << "The third argument should be -m or -v." << endl;
                exit(EXIT_FAILURE);
            }
        }
        else
        {
            cout << "The first argument should be -i." << endl;
            exit(EXIT_FAILURE);
        }
    }
    else if( argc == 7 )
    {
        if(argv[1] == string("-i"))
        {
            path = argv[2];
            if(argv[3] == string("-m"))
            {
                m_argument = true;
                m_results = argv[4];
                if(argv[5] == string("-v"))
                {
                    v_argument = true;
                    v_results = argv[6];
                }
                else
                {
                    cout << "The fifth argument should be -v." << endl;
                    exit(EXIT_FAILURE);
                }
            }
            else if(argv[3] == string("-v"))
            {
                v_argument = true;
                v_results = argv[4];
                if(argv[5] == string("-m"))
                {
                    m_argument = true;
                    m_results = argv[6];
```

```
                }
                else
                {
                    cout << "The fifth argument should be -m." << endl;
                    exit(EXIT_FAILURE);
                }
            }
            else
            {
                cout << "The third argument should be -m or -v." << endl;
                exit(EXIT_FAILURE);
            }

        }
        else
        {
            cout << "The first argument should be -i." << endl;
            exit(EXIT_FAILURE);
        }
    }
    else
    {
        cout << "The number of command line arguments is wrong." << endl;
        exit(EXIT_FAILURE);
    }

    //get number of devices available
    err = cudaGetDeviceCount(&nDevices);
    if (err != cudaSuccess) printf("%s\n", cudaGetErrorString(err));

    //get device properties
    err = cudaGetDeviceProperties(&prop, 0);
    if (err != cudaSuccess) printf("%s\n", cudaGetErrorString(err));

    //device's memory
    unsigned long long globMem = prop.totalGlobalMem;

    //device's compute capability
    major = prop.major;
    minor = prop.minor;

    //compute max qubits
    globMem /= 8;
    max_qubits = log2(globMem) - 2;

    maxThreadsPerBlock = prop.maxThreadsPerBlock;
    multiProcessorCount = prop.multiProcessorCount;

    //start timer
    start_timer = clock();

    parser(path);

    //transfer vectors from device to host
    if (step % 2 == 0)
    {
        cudaMemcpy(host_vector, device_vector_1, size * sizeof(cuFloatComplex),
cudaMemcpyDeviceToHost);
    }
    else
    {
        cudaMemcpy(host_vector, device_vector_2, size * sizeof(cuFloatComplex),
cudaMemcpyDeviceToHost);
    }

    duration = (std::clock() - start_timer) / (double)CLOCKS_PER_SEC;

    //print cuda execution time
    cout << "Computation Time on GPU: " << computation_time << "ms" << endl;
    cout << endl;

    //print measurement time
    if(measure)
    {
        cout << "Measurement Time on GPU : " << measure_time << "ms" << endl;
        cout << endl;
    }
```

```
    //print cuda execution time
    cout << "Computation Time in total: " << duration << "s" << endl;
    cout << endl;

    cout << "Waiting for results to be written to file..." << endl;
    cout << endl;

    //write results to files
    if(v_argument == true)
    {
        string s;
        int nFiles = 1;
        int cnt_entries = 0;
        char buffer[10];
        start_write_results = clock();
        int n = sprintf(buffer, "%d", nFiles);
        s = v_results + (string)buffer + ".txt";
        ofstream outfile(s);

        for (int i = 0; i < size; ++i)
        {
            if (cnt_entries == 1048576)
            {
                outfile.close();
                cnt_entries = 0;
                nFiles++;
                n = sprintf(buffer, "%d", nFiles);
                s = v_results + (string)buffer + ".txt";
                outfile.open(s);
            }
            if (cuCrealf(host_vector[i]) != 0 || cuCimagf(host_vector[i]) != 0)
            {
                cnt_entries++;
                for (int j = 0; j < qubits; ++j)
                {
                    if (i & (int)powf(2, j))
                        v.push_back(1);
                    else
                        v.push_back(0);
                }
                outfile.width(12);
                outfile << right << cuCrealf(host_vector[i]);
                outfile << " + ";
                outfile.width(12);
                outfile << right << cuCimagf(host_vector[i]) << "i    ";
                outfile << right << "|";
                for (int k = 0; k < qubits; ++k)
                {
                    outfile << v[v.size() - k - 1];
                }
                outfile << ">    " << (powf(cuCrealf(host_vector[i]), 2) +
powf(cuCimagf(host_vector[i]), 2)) * 100 << "%" << endl;
            }
            v.clear();
        }
        outfile << endl;
        outfile.close();
        time_write_r = (std::clock() - start_write_results) / (double)CLOCKS_PER_SEC;

        //print writing on file time(results)
        cout << "Writing results to file: " << time_write_r << "s" << endl;
        cout << endl;
    }
```

```
    if ((measure == true) && (m_argument == true))
    {
        start_write_measure = clock();
        string s;
        int nFiles = 1;
        int cnt_entries = 0;
        char buffer[10];
        int n = sprintf(buffer, "%d", nFiles);
        s = m_results + (string)buffer + ".txt";
        ofstream outfile(s);

        for (int i = 0; i < measure_size; ++i)
        {
            if (cnt_entries == 1048576)
            {
                outfile.close();
                cnt_entries = 0;
                nFiles++;
                n = sprintf(buffer, "%d", nFiles);
                s = m_results + (string)buffer + ".txt";
                outfile.open(s);
            }
            cnt_entries++;
            outfile << "P(" << i << ") = " << host_measure_vector[i] << endl;
        }
        outfile.close();
        time_write_m = (std::clock() - start_write_measure) / (double)CLOCKS_PER_SEC;

        //print writing on file time(measurement)
        cout << "Writing measurement results to file: " << time_write_m << "s" << endl;
        cout << endl;
    }

    //destroy events
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cudaEventDestroy(start_m);
    cudaEventDestroy(stop_m);

    //free memory
    if(measure)
    {
        free_mem_all();
    }
    else
    {
        free_mem();
    }

    cout << "Done!" << endl;
    cout << endl;

    //system("PAUSE");
    return 0;
}
```

ΠΑΡΑΡΤΗΜΑ ΙΙ

Στο δεύτερο παράρτημα παρουσιάζονται μερικά παραδείγματα από τους κώδικες περιγραφής κυκλωμάτων που προσομοιώθηκαν στο κεφάλαιο 4.

Ο κώδικας ενός κυκλώματος QFT για 9 qubits:

```
qubits: 9
init_vector: 000000000

H: q8
CRk: q7, q8, k2
CRk: q6, q8, k3
CRk: q5, q8, k4
CRk: q4, q8, k5
CRk: q3, q8, k6
CRk: q2, q8, k7
CRk: q1, q8, k8
CRk: q0, q8, k9
H: q7
CRk: q6, q7, k2
CRk: q5, q7, k3
CRk: q4, q7, k4
CRk: q3, q7, k5
CRk: q2, q7, k6
CRk: q1, q7, k7
CRk: q0, q7, k8
H: q6
CRk: q5, q6, k2
CRk: q4, q6, k3
CRk: q3, q6, k4
CRk: q2, q6, k5
CRk: q1, q6, k6
CRk: q0, q6, k7
H: q5
CRk: q4, q5, k2
CRk: q3, q5, k3
CRk: q2, q5, k4
CRk: q1, q5, k5
CRk: q0, q5, k6
H: q4
CRk: q3, q4, k2
CRk: q2, q4, k3
CRk: q1, q4, k4
CRk: q0, q4, k5
H: q3
CRk: q2, q3, k2
CRk: q1, q3, k3
CRk: q0, q3, k4
H: q2
CRk: q1, q2, k2
CRk: q0, q2, k3
H: q1
CRk: q0, q1, k2
H: q0
```

Ο κώδικας για έναν αθροιστή QFT που προσθέτει τις τιμές $(|4\rangle + |6\rangle) + |1\rangle$. Η πύλη Hadamard στην αρχή δημιουργεί την υπέρθεση.

```
qubits: 8
init_vector: 01000001

H: q5

H: q3
CRk: q2, q3, k2
CRk: q1, q3, k3
CRk: q0, q3, k4
H: q2
CRk: q1, q2, k2
CRk: q0, q2, k3
H: q1
CRk: q0, q1, k2
H: q0
SWAP: q0, q3
SWAP: q1, q2

CRk: q7, q0, k1
CRk: q6, q1, k1
CRk: q5, q2, k1
CRk: q4, q3, k1
CRk: q6, q0, k2
CRk: q5, q1, k2
CRk: q4, q2, k2
CRk: q5, q0, k3
CRk: q4, q1, k3
CRk: q4, q0, k4

SWAP: q0, q3
SWAP: q1, q2
H: q0
CRk: q0, q1, k-2
H: q1
CRk: q0, q2, k-3
CRk: q1, q2, k-2
H: q2
CRk: q0, q3, k-4
CRk: q1, q3, k-3
CRk: q2, q3, k-2
H: q3
```

Ο κώδικας για ένα αθροιστή VBE που εκτελεί την πράξη $|7\rangle + |10\rangle$:

```
qubits: 13
init_vector: 0000011101010

Tof: q5, q0, q10
CNOT: q5, q0
Tof: q9, q0, q10
Tof: q6, q1, q11
CNOT: q6, q1
Tof: q10, q1, q11
Tof: q7, q2, q12
CNOT: q7, q2
Tof: q11, q2, q12
Tof: q8, q3, q4
CNOT: q8, q3
Tof: q12, q3, q4
CNOT: q8, q3
CNOT: q8, q3
CNOT: q12, q3
Tof: q11, q2, q12
CNOT: q7, q2
Tof: q7, q2, q12
CNOT: q7, q2
CNOT: q11, q2
Tof: q10, q1, q11
CNOT: q6, q1
Tof: q6, q1, q11
CNOT: q6, q1
CNOT: q10, q1
Tof: q9, q0, q10
CNOT: q5, q0
Tof: q5, q0, q10
CNOT: q5, q0
CNOT: q9, q0
```

ΑΝΑΦΟΡΕΣ

[1] Mikhail Smelyanskiy, Nicolas P.D. Sawaya, and Alan Aspuru-Guzik, *qHiPSTER: The Quantum High Performance Software Testing Environment*, May 2016, https://arxiv.org/abs/1601.07195.
[2] Artur Ekert, Patrick Hayden and Hitoshi Inamori, *Basic Concepts in Quantum Computation*, Feb 2008.
[3] N. David Mermin, *Quantum Computer Science*, Cambridge University Press,2007.
[4] Ιωάννης Καραφυλλίδης, *Κβαντική Υπολογιστική*, Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα «Κάλλιπος», 2015.
[5] Thomas G. Draper, *Addition on a Quantum Computer*, Sep. 1998, https://arxiv.org/abs/quant-ph/0008033.
[6] Michael A. Nielsen, Isaac L. Chuang, *Quantum Computation and Quantum Information,* Cambridge University Press, June 2012.
[7] Stephane Beauregard, *Circuit for Shor's algorithm using 2n+3 qubits*, Quantum Information & Computation, Volume 3 Issue 2, March 2003, Pages 175-185.
[8] Vlatko Vedral, Adriano Barenco, Artur Ekert, *Quantum networks for elementary arithmetic operations*, Phys. Rev. A 54, 147, Jul 1996
[9] QASM, https://www.media.mit.edu/quanta/qasm2circ/.
[10] Davy Wybiral, Jiman Hwang, *Quantum Circuit Simulator*, http://www.davyw.com/quantum/.
[11] Piotr Danilewski, *CUDA Memory Hierarchy*, Saarland University, Oct. 2012.
[12] Jason Sanders, Edward Kandrot, *CUDA by Example*, Addison-Wesley, 2010.
[13] NVIDIA, CUDA C Programming Guide, 2012.
[14] Cliff Woolley, NVIDIA, *GPU Optimization Fundamentals*.
[15] CUDA Toolkit Documentation, http://docs.nvidia.com/cuda/.
[16] Blogspot for CUDA Programming, http://cuda-programming.blogspot.gr.