

Villám Barna

Kvantumszámítógép Szimulálása Groq Nyelvi Feldolgozóegységeken



ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Témavezető: Dr. Rakyta Péter

Tartalomjegyzék

1. Bevezető	2
1.1. Motiváció	2
1.2. Kvantuminformatikai műveletek	2
1.2.1. Mérés	3
1.2.2. Bloch-gömb	3
1.2.3. Egy qubiten ható kapuk	4
1.2.4. Két qubites műveletek	8
1.3. Összefonódás, mint kvantuminformatikai erőforrás	9
1.3.1. N qubites eset	11
1.3.2. No cloning tétel	11
2. Groq chip felépítése	12
2.1. Memória (MEM)	13
2.2. Vektor multiplikációs egység (VXM)	13
2.3. SXM	13
2.4. Mátrix multiplikációs egység (MXM)	13
2.5. Groq API	13
3. A szimuláció működésének leírása	14
3.1. Transzformáció	14
3.2. Transzformáció vezérelt kapukkal	16
3.3. Információtárolás a chipen	17
3.3.1. Az állapotvektor eltárolása	17
3.3.2. Permutációs térképek	18
3.3.3. State0 és state1 tenzorok	19
3.3.4. Mem gather és scatter map-ok	20
3.4. A szimuláció futásának menete	20
4. Számítási teljesítmény	23
5. További fejlesztési lehetőségek	24
6. Összegzés	26

1. Bevezető

A kvantumszámítógépek klasszikus szimulációja alapvető szerepet játszik a kvantuminformatika fejlődésében, mind a kvantumalgoritmusok numerikus tanulmányozásában, mind a zaj és a hibák modellezésében. Az egyik ilyen felhasználási mód a variációs kvantum algoritmusok (VQA) futtatása, ahol jellemzően paraméterezett kvantumáramköröket tanítunk klasszikus optimalizációs ciklusokon keresztül. A 20 qubites határ feletti rendszereket hatékonyan tudjuk szimulálni GPU-k segítségével, viszont ez alatt ha a CPU-s szimulációknál hatékonyabb teljesítményt szeretnénk elérni, akkor más megoldásra van szükségünk[3]. A dolgozatban bemutatok egy ilyen 22 qubit alatti szimulációt, melyet a nagy nyelvi modellek területén ismertté vált Groq Nyelvi Feldolgozóegység (Groq LPU) architektúrára fejlesztettünk.

1.1. Motiváció

A VQA algoritmusok problémaköréhez kapcsolódó nagy számítási-teljesítmény igény és a tanítások során fellépő egyéb elvi nehézségek miatt az irodalomban publikált eredmények csak ritka esetekben érik el a 20 qubites határt, többnyire kis rendszerekre korlátozódnak. A szükséges számítási-teljesítmény biztosításában ezen a méretskálán, a GPU alapú technológia sem nyújt kellő hatékonyságú megoldást[3]. Annak érdekében, hogy felgyorsítsuk a VQA szimulációk sebességét (és ezzel felgyorsíthassuk a kvantum gépi tanulásos algoritmusok kutatását), a Groq LPU architektúrára fejlesztünk kvantumszámítógép szimulátort, mellyel előzetes becsléseink alapján lehetővé válik nagyobb kvantum rendszerek vizsgálata is VQA kontextusban.

1.2. Kvantuminformatikai műveletek

A szimuláció bemutatása előtt rövid bevezető következik a kvantuminformatikai műveletekbe.

Egy kvantum bit (röviden qubit) a kvantuminformatika alapvető információs egysége[4]. Egy önmagában lévő qubit elképzelhető úgy mint egy kétállapotú rendszer. Kvantummechanikában elterjedt jelölésmód az, hogy az oszlopvektorokat $|v\rangle$ -el és a sorvektorokat $\langle v|$ -vel jelöljük most is ezt a jelölt fogjuk használni. Egy qubitet egy két dimenziós Hilbert-téren lévő vektorként reprezentálhatunk

$$|\Psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \quad (1)$$

(ezt nevezzük állapotvektornak), ahol $a, b \in \mathbb{C}$, egy qubitnek mint két állapotú rendszer van két ortogonális állapota ezeket egy a Hilbert-téren található ortonormált bázisvektor párokkal reprezentálunk amiket a

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (2)$$

és

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (3)$$

-el jelöljük. Ezekkel a jelölésekkel az állapotvektor felírható a következő módon:

$$|\Psi\rangle = a|0\rangle + b|1\rangle \quad (4)$$

Mivel kvantummechanikai rendszerről beszélünk ezért a hullámfüggvényünk (most állapotvektor) egyre kell, hogy normálva legyen, ez azért van mert, a és b abszolút értékének négyzete mondja meg, hogy megfigyeléskor milyen valószínűséggel milyen állapotban találjuk a qubitet tehát ahhoz, hogy a teljes valószínűség egy legyen az 5. egyenletnek teljesülnie kell.

$$|\Psi|^2 = \langle\Psi|\Psi\rangle = |a|^2 + |b|^2 = a \cdot a^* + b \cdot b^* = 1, \quad (5)$$

itt a^* a komplex konjugálást jelenti és $|a|^2$ azt mondja meg, hogy megfigyelés után mekkora eséllyel lesz a qubit a $|0\rangle$ állapotban és $|b|^2$ pedig azt, hogy mekkora eséllyel lesz az $|1\rangle$ állapotban. Ha egy két qubit által alkotott rendszerről beszélünk akkor az a két qubitet tartalmazó vektortér tenzor szorzatában található meg, ez azt jelenti, hogy egy N qubitből álló rendszert egy 2^N dimenziós Hilbert-tér tartalmazza. Egy egyedülálló qubitet transzformálni tudunk a klasszikus informatikához hasonlóan logikai kapuk segítségével ezek lehetnek bármilyen 2×2 -es unitér mátrixal reprezentálható transzformációk. Ennek egy fontos következménye az, hogy minden kvantum logikai kapu invertálható, tehát ezek a transzformációk reverzibilisek, klasszikus informatikában csak NOT művelet volt ilyen. Ha N qubitet szeretnénk transzformálni, akkor $2^N \times 2^N$ -es unitér mátrixra lenne szükségünk. Megjelennek a klasszikus informatikából is ismert vezérelt kapuk is ahol egy másik qubit állapotától függően alkalmazzuk a transzformációt az adott qubiten.

1.2.1. Mérés

Fentebb megjelent, hogy megfigyeléskor valamilyen állapotban találjuk a qubitet, ez a tulajdonság a kvantummechanikából jön és ennek a jelenségnek a működését még nem értjük teljesen. Fontos, hogy a kvantum logikai kapukkal ellentétben a megfigyelés nem reverzibilis folyamat.

1.2.2. Bloch-gömb

A (4). egyenletet mivel a és b komplex számok felírhatjuk a következő módon:

$$\begin{aligned} |\Psi\rangle &= r_1 e^{i\varphi_1} |0\rangle + r_2 e^{i\varphi_2} |1\rangle = \\ &= e^{i\varphi_1} \left(r_1 |0\rangle + r_2 e^{i(\varphi_2 - \varphi_1)} |1\rangle \right), \end{aligned} \quad (6)$$

ahol $\varphi_1, \varphi_2 \in [0, 2\pi[$ és $r_1 = |a|$, $r_2 = |b|$, mivel $e^{i\varphi}$ abszolút értéke egy így nem befolyásolja egyik állapotnak az együtthatóját sem, tehát ez nem hat ki a méréseinkre[5], ennek eredményeként elhagyhatjuk a képletből, ezen kívül az (5). képletből tudjuk, hogy $r_1^2 + r_2^2 = 1$ tehát tudunk találni olyan Θ paramétert, hogy

$$\begin{aligned} r_1 &= \cos \frac{\Theta}{2} \\ r_2 &= \sin \frac{\Theta}{2}, \end{aligned} \quad (7)$$

itt $\Theta \in [0, \pi]$. Legyen a relatív fázis $\varphi \in [0, 2\pi[$

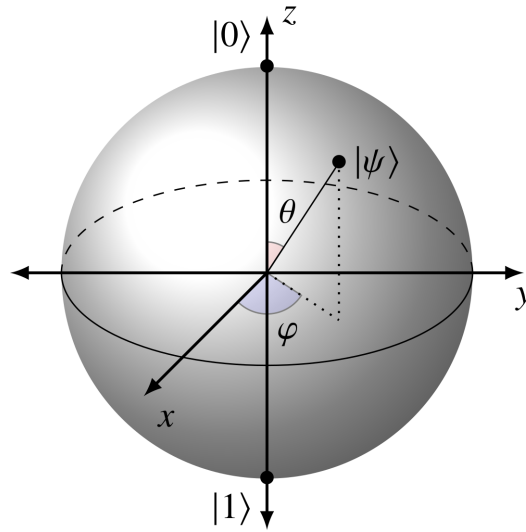
$$\varphi = \varphi_2 - \varphi_1, \quad (8)$$

ha ez kisebb lenne mint nulla, akkor hozzáadunk annyiszor π -t, hogy az ne legyen negatív. Az állapotvektor felírható Θ és φ paraméterekkel (ezt a reprezentációt ugyanúgy jelöljük mint a $\mathbb{C}^2$ -n lévőket, kontextustól függ, hogy melyiket értjük alatta):

$$|\Psi\rangle = \cos\left(\frac{\Theta}{2}\right)|0\rangle + \sin\left(\frac{\Theta}{2}\right)e^{i\varphi}|1\rangle \quad (9)$$

A két szög leír egy pontot egy egységkörön, ezt hívjuk Bloch-gömbnek a Nobel-díjas fizikus Felix Bloch után. θ -t indítsuk a z-tengelyt pozitív félegyenesétől és φ -t az x-tengely pozitív félegyenesétől a negatív körbelyárási irányba. Látható, hogy most két paraméterrel le tudjuk írni a qubit állapotát, holott eredetileg a (4). egyenletben négyet használtunk (a -nak és b -nek is van valós és imaginárius része), ez úgy lehet, hogy valójában az (5). egyenlet miatt már csak háromra van szükség és mivel az abszolút fázis nem mérhető így csak kettőre.

Vizsgáljuk meg ennek a leírásnak pár tulajdonságát. Mikor van $|\Psi\rangle$ tisztán a $|0\rangle$ állapotban? Ehez az kell, hogy $\cos\left(\frac{\theta}{2}\right)$ egy legyen és ha megfelelően paramétereztünk akkor $|1\rangle$ együtthatója nulla lesz. A $\theta = 0$ esetben mind a kettő teljesül. A tisztán $|1\rangle$ állapothoz $\theta = \pi$ -re van szükségünk, tehát láthatjuk, hogy a tisztán $|0\rangle$ és $|1\rangle$ állapotok az 1. ábrának megfelelően a Bloch-gömb pólusain találhatóak.



1. ábra. Bloch-gömb[5]

1.2.3. Egy qubiten ható kapuk

Amint már feljebb leírtam az egy qubiten ható kapuk 2×2 -es mátrixokkal reprezentálhatók. Az alábbiakban a leggyakrabban használt kapukat mutatjuk be.

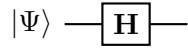
Kvantumos H kapu

A Jacques Hadamardról elnevezett Hadamard kapu mátrixa a következő:

$$\mathbf{H} = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (10)$$

Az áramköri rajzon való ábrázolása a 2. ábrán látható. Nézzük meg ez hogyan hat a két

2. ábra. H kapu áramköri rajza



bázisunkra:

$$\mathbf{H} |0\rangle = \frac{\sqrt{2}}{2} (|0\rangle + |1\rangle) \quad (11)$$

$$\mathbf{H} |1\rangle = \frac{\sqrt{2}}{2} (|0\rangle - |1\rangle) \quad (12)$$

A (11). egyenlet eredményét nevezzük el $|+\rangle$ -nak és a (12). egyenletét $|-\rangle$. Belátható, hogy ezek lineárisan függetlenek mivel egymással vett skalárszorzatuk nulla, tehát bázist alkotnak a Hilbert-terünkön.

A Bloch-gömbös reprezentációban a két új bázisvektorunk közül $|+\rangle$ az x-tengely pozitív irányába $|-\rangle$ pedig a negatív irányába mutat. A Hadamard kapu az egyik leggyakrabban használt kapu kvantum infomatikába [5], gyakran használjuk arra, hogy a $|0\rangle$ -ra inicializált qubitet kevert állaptra tegyük [5]. Az előzőekben említetteken kívül még szoktuk használni a

$$|i\rangle = \frac{\sqrt{2}}{2} (|0\rangle + i |1\rangle) \quad (13)$$

$$|-i\rangle = \frac{\sqrt{2}}{2} (|0\rangle - i |1\rangle) \quad (14)$$

bázisokat is amelyek Bloch-gömbön való reprezentációban rendre az y-tengely pozitív és negatív irányába mutatnak.

Kvantumos X kapu

Az X kapu a Pauli X mátrixal reprezentálható:

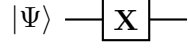
$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (15)$$

Ha áramköri rajzot készítünk akkor az $\mathbf{X}$ kaput a 3. ábra alapján ábrázoljuk. Ennek a kapunak alapvető tulajdonsága, hogy:

$$\mathbf{X} |0\rangle = 1$$

$$\mathbf{X} |1\rangle = 0$$

3. ábra. X kapu áramköri rajza



Látható, hogy ez analóg a klasszikus informatikában használt NOT kapuval

$$\mathbf{X}|\Psi\rangle = b|0\rangle + a|1\rangle \quad (16)$$

Az $\mathbf{X}$ kapu megcseréli az esélyét, hogy megfigyeléskor a qubit melyik állapotban lesz. Most nézzük meg, hogyan látszik a kapu hatása a Bloch-gömbös reprezentációban.

$$\begin{aligned} \mathbf{X}|\Psi\rangle &= \sin\left(\frac{\Theta}{2}\right)e^{i\varphi} |0\rangle + \cos\left(\frac{\Theta}{2}\right) |1\rangle = \\ &= e^{i\varphi} \left(\cos\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) |0\rangle + \sin\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) e^{-i\varphi} |1\rangle \right) = \\ &= \cos\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) |0\rangle + \sin\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) e^{-i\varphi} |1\rangle = \\ &= \cos\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) |0\rangle + \sin\left(\frac{\pi}{2} - \frac{\Theta}{2}\right) e^{i\varphi} |1\rangle \end{aligned} \quad (17)$$

Láthatjuk, hogy a qubit fázisát nem változtatja és $\theta_{uj} = \pi - \theta$. Ez egy az x-tengely körüli π -s forgatásnak felel meg [5].

Most hatassuk az $\mathbf{X}$ kaput a $|+\rangle$ és $|-\rangle$ bázisvektorokra:

$$\mathbf{X}|+\rangle = \frac{\sqrt{2}}{2}(\mathbf{X}|0\rangle + \mathbf{X}|1\rangle) = \frac{\sqrt{2}}{2}(|1\rangle + |0\rangle) = |+\rangle \quad (18)$$

$$\mathbf{X}|-\rangle = \frac{\sqrt{2}}{2}(\mathbf{X}|0\rangle - \mathbf{X}|1\rangle) = \frac{\sqrt{2}}{2}(|1\rangle - |0\rangle) = -|-\rangle \quad (19)$$

Ebből azt láthatjuk, hogy az $\mathbf{X}$ mátrix sajátvektorai a $|+\rangle$ és a hozzátartozó sajátérték egy, és $|-\rangle$ és a hozzátartozó sajátérték mínusz egy.

Kvantumos Z kapu

Az Z kapu a Pauli Z mátrixal reprezentálható:

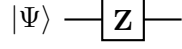
$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (20)$$

Ezt hattatva az állapotvektorra:

$$\mathbf{Z}|\Psi\rangle = a|0\rangle - b|1\rangle \quad (21)$$

Ebből mindjárt látszik is, hogy $\mathbf{Z}$ sajátvektorai a $|0\rangle$ és az $|1\rangle$, a hozzájuk tartozó sajátérték pedig rendre az egy és mínusz egy.

4. ábra. Z kapu áramköri rajza



Ez nem befolyásolja az egyes állapotok valószínűségét. Viszont a fázist igen, vizsgáljuk meg hogyan:

$$\begin{aligned} \mathbf{Z}|\Psi\rangle &= r_1|0\rangle - r_2e^{i\varphi}|1\rangle = \\ &= r_1|0\rangle + r_2e^{i\pi}e^{i\varphi}|1\rangle = \\ &= r_1|0\rangle + r_2e^{i(\varphi+\pi)}|1\rangle \end{aligned} \quad (22)$$

Azt látjuk, hogy a Z kapu eltoja a fázist π -vel ezt hívjuk *phase flip*-nek, ha ezzel nagyobb lenne mint 2π akkor kivonunk belőle 2π -t ezzel korrigálva. A Bloch-gömbön ez egy π -nyi forgatást jelent a z-tengelyen.

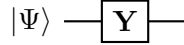
Kvantumos Y kapu

Az Y kapu a Pauli Y mátrixal reprezentálható:

$$\mathbf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad (23)$$

Ezt hattatva az állapotvektorra:

5. ábra. Y kapu áramköri rajza



$$\mathbf{Y}|\Psi\rangle = -bi|0\rangle + ai|1\rangle = -i(b|0\rangle - a|1\rangle) \quad (24)$$

Az Y kapu megcseréli a két állapot valószínűségét és ezen kívül egy *phase flip*-et is végrehajt. Ezt a Bloch-gömbön egy y-tengely körüli π -nyi forgatásként tudjuk elképzelni.

Hatassuk $\mathbf{Y}$ -t $|i\rangle$ és $|-i\rangle$ -re:

$$\mathbf{Y}|i\rangle = \frac{\sqrt{2}}{2}(\mathbf{Y}|0\rangle + i\mathbf{Y}|1\rangle) = \frac{\sqrt{2}}{2}(i|1\rangle + |0\rangle) = |i\rangle \quad (25)$$

$$\mathbf{Y}|-i\rangle = \frac{\sqrt{2}}{2}(\mathbf{Y}|0\rangle - i\mathbf{Y}|1\rangle) = \frac{\sqrt{2}}{2}(i|1\rangle - |0\rangle) = -|-i\rangle \quad (26)$$

Megtaláltuk az $\mathbf{Y}$ kapu sajátértékeit és sajátvektorait melyek az $|i\rangle$ az egy sajátértékkel és $|-i\rangle$ a mínusz egy sajátértékkel.

1.2.4. Két qubites műveletek

Mint már az első szakaszban említettem egy két qubitből álló rendszert két egy qubites vektortér tenzor szorzatában tudunk reprezentálni ami egy négy dimenziós Hilbert-teret képez. Bázisvektoroknak válasszuk a két dimenzióban is használt bázisvektoraink egymással vett tenzorszorzatát a következő jelölésmóddal:

$$\begin{aligned}
 |00\rangle &= |0\rangle_1 \otimes |0\rangle_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
 |01\rangle &= |0\rangle_1 \otimes |1\rangle_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\
 |10\rangle &= |1\rangle_1 \otimes |0\rangle_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \\
 |11\rangle &= |1\rangle_1 \otimes |1\rangle_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
 \end{aligned} \tag{27}$$

Vegyük két qubitet:

$$|\Psi\rangle_1 = a_1 |0\rangle_1 + b_1 |1\rangle_1 \tag{28}$$

$$|\Psi\rangle_2 = a_2 |0\rangle_2 + b_2 |1\rangle_2 \tag{29}$$

Számoljuk ki ezek tenzor szorzatát:

$$|\Psi\rangle_1 \otimes |\Psi\rangle_1 = a_1 a_2 |00\rangle + a_1 b_2 |01\rangle + a_2 b_1 |10\rangle + a_2 b_2 |11\rangle \tag{30}$$

Ennek az állapotvektornak szintén egyre kell lennie normálva, mivel az együtthatók még mindig az egyes állapotok valószínűségét jelentik. Ellenőrizzük ezt le az alábbi rövid számolással.

$$\begin{aligned}
 ||\Psi\rangle_1 \otimes |\Psi\rangle_1|^2 &= |a_1 a_2|^2 + |a_1 b_2|^2 + |a_2 b_1|^2 + |a_2 b_2|^2 = \\
 &= |a_1|^2 |a_2|^2 + |a_1|^2 |b_2|^2 + |a_2|^2 |b_1|^2 + |a_2|^2 |b_2|^2 = \\
 &= |a_1|^2 (|a_2|^2 + |b_2|^2) + |b_1|^2 (|a_1|^2 + |b_1|^2) = \\
 &= |a_1|^2 + |b_1|^2 = 1
 \end{aligned} \tag{31}$$

1.3. Összefonódás, mint kvantuminformatikai erőforrás

A négy dimenziós Hilbert-térben lévő vektorok nem csak egyes egy qubitet reprezentáló vektorok tenzorszorzatai lehetnek, hanem olyanok is amelyek nem írhatóak fel tenzor szorzat alakban. Ha ilyen rendszerrel találkozunk akkor beszélünk az összefonódás jelenségéről, ilyenkor a rendszer nem különíthető el két független qubitre, mivel azok állapotainak valószínűsége nem függetlenek egymástól.

Most nézzünk meg egy lehetséges kombinált állapotát két qubitnek:

$$|\Psi^+\rangle = \frac{\sqrt{2}}{2} |01\rangle + \frac{\sqrt{2}}{2} |10\rangle \quad (32)$$

Láthatjuk, hogy mérésékor a $|01\rangle$ és az $|10\rangle$ állapotnak is ugyanannyi

$$\left(\frac{\sqrt{2}}{2}\right)^2 = 0.5$$

a valószínűsége. Viszont mi történik, ha megmérjük az első qubitet és az $|1\rangle$ -es állapotban találjuk? Ekkor a teljes rendszer vagy az $|10\rangle$ vagy az $|11\rangle$ állapotban lesz viszont az utóbbinak az esély nulla, tehát a második qubit biztosan a $|0\rangle$ -ás állapotban lesz. Ezt a jelenséget hívjuk összefonódásnak. A fent használt állapotot Bell-állapotnak hívjuk[5]. Bell-állapotnak a teljesen összefonódott állapotokat nevezzük, ezekből összesen négy létezik, amelyek ortogonális rendszert alkotnak a Hilbert terünkön:

$$|\Psi^+\rangle = \frac{\sqrt{2}}{2} |01\rangle + \frac{\sqrt{2}}{2} |10\rangle \quad (33)$$

$$|\Psi^-\rangle = \frac{\sqrt{2}}{2} |01\rangle - \frac{\sqrt{2}}{2} |10\rangle \quad (34)$$

$$|\Phi^+\rangle = \frac{\sqrt{2}}{2} |00\rangle + \frac{\sqrt{2}}{2} |11\rangle \quad (35)$$

$$|\Phi^-\rangle = \frac{\sqrt{2}}{2} |00\rangle - \frac{\sqrt{2}}{2} |11\rangle \quad (36)$$

$$(37)$$

CX kapu

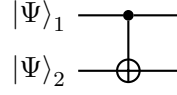
Mint azt már feljebb láttuk a két qubites kapuk lehetnek akármilyen unitér mátrixal reprezentálható két qubiten ható transzformációk és ezekbe beletartozik az egy qubites kapuk vezérelt változatai.

A **CX** kapu azaz vezérelt **X** kapu a hagyományos informatikában megtalálható CNOT kapu megfelelője, mátrixa a következőképp írható:

$$\mathbf{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (38)$$

Ez a kapu a kvantum informatika egyik legfontosabb kapuja, mivel ezt használjuk a leg-

6. ábra. CX kapu áramköri rajza



gyakrabban összefonódott állapotok létrehozásához[5].

A **CX** hatása a bázisvektorainkra:

$$\mathbf{CX} |00\rangle = |00\rangle \quad (39)$$

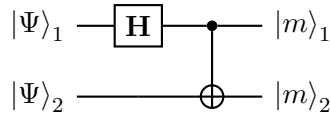
$$\mathbf{CX} |01\rangle = |01\rangle \quad (40)$$

$$\mathbf{CX} |10\rangle = |11\rangle \quad (41)$$

$$\mathbf{CX} |11\rangle = |10\rangle \quad (42)$$

A **CX** és **H** kapuk segítségével át tudunk térni a Bell-állapotok rendszerére (hasonlóan ahhoz amit egy qubitnél láttunk a **H** kapuval)

7. ábra. Áttérés Bell-állapotok rendszerére



$$\mathbf{H} |\Psi\rangle_1 = \frac{\sqrt{2}}{2} [a_1(|0\rangle + |1\rangle) + b_1(|0\rangle - |1\rangle)] \quad (43)$$

$$\begin{aligned} \mathbf{CX}(\mathbf{H} |\Psi\rangle_1 \otimes |\Psi\rangle_2) &= \mathbf{CX} \left(\frac{\sqrt{2}}{2} [a_1 a_2 (|00\rangle + |10\rangle) + a_1 b_2 (|01\rangle + |11\rangle) + \right. \\ &\quad \left. + b_1 a_2 (|00\rangle - |10\rangle) + b_1 b_2 (|01\rangle - |11\rangle)] \right) = \\ &= \frac{\sqrt{2}}{2} [a_1 a_2 (|00\rangle + |11\rangle) + a_1 b_2 (|01\rangle + |10\rangle) + \\ &\quad + b_1 a_2 (|00\rangle - |10\rangle) + b_1 b_2 (|01\rangle - |10\rangle)] = \\ &= a_1 a_1 |\Phi^+\rangle + a_1 b_2 |\Psi^+\rangle + b_1 a_2 |\Phi^-\rangle + b_1 b_2 |\Psi^-\rangle \end{aligned} \quad (44)$$

$\mathbf{H}^{\otimes 2}$ kapu

A általánosítsuk a $\mathbf{H}$ kaput két qubites kapuvá. Ezt úgy tesszük meg, hogy hattatjuk a $\mathbf{H}$ -et külön külön mindkét qubitre és megnézzük az eredmények tenzorszorátát

$$\mathbf{H}|\Psi\rangle_1 = \frac{\sqrt{2}}{2} [a_1(|0\rangle + |1\rangle) + b_1(|0\rangle - |1\rangle)] \quad (45)$$

$$\mathbf{H}|\Psi\rangle_2 = \frac{\sqrt{2}}{2} [a_2(|0\rangle + |1\rangle) + b_2(|0\rangle - |1\rangle)] \quad (46)$$

Kihasználju, a következő azonosságot[5]

$$\mathbf{H}^{\otimes 2}|\Psi\rangle_{12} = (\mathbf{H}|\Psi\rangle_1) \otimes (\mathbf{H}|\Psi\rangle_2) = (\mathbf{H} \otimes \mathbf{H})(|\Psi\rangle_1 \otimes |\Psi\rangle_2) \quad (47)$$

Ahol $|\Psi\rangle_{12} = |\Psi\rangle_1 \otimes |\Psi\rangle_2$ két qubites rendszer állapotvektora. Így:

$$\mathbf{H}^{\otimes 2} = (\mathbf{H} \otimes \mathbf{H}) = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \cdot \mathbf{H} & 1 \cdot \mathbf{H} \\ 1 \cdot \mathbf{H} & -1 \cdot \mathbf{H} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \quad (48)$$

Most hatassuk a $|00\rangle$ állapotra a mátrixunkat:

$$\mathbf{H}^{\otimes 2}|00\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \quad (49)$$

Látható, hogy az egy qubites esethez hasonlóan ez is az összes állapot egyenlő együtthatójú szuperpozíciójába helyezi a qubitet.

1.3.1. N qubites eset

Az N qubites esetben a két qubites esethez hasonlóan kell eljárunk, itt rendszert egy 2^N -es Hilbert-térben tudjuk reprezentálni, létre tudnk hozni bázisvektorokat N db egy qubites bázisvektor tenzorszorzatából. A transzformációkat itt is unitér mátrixok írják le.

Kapuk adott halmazát univerzálisnak nevezzük ha akármilyen N qubites kapu reprezentálható velük, ilyen például az összes U3 kapu kiegészítve a CNOT kapuval[2].

1.3.2. No cloning tétel

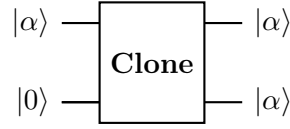
Ez a tétel azt mondja ki, hogy kvantuminformatikában, nem lehetséges egy általános kvantum állapot megbízható másolása[4].

Bizonyítás[4]: Ahhoz, hogy lemásoljuk $|\alpha\rangle$ kvantumállatot kell egy olyan U unitér operátor, hogy

$$U(|\alpha\rangle |0\rangle) = |\alpha\rangle |\alpha\rangle \quad (50)$$

ha ez minden állapotra működik akkor nem függhet α -tól, így $|\beta\rangle \neq |\alpha\rangle$ -ra

$$U(|\beta\rangle |0\rangle) = |\beta\rangle |\beta\rangle \quad (51)$$



8. ábra. Másoló áramkör

Viszont ekkor $|\gamma\rangle = (|\alpha\rangle + |\beta\rangle)/\sqrt{2}$ -re hattatva U -t:

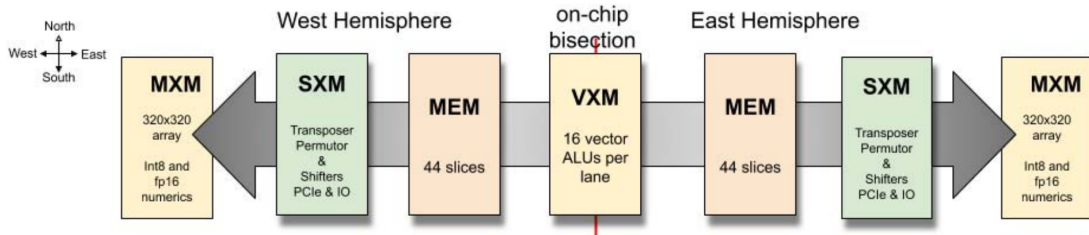
$$U(|\gamma\rangle |0\rangle) = (|\alpha\rangle |\alpha\rangle + |\beta\rangle |\beta\rangle)/\sqrt{2} \neq |\gamma\rangle |\gamma\rangle \quad (52)$$

Tehát a másolás kudarcot vallott. Megfigyelhetjük, hogy a másolás működhet egyes állapotokon, de hacsak nem tudjuk, hogy a másolandó állapotunk egy ilyen állapot akkor nem tudjuk garantálni, hogy a másolás sikerrel fog járni.

2. Groq chip felépítése

A számítás komplexitásának és mennyiségének növekedésével a hagyományos CPU és GPU architektúrák komoly skálázhatósági és teljesítménybeli problémákat mutatnak[1], sok mikroarchitektúra hardveres összetettsége megnehezíti a futásidőbeli elakadásokról való gondolkodást. Ezt próbálja megoldani a tenzor absztrakció köré épített *Language Processing unit* architektúra, innentől LPU. A tenzorműveleteket streaming feldolgozási modellben végezzük, ahol a számítási elemek térbeli elrendezése funkció szerint történik, hogy kihasználják az adatáramlás lokalitását, ahogy a tenzorok elhaladnak mellettük[1].

A chip felépítését a következő absztrakció szerint tudjuk elképzelni: A chipet bontsuk fel a három tengelye szerint x, y és z tengelyre. Az x tengelyt két féltékére osztjuk fel, egy keletire és egy nyugatira, mind a kettőn ugyanazok a funkcionális egységek találhatók meg az alábbi módon (9. ábra): A chip két szélén a *Matrix Execution Module* (MXM) található, a chipen befelé haladva a *Switch Execution Module* (SXM), és mellette a *Memory* (MEM), a két MEM modul között található a *Vector Execution Module* (VXM).



9. ábra. A chip-en lévő funkcionális egységek elrendezése[1]

Az y tengelyen minden operációt 320 *lane*-re vektorizálunk (1 byte/lane), azaz a tenzor-nak 320 elemét tudjuk párhuzamosan feldolgozni (azaz egy 320 széles SIMD), a 320 *lane*-t 16 *superlane*-re osztjuk.

Az z tengely 32 adat *stream*-et (mindegyik az y alapján 320 lane-t) foglal magában amelyeken egyidejűleg folyik az adat az egyes funkcionális egységeken keresztül (x tengely).

2.1. Memória (MEM)

A hagyományos CPU és GPU architektúrákkal ellentétben a Groq chipen a memória nem hierarchikus cache-ekbe rendezett, hanem egy egyenrangú 88 *slice*-ra (44-44 a két féltekén) osztott memóriát kapunk. Slice-onként 8192 memória *address*-t található, address-enként 320 byte, azaz egy slice nagyjából $320B \cdot 8192 \approx 2,6MB$ -nyi adatot képes tárolni és a teljes chip-en $2,6MB \cdot 88 \approx 230MB$ memória található.

A memóriából történő kiolvasás vagy írás két féle módon történhet, az egyik amikor az utasításban megszabjuk, hogy melyik *address*-ekről szeretnénk kiolvasni, a másik amikor ehhez a chipre előre feltöltött térképeket használunk. Erre akkor lehet szükség, ha egy programunkban a bemenő adattól függően kell bizonyos helyekről kiolvasni/írni, de nem szeretnénk az összes lehetséges esetre egy külön programot.

Egy *slice*-on lévő memóriát felbontuk két memória bankra az *address*-ek első fele az első a második fele a második bankba tartozik. Erre azért van szükség egy *slice*-ról egy időben csak akkor tudunk kiolvasni és írni, ha az *address*-ek amikre írunk/olvasunk eltérő bankon vannak.

2.2. Vektor multiplikációs egység (VXM)

A VXM 16 *funkcionális slice*-al (ALU-val) rendelkezik ezeken tudunk elemenkénti műveletet végrehajtani a tenzorokon. Az ALU-k fele nagy a fele kicsi, ezek közt a különbség az, hogy milyen műveleteket támogatnak. Az ALU-k négy *rank*-ba vannak csoportosítva amiken belül az ALU-kon egyidejűleg tudunk műveletekt létrehozni.

2.3. SXM

Kettő SXM *funkcionális slice* található a chip-en egy mind a két féltekén. Ezt adatmanipulációra használjuk pl. transzponálás, permutálás, stb. Ezen kívül nagyobb tenzorok esetén az MXM műveletek elvégzéséhez szükségünk lehet az SXM-re is.

2.4. Mátrix multiplikációs egység (MXM)

A chip-en két MXM *funkcionális slice* található hasonlóan az SXM-hez, mind a kettő tartalmaz két *plane*-t, egy *lower*- és egy *higher byte plane*-t, egy MXM-ben lehet végezni két független mátrix szorzást INT8-as adatokon, vagy egyet FLOAT16-oson.

2.5. Groq API

A groq chipre való kód írására Python API-t használunk. Ez alapján jön létre a gépi kód amit fel tudunk tölteni a chipre. Az API-ban megtaláljuk az utasításokat a chip különböző funkcionális egységeihez.

3. A szimuláció működésének leírása

Most a chip felépítésének bemutatása után rátérek a szimuláció felépítésére. Először bemutatom, hogy hogyan végezzük az állapotvektor transzformációját, majd rátérek ennek a gyakorlati implementálására.

3.1. Transzformáció

A technikai leírás elkezdése előtt bemutatom, hogy hogyan transzformálódik az állapotvektor, egy egy qubites kapu adott qubitre való alkalmazásakor (mivel csak ilyenekre és ezek vezérelt változatára van szükségünk, ahogy azt az 1.3.1. fejezetben láttuk).

Ha egy egy qubites kaput szeretnénk alkalmazni az i -edik qubitre egy N qubites rendszerben, ahol i és N pozitív egész számok és

$$0 \leq i < N, \quad (53)$$

akkor azt felírhatjuk úgy, hogy a 2^N -es állapotvektorunkra hat egy $2^N \times 2^N$ -es mátrix, ami N darab 2×2 -es mátrix tenzorszorzatából jön össze amik közül az i -edik a kapunk mátrixa a többi az egységmátrix. Belátható, hogy egy ilyen mátrix minden sorában maximum két nem nulla elem van, azaz az eredményünk állapotvektorának minden eleme kiszámolható az eredeti vektor két eleméből. Ennek belátáshoz vagyunk egy pozitív egész számot b -t úgy, hogy

$$i + b = N - 1 \quad (54)$$

Az i -edik qubit azt jelenti, hogy az állapotvektorunk első 2^i -edik eleme az i -edik qubit $|0\rangle$ állapotához tartozik második 2^i -edik a $|1\rangle$ -hez és innentől kezdve vektor végéig 2^i elemenként vált.

Jelöljük a két dimenziós egységmátrixot I -vel az egy qubites kapunk mátrixát x -szel és az állapotvektorunkra ható mátrixot T -vel, ekkor

$$T = I \otimes I \otimes I \otimes \dots \otimes x \otimes I \otimes I \otimes \dots \otimes I = I^{\otimes b} \otimes x \otimes I^{\otimes i} \quad (55)$$

itt az x előtt b darab I -t tenzor szorzunk össze, ezt jelöljük $I^{\otimes b}$ -vel, ez egy 2^b dimenziós egységmátrix. Hasonlóan x után i darab I -t szorzunk össze és ez lesz $I^{\otimes i}$. Az könnyen látható, hogy $x \otimes I^{\otimes i}$ -nek minden sorában maximum kettő nem nulla eleme van és ha ezt balról megtenzorszorozzuk $I^{\otimes b}$ -vel akkor az látszik, hogy ennek egy sorában pontosan annyi nem nulla eleme van mint $x \otimes I^{\otimes i}$ -nek volt ami maximum kettő.

Ez azt jelenti, hogy az eredményt össze tudjuk rakni úgy hogy az eredeti állapotvektort elemeit súlyozzuk a kapumátrixunk elemeivel, vesszük a vektorunk valamilyen permutációját, azt is súlyozzuk a mátrixunk elemeivel és ezeket összeadjuk. A súlyokat és a permutációt a következő képpen kell meghatározni.

Az egyszerűség kedvéért először nézzük meg a három qubites esetet, úgy hogy legyen a

célqubitünk (amire a kaput hattatjuk) az első qubit és legyen

$$x = \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix} \quad (56)$$

$$|\Phi\rangle = \begin{pmatrix} \Phi_{000} \\ \Phi_{001} \\ \Phi_{010} \\ \Phi_{011} \\ \Phi_{100} \\ \Phi_{101} \\ \Phi_{110} \\ \Phi_{111} \end{pmatrix} \quad (57)$$

Ahol x és $|\Phi\rangle$ minden eleme komplex szám. Mivel a célqubitünk az első így $i = 1$, ekkor a mátrix amivel az állapotvektort transzformáljuk a következőképpen néz ki.

$$\begin{aligned} T = I \otimes x \otimes I &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \\ &= \begin{pmatrix} x_{00} & 0 & x_{01} & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{00} & 0 & x_{01} & 0 & 0 & 0 & 0 \\ x_{10} & 0 & x_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{10} & 0 & x_{11} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_{00} & 0 & x_{01} & 0 \\ 0 & 0 & 0 & 0 & 0 & x_{00} & 0 & x_{01} \\ 0 & 0 & 0 & 0 & x_{10} & 0 & x_{11} & 0 \\ 0 & 0 & 0 & 0 & 0 & x_{10} & 0 & x_{11} \end{pmatrix} \end{aligned} \quad (58)$$

Hattassuk ezt a mátrixot az állapotvektorra:

$$T|\Psi\rangle = \begin{pmatrix} \Psi_{000}x_{00} + \Psi_{010}x_{01} \\ \Psi_{001}x_{00} + \Psi_{011}x_{01} \\ \Psi_{000}x_{10} + \Psi_{010}x_{11} \\ \Psi_{001}x_{10} + \Psi_{011}x_{11} \\ \Psi_{100}x_{00} + \Psi_{110}x_{01} \\ \Psi_{101}x_{00} + \Psi_{111}x_{01} \\ \Psi_{100}x_{10} + \Psi_{110}x_{11} \\ \Psi_{101}x_{10} + \Psi_{111}x_{11} \end{pmatrix} = \begin{pmatrix} \Psi_{000}x_{00} \\ \Psi_{001}x_{00} \\ \Psi_{010}x_{11} \\ \Psi_{011}x_{11} \\ \Psi_{100}x_{00} \\ \Psi_{101}x_{00} \\ \Psi_{110}x_{11} \\ \Psi_{111}x_{11} \end{pmatrix} + \begin{pmatrix} \Psi_{010}x_{01} \\ \Psi_{011}x_{01} \\ \Psi_{000}x_{10} \\ \Psi_{001}x_{10} \\ \Psi_{110}x_{01} \\ \Psi_{111}x_{01} \\ \Psi_{100}x_{10} \\ \Psi_{101}x_{10} \end{pmatrix} \quad (59)$$

Látszik, hogy a permutációt úgy végeztük, hogy felcseréltük azokat, amelyek csak az első qubitben különböznek. A súlyozást úgy végezzük, hogy a permutálatlan vektort súlyozzuk a kapumátrix diagonális elemeivel olyan módon, hogy az első qubit $|0\rangle$ -ás állapotaihoz tartozik a 00-ás elem és az $|1\rangle$ -es állapotaihoz az 11-es elem. A permutált vektort pedig az off-diagonális elemekkel úgy, hogy a $|0\rangle$ -ás állapothoz 10-ás elemek a $|1\rangle$ -eshez pedig a 01-es.

Azt, hogy ez általánosan is így van beláthatjuk abból, hogy a mátrix egy sorának két eleme közt $2^i - 1$ nulla van mivel az x mátrixot tenzorszorozzuk jobbról $I^{\otimes i}$ -vel. Az $x \otimes I^{\otimes i}$, b -szer jelenik meg a T mátrixban mivel $I^{\otimes b}$ -vel tenzorszorozzuk azt balról és így elképzelhetjük T -t úgy mint egy 2^{i+1} dimenziós mátrixokból álló mátrix ami a főátlóban mindenhol az $x \otimes I^{\otimes i}$ -t tartalmazza a többi elem nulla. Mivel a fentiekből már tudjuk, hogy T mátrix egy sorában a két nem nulla elem között $2^i - 1$ elem van és azt is tudjuk, hogy az állapotvektorban az i -edik qubit különböző állapotaihoz tartozó elemek 2^i -enként váltanak, hogy melyik állapothoz tartoznak, ez alapján a T mátrix minden sorának két nem nulla eleme az állapotvektor olyan elemével szorozódik amelyek csak az i -edik qubitben térnek el, így a permutációt úgy kell elvégeznünk, hogy azokat az elemeket amelyek csak az i -edik qubitben térnek el felcseréljük.

Mivel a $x \otimes I^{\otimes i}$ mátrixot úgy is felírhatjuk, hogy

$$x \otimes I^{\otimes i} = \begin{pmatrix} x_{00}I^{\otimes i} & x_{01}I^{\otimes i} \\ x_{10}I^{\otimes i} & x_{11}I^{\otimes i} \end{pmatrix} \quad (60)$$

belátható, hogy ez a mátrix úgy néz ki, hogy az első 2^i -edik sorában az x mátrix első sorának két eleme van $2^i - 1$ nullával elválasztva, minden sor az előzőhöz képest egyel jobbra tolva. Mivel ez a mátrix szerepel a T mátrix átlójában így már belátható, hogy a permutálatlan vektor első 2^i eleme x_{00} -val szorozódik a következő az x_{11} -gyel és innentől 2^i -ként váltakozik. A permutálatlan pedig ugyanígy csak az x_{01} és az x_{01} elemmel.

Összefoglalva az i -edik qubiten végzett műveletet felírhatjuk úgy, hogy az eredeti állapotvektor elemeit súlyozzuk a az x mátrix diagonális elemelivel attól függően, hogy az i -edik qubit $|0\rangle$ -ás vagy $|1\rangle$ -es állapotához tartozik, a permutált vektor elemeit pedig a mátrix off-diagonális elemeivel ugyanígy, majd a két kapott vektor összegéből kapjuk a transzformált vektorunkat.

3.2. Transzformáció vezérelt kapukkal

A vezérelt kapuk lényege, hogy csak akkor hattatjuk a kapunkat az állapotvektorunkra amikor a vezérlő qubit (innentől control qubit) az $|1\rangle$ állapotban van. Ez azt jelenti, hogy a vektorunknak azon elemei ahol a control qubit $|0\rangle$ állapotban van nem változnak. A művelet további része ugyanúgy viselkedik ahogy eddig. Tehát a 3.1. fejezetben látottak alapján súlyozzuk az állapotvektort és a permutált változatának elemeit, de azokban az elemekben ahol a control qubit a $|0\rangle$ állapotban van a permutálatlan vektort konstans egyel súlyozzuk a permutáltat pedig nullával.

A T mátrix ebben az esetben úgy néz ki, hogy ha az állapotvektor n -edik sora a control qubit $|0\rangle$ állapothoz tartozik, akkor a T mátrix n -edik sorának n -edik eleme egyet a többi nullát tartalmaz, a többi sor ugyanaz mint a nem vezérelt esetben volt. Ezzel elérjük azt, hogy az állapotvektor csak azon elemei transzformálódnak amelyekhez a control qubit $|1\rangle$ állapota tartozik.

A három qubites példánkhoz visszatérve ($i = 1$, x az (56). egyenlet alapján, $|\Psi\rangle$ az (57).

egyenlet alapján), legyen a control qubit a nulladik qubit, ekkor:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{00} & 0 & x_{01} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_{10} & 0 & x_{11} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x_{00} & 0 & x_{01} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & x_{10} & 0 & x_{11} \end{pmatrix} \quad (61)$$

Ezt hattatva $|\Psi\rangle$ -re:

$$T|\Psi\rangle = \begin{pmatrix} \Psi_{000} \\ \Psi_{001}x_{00} + \Psi_{011}x_{01} \\ \Psi_{010} \\ \Psi_{001}x_{10} + \Psi_{011}x_{11} \\ \Psi_{100} \\ \Psi_{101}x_{00} + \Psi_{111}x_{01} \\ \Psi_{110} \\ \Psi_{101}x_{10} + \Psi_{111}x_{11} \end{pmatrix} = \begin{pmatrix} \Psi_{000} \\ \Psi_{001}x_{00} \\ \Psi_{010} \\ \Psi_{011}x_{11} \\ \Psi_{100} \\ \Psi_{101}x_{00} \\ \Psi_{110} \\ \Psi_{111}x_{11} \end{pmatrix} + \begin{pmatrix} 0 \\ \Psi_{011}x_{01} \\ 0 \\ \Psi_{001}x_{10} \\ 0 \\ \Psi_{111}x_{01} \\ 0 \\ \Psi_{101}x_{10} \end{pmatrix} \quad (62)$$

Látható, hogy a vártak megfelelően ahol, a control qubit $|0\rangle$ ott a permutálatlan mátrix eleme egyel van súlyozva a permutálté pedig nullával.

3.3. Információtárolás a chipen

Ebben a fejezetben bemutatom a chip-en tárolt tenzorokat, és azok szerepét a szimuláció működésében.

3.3.1. Az állapotvektor eltárolása

Egy N qubites rendszer állapotvektorának eltárolásához szükségünk van

$$S_\Psi = 2^N \cdot 2 \cdot 4 = 2^{N+3} \text{ byte} \quad (63)$$

tárhelyre, mivel egy N qubites rendszer 2^N elemű vektorral reprezentálható, minden elem két számból áll: a valós és a képzetes rész, és minden számot 32 bites float-ként tárolunk ami négy byte.

A chipen a memória slice-okra, azon belül pedig address-ekre van bontva és egy address 320 byte adatot tud tárolni, mint az a 2.1. fejezetben olvasható. A valós és képzetes részeket különítsük el külön slice-okra. És tároljuk el 4 byte-os számokat úgy, hogy a négy byte egymás melletti slice-okon ugyanaz az address-en ugyanarra a helyre kerüljön. Ekkor a négy-négy slice-on egy addressen el tudjuk helyezni a vektor 320 elemét.

Nem fogjuk kihasználni mind a 320 byte-ot mivel ahogy a 3.1. fejezetben láttuk az, hogy az állapotvektor az i -edik qubit melyik állapotát tárolja az 2^i -ként vált, így érdemes lesz egy address-en csak 256 elemet tárolni, hogy egy address-en belül a nyolcadik és felette lévő qubitekhez csak egy féle állapot tartozzon, ezzel megkönnyítjük a dolgunkat a kiolvasás és a permutálás elvégzésekor.

Így ha nyolc qubit alatti qubiten akarunk transzformációt végrehajtani akkor minden address-en lévő vektoron ugyanazt a műveltet kell elvégeznünk, ha pedig nyolc vagy a fölött vagyunk akkor csak az address-eken lévő vektorok lineárkombinációra van szükség.

Az állapotvektort két bankon tároljuk, mivel van olyan eset amikor a visszaírás elkezdésekor még nem olvastuk ki az egész vektort, és ebben az esetben csak akkor tudunk egyszerre írni, ha az address amire írunk és amelyikből kiolvasunk különböző bankokon van. Az egyetlen eset amikor ez nem fog segíteni, az a nyolcadik qubit, mivel az ő egy adott állapotához tartozó elemek mind egy bank-on szerepelnek.

Mivel a transzformáció elvégzéséhez szükségünk van az eredeti vektorra és a permutáltjára így az kétszer is szerepelni fog chipen, így az állapotvektor eltárolásához

$$S_{\Psi} = 2^{N+4} \text{byte} \quad (64)$$

Mi 21 qubitig szeretnénk felgyorsítani a szimulációt, mivel ekkora rendszereken szeretnénk kvantum gépi tanulást csinálni és a GPU-s szimulációk, csak e fölött jelentenek gyorsítást[3] a CPU-hoz képest, így a mi esetünkben

$$S_{\Psi} = 2^{25} \text{byte} \quad (65)$$

és mivel egy address-en 256 byte-ot tárolunk és összesen 16 slice-on (valós és képzetes rész, kétszer mentjük le a vektort, hogy párhuzamosan tudjunk kiolvasni belőle két elemet, és 4 byte-os számokat mentünk), így egy slice-on

$$A_{\text{per slice}} = \frac{2^{25}}{256 \cdot 16} = 8196 \quad (66)$$

egy slice pont ennyi address-t tartalmaz, tehát elég 16 slice 21 qubites rendszernél is.

Nevezzük el a nyolcadik qubit alatti qubiteket S-nek (small) a nyolcadik és a tizenhetedik közöttieket L-nek (large) és tizenhetedik felettieket H-nak (high). A kapukat így két betűvel tudjuk jelölni, ahol az első arra vonatkozik, hogy a cél qubit a qubitek melyik csoportjába és második pedig, hogy a control qubit a qubitek melyik csoportjába tartozik. Az L és H qubiteket azért kell elválasztanunk, mert a hozzájuk tartozó kiolvasást és visszaírást segítő memória térkép tenzorokat különböző módon tároljuk lásd 3.3.4 fejezet.

3.3.2. Permutációs térképek

Amikor az S qubiteken végezzük a transzformációt, akkor az address-eken lévő vektorainkat mind ugyanolyan módon kell permutálnunk, ehhez meg kell adnunk, hogy ez mi alapján történik. Ezt úgy tesszük, hogy a programunkkal együtt permutációs térképeket töltünk fel

a chipre az összes cél qubit szerint. A program futtatásakor már csak a megfelelő térkép kiválasztására használandó tenzort töltjük fel, mivel ez jóval kevesebb adat mint maga a térkép, és szeretnénk minimalizálni a feltöltendő adat mennyiségét, mivel ez az egyik legidőigényesebb feladat.

Egy ilyen térkép n -edik eleme azt mondja meg, hogy az eredeti vektor melyik eleme kerüljön a permutált n -edik elemébe, ahol n egy nem negatív egész szám. Azt már a 3.1. fejezetben láttuk, hogy hogyan kell permutálnunk a mátrixot. Ehez legegyszerűbben a permutációs térképet a következő módon számíthatjuk ki az i -edig célqubitire: legyen $p_i[n]$ a permutációs térképünk n -edik eleme, ahol n, i nem negatív egész számok, ekkor

$$p_i[n] = n \wedge 2^i, \quad (67)$$

ahol $\wedge$ bitek szerinti kizáró vagyot jelenti.

Mivel 2^i bitek szerinti reprezentációban csak az i -edik helyen tartalmaz 1-et, az ezzel való $\wedge$ operátor átfordítja a n i -edik bit-jét, ami pont annak felel meg, hogy azokat az elemeket amik csak az i -edik qubit szerint térnek el felcseréljük.

Példaként nézzük meg megint a három qubites esetet $i = 1$ -gyel, ekkor

$$p_1[n] = n \wedge 2 \quad (68)$$

$$p_1 = \begin{pmatrix} 000_2 \wedge 10_2 \\ 001_2 \wedge 10_2 \\ 010_2 \wedge 10_2 \\ 011_2 \wedge 10_2 \\ 100_2 \wedge 10_2 \\ 101_2 \wedge 10_2 \\ 110_2 \wedge 10_2 \\ 111_2 \wedge 10_2 \end{pmatrix} = \begin{pmatrix} 010_2 \\ 011_2 \\ 000_2 \\ 001_2 \\ 110_2 \\ 111_2 \\ 100_2 \\ 101_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 1 \\ 6 \\ 7 \\ 4 \\ 5 \end{pmatrix} \quad (69)$$

Látható, hogy ez a várt módon viselkedik. A mi esetünkben a permutálandó vektor 320 elemből áll és abból az első 256 elem tartalmazza a számunkra fontos elemeket, tehát a permutációs térkép csak első 256 eleme fog így kinézni, a maradék elemeket nem változtatjuk, így ha $n > 255$ (mivel 0-val kezdjük a számozást nem 256-ig megyünk)

$$p_i[n] = n. \quad (70)$$

Ezeket a térképeket egy slice-on fogjuk eltárolni és megfelelő encode-olással el tudunk tárolni egy ilyen térképet egy address-en, összesen 8 ilyen térképet tárolunk (mivel csak a 8. qubit alatti qubitekre kellenek ezek).

3.3.3. State0 és state1 tenzorok

State0 tenzornak hívjuk azt a tenzort amelyik csupa 1-et tartalmaz ott ahol az állapotvektor a $|0\rangle$ állapothoz tartozik és 0-t mindenhol máshol, state1 tenzornak azt ami csupa 1-et tartalmaz ott ahol az állapotvektor a $|1\rangle$ állapothoz tartozik, 0-t mindenhol máshol.

Ezeket a tenzorokat használjuk arra, hogy kiszámoljuk, hogy az állapotvektor és permutáltjának elemeit a kapu mátrixának melyik elemeivel szorozzuk. Ezt úgy tesszük, hogy az adott kapuelemből készített vektor és a megfelelő cél vagy control qubithez tartozó `state0` vagy `state1` tenzor között bitek szerinti AND műveletet hajtunk végre.

Mivel az állapotvektor minden eleme négy slice-ra van elosztva, így a state tenzorokat is négy slice-ra osztjuk, és mivel minden kapu-hoz kell két `state0` és két `state1` tenzor (egy a cél és egy a control qubithez) így négy slice-on tároljuk, minden slice-on annyi address-en ahány kapu-t alkalmazunk a szimuláció során.

3.3.4. Mem gather és scatter map-ok

Mem gather map-nek hívjuk azokat a térképeket amik segítségével megfelelő sorrendben kiolvassuk az állapotvektort, mivel amikor a qubit az L vagy H qubitek közé tartozik, akkor nem kell permutálnunk a mátrixot, hanem elég a megfelelő sorrendben kiolvasni az elemeket. A mem scatter map-ok segítségével pedig visszaírjuk az állapotvektort a megfelelő helyre.

Mivel ezekben az esetekben az egyes address-ekre kerülő érték két address-en lévő elem lineárkombinációjából jön létre, így előfordulhat olyan eset is amikor ha az eredeti sorrendben olvasnánk ki az állapotvektort, akkor az egyes address-ek értéke felülíródna mielőtt kiolvasnánk őket. Ezt elkerülve a mem gather és scatter map-okat úgy kell létrehozunk, hogy ez ne történjen meg.

A vezérelt kapuk esetében, ha a control qubit az L vagy H qubitek közé tartozik a mem gather és scatter map-ok azért is felelősek, hogy csak a control qubit $|1\rangle$ állapotához tartozó elemek kerüljenek kiolvasásra és visszaírásra.

Az L és H kapuk esetén a mem gather és scatter map-okat külön slice-okon tároljuk. Vezérelt kapuk esetén adott cél qubit csoporthoz (S, L vagy H) tartozó mem gather map-okat egy slice-on tároljuk, mivel ezek fele annyi helyet foglalnak mint a nem vezérelt (mivel csak a fél állapotvektort kell kiolvasnunk, így fele annyi address-t tartalmaznak), úgy osztjuk fel őket, hogy a slice-on lévő address-ek első felét az L és második felét H a control qubithez rendeljük.

3.4. A szimuláció futásának menete

A szimuláció futtatásának első lépése a statikus adatok létrehozása és a chipre való feltöltése, ebbe a kategóriába tartozik az S qubitek esetében az állapotvektor permutálására használt permutációs térképek, az L és H qubitek esetén használt mem gather és scatter map-ok a memória helyes sorrendbe való kiolvasásához, az egyes qubitekhez tartozó `state0` és `state1` tenzorok és a distributor map-ok.

A szimulációhoz fel kell töltenünk az egyes kapuk mátrixát és az ehez tartozó cél- és control qubitekhez tartozó tenzorokat, amik azért felelősek, hogy a permutációs térképek, a mem gather és scatter map-ok és a `state0` és `state1` tenzorok közül kiválasszák a megfelelőeket, ezeket hívjuk selector tenzoroknak. Ahhoz, hogy minden kapuhoz ki tudjuk olvasni a megfelelő selectort még szükségünk lesz ezekhez tartozó számláló tenzorokra is amik azt az

információt tárolják, hogy hanyadik kapunál tartunk, így azokat minden kapuművelet után megnövelünk.

S kapu

Abban az esetben, ha nem vezérelt kapunk van és a célqubit az S qubitek közül való, akkor kiolvassuk a teljes állapotvektort és a számláló tenzorunk segítségével kiolvassuk a kapuhoz tartozó permutációs térkép selector tenzort, majd ennek segítségével kiolvassuk a megfelelő permutációs térképet. Az állapotvektor valós részét az egyik oldali a képzetes részét a másik oldali permutor segítségével a kiolvasott permutációs térképek szerint permutáljuk, majd a permutált vektort elmentjük a chip-en. Ezután kiolvassuk a megfelelő state0 és state1 tenzorokat a megfelelő számláló tenzorral kiválasztott state0 és state1 selectorainkal. Kiolvassuk a kapuelemeket és úgy, hogy a 3.1. fejezetben látottak szerint lehessen súlyozni velük az állapotvektort, bitek szerinti AND műveletet hajtunk végre a kapuelemek és a state0 és state1 tenzorok között. Ezek után a diagonális elemeket megszorozzuk a permutálatlan állapotvektorral az off-diagonálisakat pedig a permutáltal. Végül lementjük a transzformált állapotvektort az eredeti és a permutált helyére is.

SS kapu

Abban az esetben amikor a cél és control qubit is az S qubitek közül való, az előzőhöz képest annyi a különbség, hogy ki kell olvasnunk a control qubitnek megfelelő state0 és state1 selectort majd annak segítségével a state0 és state1 tenzorokat, majd a kapu diagonális és off-diagonális elemeire egyaránt bitek szerinti AND operációt hajtunk végre a state1 tenzorral. Majd a state0 tenzoron bitek szerinti AND operációt hajtunk végre egy csupa 1 tenzorral és ezt hozzáadjuk a diagonális elemekhez. Ezután az előzőhöz hasonlóan végezzük a transzformációt, és mentjük el a kapott állapotvektort.

S8 kapu

Amikor a cél qubit az S qubitek közül való és control qubit a nyolcadik, akkor az első verzióknak megfelelően alkalmazzuk a transzformációt, de csak minden második address-t írjuk vissza a helyére. Mivel minden második address a nyolcadik qubit másik állapotához tartozik, és mivel ezek mind egy bankon vannak, így hiába olvasnánk ki csak a fél vektort ugyanannyi időbe telik a transzformáció.

SL és SH kapu

Ha a cél qubit az S qubitek a control pedig a H, vagy L qubitek közül való akkor elég kiolvasnunk a megfelelő address-eket a permutáláshoz, ehhez kiolvassuk a control qubithez tartozó mem gather map-ot, aminek segítségével kiolvassuk azokat az address-eket ahol a control qubit az $|1\rangle$ -es állapotban van, és így az állapotvektorunknak csak a felét kell átfolyatnunk a permutoron. A control qubit=8 esetben ezt azért nem tudtuk megtenni, mivel minden address ugyanarra a bank-ra esett, így ott nem tudtuk volna gyorsabban

visszaírni az állapotvektort. Ezek után a transzformációhoz való kiolvasáshoz is mem gather map-ot használunk a permutált és permutálatlan állapotvektorra is. Majd ha elvégeztük a transzformációt akkor mem scatter map-ok segítségével visszaírjuk az állapotvektort a helyére.

8-as kapu

Amikor a a nyolcadik qubitre hattatjuk a kaput, mivel nem ugyanazt a műveletet alkalmazzuk minden address-re, hanem különböző address-eken lévő elemek lieárkombinációját írjuk vissza az address-ekre, így nem mindegy, hogy milyen sorrendbe olvassuk ki az adatokat. Ennek az az oka, hogy előfordulhat olyan helyzet, hogy olyan helyről olvasunk ki amit már felülírtunk a transzformáció során, ezt elkerülendő használjuk a mem gather map-okat a H és L qubitek esetében, mivel ez a gate külön szerepel nem kell mem gather map-ot használnunk, hanem expliciten megmondjuk, hogy milyen sorrendbe kell kiolvasni az állapotvektor elemeit (ezt az esetet azért választjuk el az L és H kapuktól, mivel amint azt 3.3. fejezetben említettem az állapotvektor a nyolcadik qubit egy adott állapotához tartozó elemei mind egy bank-ra esnek). Elvégezzük a kapuműveletet, majd visszamentjük az állapotvektort mindkét helyre a mem scatter map-ok segítségével.

8S kapu

Itt a 8-as kapuhoz hasonlóan kell kiolvasnunk az állapotvektor elemeit, a kapuk elemeit elő kell készítenünk, úgy ahogy azt az SS kapuknál tettük.

8L és 8H kapu

Itt az állapotvektor elemeinek kiolvasásához a control qubitnek megfelelő mem gather map-ot kell használnunk, ami segítségével megfelelő sorrendbe kiolvassuk a control qubit $|1\rangle$ állapotához tartozó elemeket. Majd a control qubitnek megfelelő mem scatter map-pal lementjük a transzformált állapotvektort.

L és H kapu

Itt a 8-as kapunál leírt indokból az állapotvektor kiolvasásához mem gather map-okat használunk, melyek kiolvasásához szükségünk lesz a hozzá tartozó számlálóra. Ezután elvégezzük a kapuműveletet, majd visszaírjuk a vektort a két helyre a számláló segítségével kiválasztott mem scatter map-pal.

LS és HS kapu

Az állapotvektor kiolvasása itt ugyanúgy végződik, mint az L és H kapuk esetében, viszont a kapuelemeket elő kell készítenünk úgy ahogyan azt az SS és 8S kapuk esetében tesszük azaz a diagonális és off-diagonális elemeket kinullázzuk ott ahol a control qubithez tartozó

state1 nulla és a diagonálshoz hozzáadunk egyet ott ahol a control qubithez tartozó state0 egy.

L8 és H8 kapu

Ezekben az esetekben az S8 kapuhoz hasonlóan csak minden második address-en lévő elemeken végezzük a transzformációt, viszont mivel ezek ugyanazon a bankon vannak nem tudjuk gyorsabban visszaírni, az állapotvektor elemeinek kiolvasásának és visszaírásának sorrendje az L és H kapukhoz hasonlóan a mem gather mapok segítségével történik.

LL, LH, HL és HH kapuk

Ezen kapuk esetében az L és H kapukhoz hasonlóan a megfelelő sorrendben kell kiolvasnunk az állapotvektor elemeit, és az SH és SL kapukhoz hasonlóan szintén csak a control qubitnek $|1\rangle$ állapotának megfelelő elemeken végezzük a transzformációt.

4. Számítási teljesítmény

A dolgozat írásakor az S, 8, L és a hozzájuk tartozó vezérelt kapuk vannak készen, és egymás után húsz kaput tudunk felfűzni.

A groq chip-en lévő futási időt fel tudjuk bontani különböző blokkokra, és tudjuk mérni ezek futási idejét.

Blokk	t[s]
Program feltöltése	0.067
Bemenet felkészítése	0.00009
Kapu adatok feltöltése	0.0023
Eredeti állapotvektor feltöltése	0.0014
kapu operáció	0.00012
Állapotvektor letöltése	0.0014
Teljes futási idő	0.076

1. táblázat. A különböző blokkok futási ideje 17 qubit, 20 kapu esetén

A program feltöltése annak az idejét jelzi, amennyi időt vesz igénybe feltölteni a chip gépi kódjára lefordított programot, a statikus adatokkal együtt. A bemenet felkészítése a selector és distributor mapok felkészítését jelenti. A kapu adatok feltöltése a kapu mátrixának a chip-nek megfelelő formátumba való előkészítése és a chipre való feltöltését jelenti.

Látható, hogy a futási idő legnagyobb részét a program feltöltése teszi ki, amit csak egyszer kell megtennünk gyakorlati használatban. Ezután a sok időt vesz igénybe a kapuk áttranszformálása és feltöltése, az állapotvektor letöltése és az eredeti állapotvektor feltöltése. A kapukat egy hosszab áramkör esetében szükséges lesz több adagban feltölteni, viszont az állapotvektor fel és letöltése egyszer elvégzendő művelet.

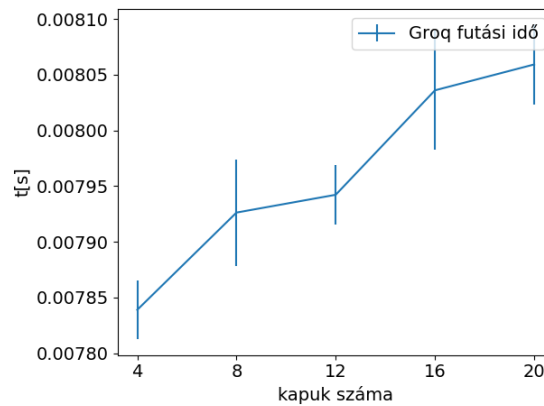
A szimuláció ötszöri véletlenszerű eredeti állapotokkal és kapukkal való lefuttatása után hasonlítsuk össze egy AMD EPYC 7542 32-Core Processor-on futó 1.8.5-ös squander szimulációval, a program feltöltési idejét nem figyelembe véve:

	CPU	Groq (program feltöltési ideje nélkül)	Program feltöltési ideje
t[s]	0.055 ± 0.005	0.00806 ± 0.00004	0.067 ± 0.001

2. táblázat. Futási idő összehasonlítása a CPU-s szimulációval 17 qubit, 20 kapu esetén

17 qubit és 20 kapu esetén nagyjából hétszeres gyorsítást kaptunk.

Most nézzük meg, hogy hogyan nő a program futási ideje az elvégzendő kapuk függvényében. Láttuk, hogy az adatok feltöltése tart a legtöbb ideig, tehát arra számítunk, hogy a kapuk növelésével nem fog szignifikánsan nőni a szimuláció hossza a CPU-hoz képest.



10. ábra. Groq szimuláció futási ideje a program feltöltése nélkül

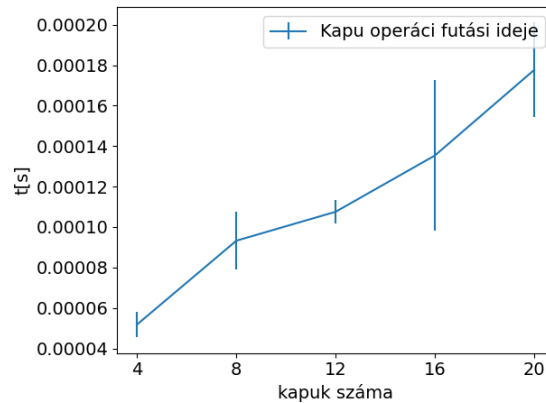
A 10. ábra alapján láthatjuk a szimulációnk futási idejének függését a kapuk számától függően, a 11. ábrán láthatjuk csak a kapu operáció futási idejét.

A 12. ábra alapján megállapíthatjuk, hogy a CPU futási ideje 4 qubit felett már magasabb lesz mint a Groq-s implementációé.

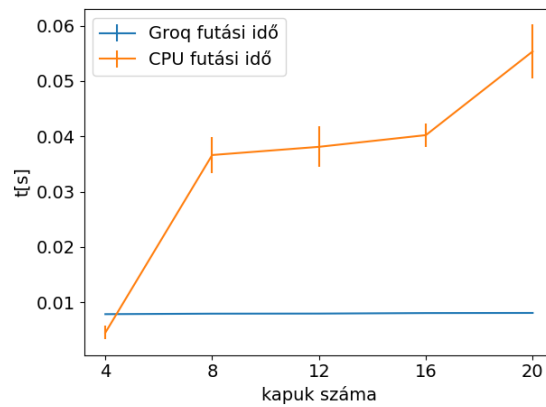
A 13. ábrán a CPU futási idejét és a Groq LPU-n végzett kapu operációk idejét elosztottuk azzal az idővel, amennyi időbe telik lefuttatniuk a 4 kapuból álló áramkört. Ebből azt tudjuk megállapítani, hogy a kapuk növekedésével hányszorosára növekedik a futási idő. Azt láthatjuk, hogy az LPU-s megoldásunk nem csak hogy kevesebb időt vesz igénybe, ahogy azt a 12. ábrán láttuk, de a kapuk növelésével kisebb ütemben nő a futási idő.

5. További fejlesztési lehetőségek

A további fejlesztés első lépése a qubitek számának 17-ről 21-re való emelése, és húsznál több kapu felfűzése. Ezek után még tovább optimalizálhatjuk a szimulációt az alább bemutatott módokon.



11. ábra. Csak a kapu operáció futási ideje

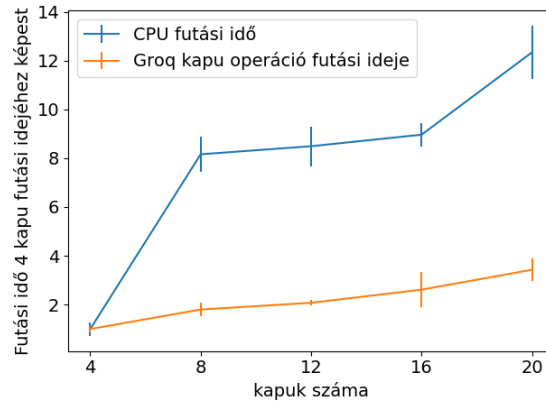


12. ábra. A futási idők összehasonlítása

Amint azt a 3.3 fejezetben említettem a transzformáció során használt tenzorok nagy része a CPU-n van előkészítve és ez nem csak több időt vesz igénybe, de több adatot is kell a chipre feltöltenünk. Ezeknek a chip-re való áthelyezésével csökkenthetnénk a bemenet felkészítésével töltött időt.

Az 1. táblázatból látható, hogy a kapu mátrixának a chipre való előkészítése az egyik leg időigényesebb feladat, ezt tudjuk rövidíteni úgy, hogy ezt nem a CPU-n végezzük, hanem a chip-re való feltöltés után annak segítségével alakítjuk a chip-való transzformációkhoz megfelelő formátumúra.

A variációs kvantum algoritmusok használatakor sok esetben az algoritmus kezdetekor a qubitek mind a $|0\rangle$ állapotban vannak, ennek a figyelembe vételével a kezdeti állapotvektor feltöltési idejét le lehet csökkenteni.



13. ábra. A futási idő aránya 4 kapu futási idejéhez képest

6. Összegzés

A dolgozat a Groq chipen futó kvantumszimuláció megvalósításával foglalkozik, amely a kvantumalgoritmusok, különösen a Variációs Kvantum Algoritmusok (VQA) gyorsítását célozza. A szimuláció célja, hogy a hagyományos CPU és GPU megoldásokhoz képest, a 22 qubit alatti kvantumrendszereket hatékonyan szimulálja. Ehhez a Groq nyelvi feldolgozóegységeit (LPU) használjuk, amelyek segítségével az előzetes becsléseink alapján felgyorsíthatjuk a 22 qubit alatti szimulációt.

A chip memóriáját address-ekre bontjuk és egy address-en el tudjuk tárolni az állapotvektor 256 elemét. Egy ilyen address-en lévő elemeket egyszerre tudjuk kiolvasni, és párhuzamosan tudunk rajtuk műveleteket végezni. A szimulációban a chip memóriájának felépítéséből adódóan fontos szerepet játszik a qubitek hierarchikus felosztása, amelyeket S(small), L(large) és H(high) qubitscsoportokba osztunk. Az S csoport az első 8 qubitet tartalmazza, ahol minden address-en lévő vektort ugyanolyan módon kell transzfromálnunk. A qubitek számozását a nullától kezdve a 9-17. qubitek alkotják az L csoportot, ahol a transzfromált állapotvektor elemeit az address-ek lineárkombinációjából számoljuk ki. Itt ugyanakkor figyelniünk kell arra, hogy a kivolasás előtt ne írjuk felül az egyes address-ek értékeit. A H csoport a 18. qubittól kezdődik, ezeket azért különítjük el az L csoporttól, mert az állapotvektor kiolvasásának sorrendjét a memória felépítése miatt máshogy kell végeznünk. A 8. qubit transzfromációja eltér a többitől, ennek az oka az, hogy az address-eket két bank-ra csoportosítjuk, az állapotvektor a 8. qubit adott állapotához tartozó elemei az L és H qubitekkel ellentétben ugyanazon a bank-on vannak, és ennek következtében más sorrendben kell kiolvasnunk az elemeket. A bank-ok szerepe az, hogy ha két address különböző bank-on van akkor ezeket egyszerre tudjuk írni és olvasni. Vezérelt kapuk esetén ugyanezekből az okokból a control qubiteket is ilyen csoportokba kell bontanunk. S control qubitek esetében csak az egyes address-eken végzett művelet változik, viszont a 8-as, az L és a H control qubitek esetében elég kiolvasnunk és transzfromálnunk az állapotvektor a

control qubit megfelelő állapotához tartozó elemeit.

Az implementáció közben a kihívást többek közt az jelentette, hogy minden qubit és control qubit csoporthoz a náluk felmerülő egyedi kihívások miatt külön programot kellett elkészítenünk. Az S qubiteken a szükséges permutációkat, az L és H csoportok esetében a kiolvasás sorrendjét, a kapuk elmeinek előkészítését mind egyedi térkép tenzorokkal kellett megvalósítani, hogy a műveletek megfelelő módon történjenek. További kihívást jelentett az adatáramlás optimalizálása, mivel a jobb teljesítmény eléréséhez a számítási műveleteket szorosan össze kell hangolnunk.

A számítási teljesítmény mérésénél a Groq chip futási idejét blokkokra bontva vizsgáltuk, és ezek futási idejét egyenként mértük. A szimuláció során a legnagyobb időigényt a program chipre történő feltöltése és az állapotvektor kezelése jelentette, amelyet azonban csak egyszer szükséges elvégezni. A kapuk adatainak feltöltése szintén jelentős időt igényel, de ezeket csak hosszabb áramkörök esetében kell többször feltölteni. Az eredményeket összevetve egy AMD EPYC 7542 processzorral futtatott szimulációval, a 17 qubit és 20 kapu esetén a program a chipre-való feltöltését nem figyelembe véve a Groq chip hétszeres gyorsítást ért el. A 20 kapu alatti méréseink azt mutatják, hogy nem csak gyorsabb lett, de a kapuk növelésével a CPU-hoz képest jobban is skálázódik a szimulációnk.

Hivatkozások

- [1] Dennis Abts és tsai. “Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads”. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 2020, 145–158. old. DOI: 10.1109/ISCA45697.2020.00023.
- [2] Adriano Barenco és tsai. “Elementary gates for quantum computation”. *Physical Review A* 52.5 (1995. nov.), 3457–3467. old. ISSN: 1094-1622. DOI: 10.1103/physreva.52.3457. URL: <http://dx.doi.org/10.1103/PhysRevA.52.3457>.
- [3] “Choosing hardware for your qsim simulation”. (). URL: https://quantumai.google/qsim/choose_hw.
- [4] Andrew Steane. “Quantum computing”. *Reports on Progress in Physics* 61.2 (1998. febr.), 117–173. old. ISSN: 1361-6633. DOI: 10.1088/0034-4885/61/2/002. URL: <http://dx.doi.org/10.1088/0034-4885/61/2/002>.
- [5] R.S. Sutor. *Dancing with Qubits: How Quantum Computing Works and how it Can Change the World*. Expert Insight. Packt Publishing, 2019. ISBN: 9781838827366. URL: <https://books.google.hu/books?id=uwAeywEACAAJ>.