

Memóriafelhasználás optimalizálása

kvantumalgoritmusok szimulációja során

Nemkin Viktória

dr. Friedl Katalin
Számítástudományi és Információelméleti Tanszék



Motiváció: "Protein folding" megoldása kvantumalgoritmussal

- **Protein:**

- ▶ Aminosavakból alkotott lánc.
 - ★ **Piros = Hidrofób ("vízkerülő").**
 - ★ **Kék = Poláris ("vízszerető").**
- ▶ Hajtogatás: 3D kockarács pontjain.
- ▶ Cél: Minimális energiájú elhelyezés.

- **Kódolás:**

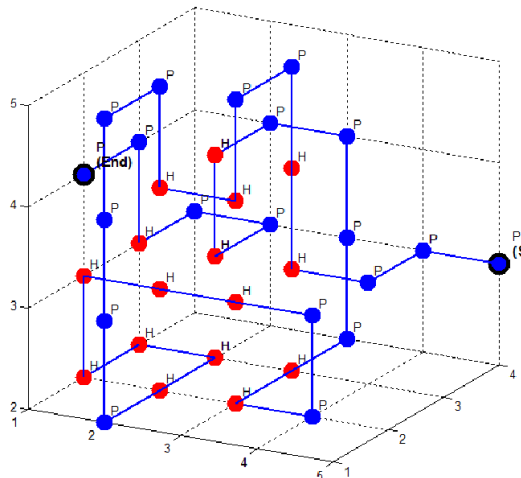
- ▶ Origóból, lépésenként
6 irány = 6 kvantumbit (qubit).

- **Orákulum:**

- ▶ Energiaviszonyok lepontozása.

- **Grover keresés:** $O(\sqrt{N})$

- ▶ "Kvantum párhuzamosság"-ot kihasználva.
- ▶ Energiaminimum megtalálása.



Egy összehajtogatott protein.¹

¹Forrás: Traykov et al. (2018). Protein Folding in 3D Lattice HP Model Using Heuristic Algorithm.

Feladat

- **Cél:**
 - ▶ Kvantumalgoritmusok kipróbálása és elemzése.
- **Eszköztár:**
 - ▶ Regiszterek.
 - ▶ Operátorok: Hadamard, Grover, Sum, MC-NOT.
 - ★ "Fekete doboz"-ként szimulálni.
- **Probléma:**
 - ▶ Kvantumszámítógép: Publikusan nem elérhető (elég nagy).
 - ▶ Klasszikus szimuláció: Túl sok memóriát fogyaszt.

Probléma: Memóriahasználat

Lánc	Qubitek	Regiszter	Operátor
n	$6(n-1)$	$2^{6(n-1)} \cdot 16B$	$2^{12(n-1)} \cdot 16B$

Probléma: Memóriahasználat

Lánc	Qubitek	Regiszter	Operátor
n	$6(n-1)$	$2^{6(n-1)} \cdot 16B$	$2^{12(n-1)} \cdot 16B$
2	6	1 KB	64 KB
3	12	64 KB	256 MB
4	18	4 MB	1 TB
5	24	256 MB	4 PB
6	30	16 GB	16384 PB
7	36	1 TB	67108864 PB

Probléma: Memóriahasználat

Lánc	Qubitek	Regiszter	Operátor
n	$6(n-1)$	$2^{6(n-1)} \cdot 16B$	$2^{12(n-1)} \cdot 16B$
2	6	1 KB	64 KB
3	12	64 KB	256 MB
4	18	4 MB	1 TB
5	24	256 MB	4 PB
6	30	16 GB	16384 PB
7	36	1 TB	67108864 PB

- Optimalizációk (regiszter és operátor esetében is):

- ▶ Ritka mátrixos tárolás:
 - ★ IBM Qiskit, Google Cirq, stb.
 - ★ Nem mindig ritkák a mátrixok.
- ▶ Döntési fa alapú adatszerkezet:
 - ★ Újabb kutatási irány.
 - ★ Nem mindig segít.

Megoldás: "On-the-fly" és "Függvény-alapú" operátorok

Lánc	Qubitek	Regiszter	Operátor	"On-the-fly"	"Függvény-alapú"
n	$6(n-1)$	$2^{6(n-1)} \cdot 16B$	$2^{12(n-1)} \cdot 16B$	= Regiszter.	Nem kell tárolni.
2	6	1 KB	64 KB	1 KB	0 B
3	12	64 KB	256 MB	64 KB	0 B
4	18	4 MB	1 TB	4 MB	0 B
5	24	256 MB	4 PB	256 MB	0 B
6	30	16 GB	16384 PB	16 GB	0 B
7	36	1 TB	67108864 PB	1 TB	0 B

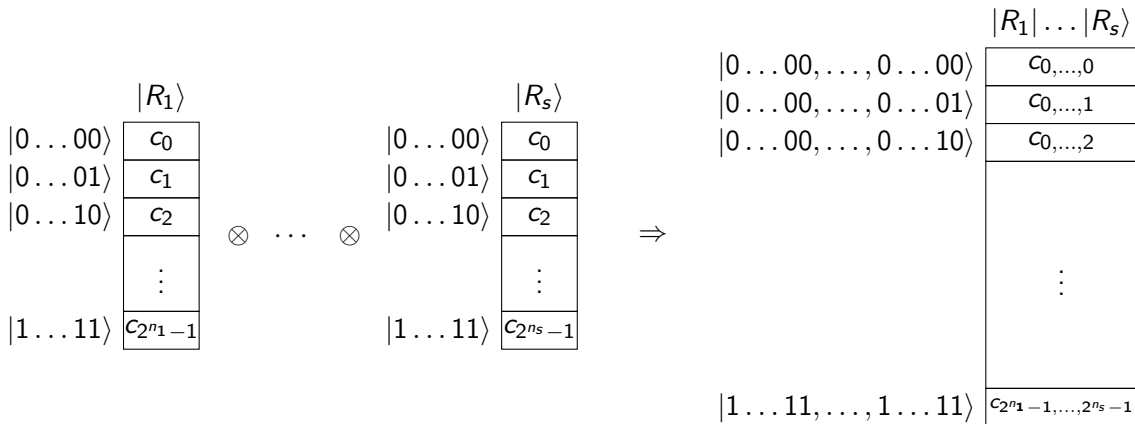
Megoldás: "On-the-fly" és "Függvény-alapú" operátorok

Lánc	Qubitek	Regiszter	Operátor	"On-the-fly"	"Függvény-alapú"
n	$6(n-1)$	$2^{6(n-1)} \cdot 16B$	$2^{12(n-1)} \cdot 16B$	= Regiszter.	Nem kell tárolni.
2	6	1 KB	64 KB	1 KB	0 B
3	12	64 KB	256 MB	64 KB	0 B
4	18	4 MB	1 TB	4 MB	0 B
5	24	256 MB	4 PB	256 MB	0 B
6	30	16 GB	16384 PB	16 GB	0 B
7	36	1 TB	67108864 PB	1 TB	0 B

- Qiskit, stb. nyílt forráskódúak...
- ...de szerves része a kódnak az operátor tárolása
- → saját szimulátor implementáció.

Regiszterek állapotának tárolása

- Minden bitsorozathoz egy komplex valószínűségi amplitúdó.
- Szuperpozíció és összefonódás \rightarrow indexek Descartes-szorzata.
- Méretek: $2^{n_1}, \dots, 2^{n_s} \rightarrow \prod_{i=1}^s 2^{n_i}$.



Regiszterkezelés nehézségei

Példa: $A_{4 \times 4}$ (2 qubites) operátor.

Utolsó 2 qubit-re:

$$|\square \dots \square \blacksquare \blacksquare\rangle$$



$$\begin{pmatrix} A & 0 & \dots & 0 \\ 0 & A & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A \end{pmatrix}$$



Első 2 qubit-re:

$$|\blacksquare \blacksquare \square \dots \square\rangle$$



$$\begin{pmatrix} A_{0,0}/ & A_{0,1}/ & A_{0,2}/ & A_{0,3}/ \\ A_{1,0}/ & A_{1,1}/ & A_{1,2}/ & A_{1,3}/ \\ A_{2,0}/ & A_{2,1}/ & A_{2,2}/ & A_{2,3}/ \\ A_{3,0}/ & A_{3,1}/ & A_{3,2}/ & A_{3,3}/ \end{pmatrix}$$



Köztes esetek?

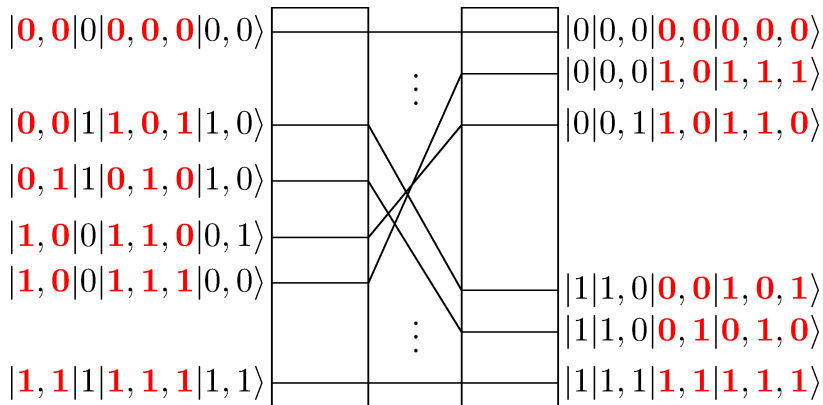
$$|\square \dots \square \blacksquare \square \dots \square \blacksquare \square \dots \square\rangle$$

Qubit és index leképezés

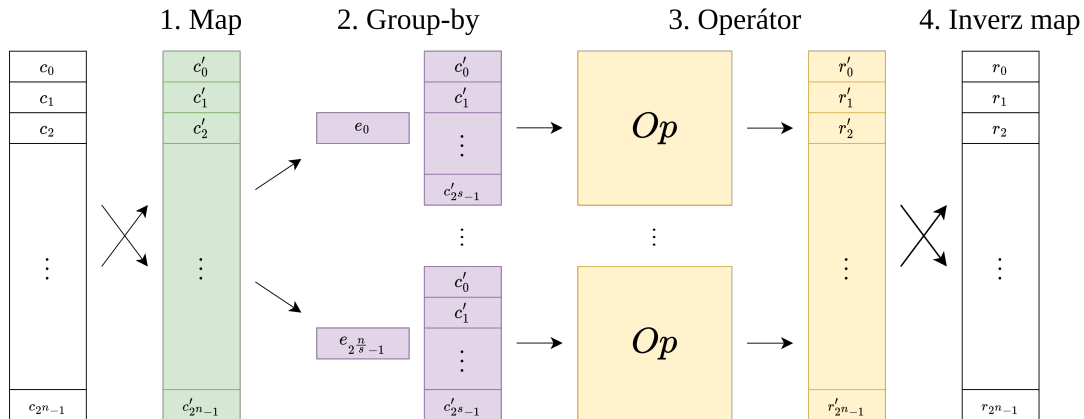
Qubit map

$$|q_0, q_1|q_2|q_3, q_4, q_5|q_6, q_7\rangle \Rightarrow |q_2|q_6, q_7|q_0, q_1|q_3, q_4, q_5\rangle$$

Index map



Operátor végrehajtás



Operátorok megvalósítása

Megvalósítás:

- Inverzió (Visitor minta):
 - ▶ Operátornak odaadom a regisztert, végrehajtja magát rajta.
 - ▶ Belső működés eltakarva \rightarrow mátrix reprezentáció nem szükséges \rightarrow memória spórolás.
- Operátor típusok:
 - ▶ "On-the-fly" operátorok:
 - ★ A mátrix egy sorát generálok \rightarrow skalárszorzás.
 - ★ Hadamard, Grover.
 - ▶ "Függvény-alapú" operátorok:
 - ★ $u : |0 \dots 0, \text{in}\rangle \rightarrow |\text{out}, \text{in}\rangle$
 - ★ Sum: $\sum : |0 \dots 0, \text{in}\rangle \rightarrow |\text{count}(\text{in}), \text{in}\rangle$
 - ★ MC-NOT: $\text{mcnot} : |0, \text{in}\rangle \rightarrow |\text{any}(\text{in}), \text{in}\rangle$
- Strategy minta:
 - ▶ Egységes algoritmus interfész: több lehetséges implementáció.
- Operátor csak egy "handle", példány nem foglal memóriát.

Eredmények:

- Regiszterek:
 - ▶ Ritka mátrixos tárolás.
 - ▶ Tetszőleges célregiszterek: qubit és index leképzés.
- Operátorok:
 - ▶ "On-the-fly": Hadamard, Grover.
 - ▶ "Függvény-alapú": Sum, MC-NOT.
- Könnyen bővíthető architektúra.
- Forráskód: <https://github.com/nemkin/qmem> (MIT licensz).

Jövőbeli tervek:

- Protein folding algoritmus implementálása.
- Döntési fa alapú regisztertárolás kipróbálása.

Bíráló kérdései - 1.

A dolgozat bevezetésében írja, hogy az egyik motivációt a munkához a bioinformatika adja, ezen belül is a fehérje feltekeredés (protein folding) vizsgálata.

A dolgozatban azonban - érthető okokból - egy egyszerűbb problémával, a Sudoku általános változatával dolgozik.

Mégis, meg tudná mondani, hogyan lehetne (milyen jellegű továbbfejlesztésekkel) a kidolgozott eljárást a bioinformatikában használni?

Bíráló kérdései - 2.

Mi a szerző várakozása a kifejlesztett keretrendszer működési korlátait illetően?

Például:

Hány qubites Grover-keresést fog tudni implementálni?

- 30 qubit körüli.

Mekkora n esetén tudja implementálni a Sudoku verifiert?

- 3×3 -mas Sudoku táblát lehet (27 qubit) \Leftrightarrow 4×4 -es táblát már nem (64 qubit).

A maximális méretű problémát implementáló kód mennyi idő alatt fog lefutni egy laptopon?

- A tároláshoz 27 qubitre + 1 orákulum kimeneti qubitre van szükség.
- Az orákulum implementálása "Függvény-alapú" operátorral megoldható.
- Becsléseim szerint egy-két napon belül lefutna.
- Qiskit-ben ezt elkészítettem, ott a 3×3 -mas Sudoku táblához 67 TB memóriára lenne szükség.

Bíráló kérdései - 3.

A keretrendszer struktúrájában vagy implementációjában meg kell-e különböztetni a CPU-n és a GPU-n való futtatás esetét?

- "On-the-fly" típusú operátorok könnyen implementálhatóak GPU-n, a meglévő keretrendszerben.
- "Függvény-alapú" operátorok nem vihetők át GPU-ra (nincs skalárszorzás) → sokszálú CPU-val párhuzamosíthatók.

Bíráló kérdései - 4.

Valódi kvantumszámítógépekben a kvantumbitek és a környezet kölcsönhatása dekoherenciához vezet.

A Grover-keresés működőképes marad-e, ha a dekoherenciát figyelembe vesszük a szimuláció során?

- A dekoherencia csökkenti a találati valószínűséget.
- Amíg ez nem lép túl egy korlátot, addig többszöri ismételt futtatással javítható.

Lehetséges-e a dolgozatban kifejtett hatékony keretrendszert általánosítani dekoherencia jelenléte esetére, és ha igen, akkor meg lehet-e ezt tenni úgy hogy a hatékonyság is megmarad?

- A dekoherencia modellezhető bizonyos valószínűséggel alkalmazott mátrixműveletekkel, ezek operátorként megvalósíthatók ebben a rendszerben.