Kvantumszámítógép szimulációja GPU használatával

Dobó Imre
dobo.imre@gmail.com

Konzulens: Dr. Imre Sándor, BME Híradástechnikai Tanszék
imre@hit.bme.hu

Tartalomjegyzék

1. Bevezetés

A számítástechnika fejlődése az elmúlt időkben egyre dinamikusabbá és szerteágazóbbá vált. Rengeteg új technológia, új megoldás született. Nagyon ígéretes és egyre dinamikusabban fejlődő terület a kvantuminformatika világa.

A kvantumszámítógépek olyan eszközök, amelyek a kvantummechanikai jelenségeket használják a számítások elvégzéséhez. Az új paradigma jelentős fejlődést rejt magában, hiszen kvantumszámítógépen sok algoritmus jóval gyorsabban lefuttatható, mint a ma használatos eszközökön. Sajnos azonban ma még, bizonyos fizikai problémák miatt nem lehetséges az új elv alapján működő eszközök létrehozása.

Dolgozatomban arra a kérdésre keresem a választ, hogy vajon lehetséges-e asztali számítógépen célhardver használata nélkül szimulálni kvantumalgoritmusokat. A kutatás során arra jutottam, hogy megvalósítható 10-12 bites rendszerek modellezése a ma elérhető hardverarchitektúrákon. Fontos eredmény tovább, hogy a szimuláció futása jelentősen meggyorsítható GPU használatával.

A dolgozat első fejezetében a motivációkról írok. Röviden bemutatom, hogy mi inspirálta a kvantuminformatika létrejöttét, mik az eddig elért eredmények és mik az aktuális kihívások. Szó lesz arról is, hogy miért érdemes kvantumrendszerek szimulációjával foglalkozni illetve hogy milyen problémák merülhetnek fel a megvalósítás során.

Ezt követően bevezetem a kvantuminformatika alapfogalmait (például qbit, mérés, stb.) és alapjelenségeit (mint a szuperpozíció elve vagy az összefonódás). Megismerkedünk egy matematikai leírással, amely jól használhatónak bizonyul az implementáció során.

A következő fejezetben ismertetem az eredményeket, amelyekből kiderül a CPU és GPU közötti sebességkülönbség és az, hogy a két architektúra közötti eltérés hogyan befolyásolja a szimuláció teljesítményét.

A dolgozat zárásaként a szimuláció továbbfejleszthetőségéről írok. Választ keresek arra a kérdésre, hogy hogyan lehetne növelni a teljesítményt, milyen más architektúrákon érdemes a vizsgálódást folytatni és milyen egyéb lehetőség rejlik egy kvantumszámítógép szimulációban.

2. Motivációk

Az alábbi bekezdésben a kvantuminformatika motivációiról lesz szó. A Moore-törvény végének közeledtével egyre aktuálisabbá válnak az új hardvermegoldások. A motivációk áttekintése után a kvantumszámítógép rövid története illetve néhány elért eredmény ismertetése kerül sorra. A fejezet második felében a szimuláció motivációit vázolom fel.

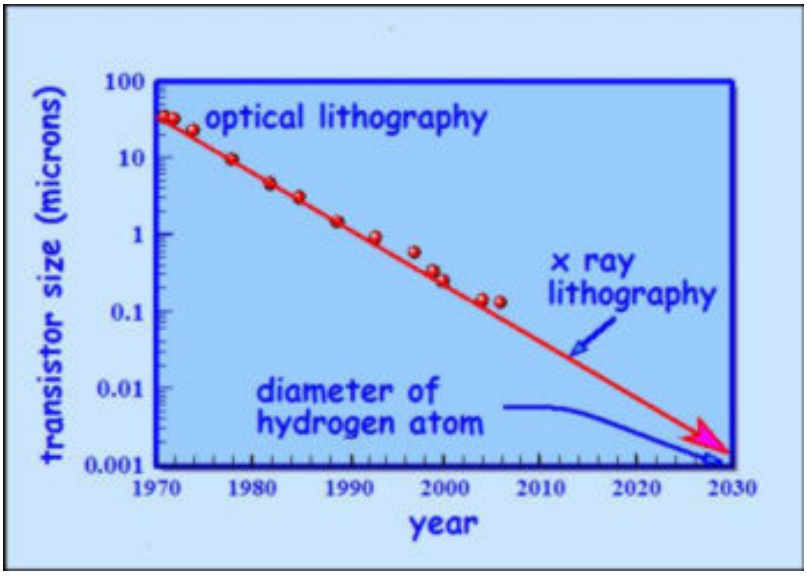
2.1. Mi a kvantumszámítógép?

a) A Moore-törvény és annak vége

A korai – még elektroncsöves – számítógépek nagyméretűek és megbízhatatlanok voltak. A félvezetők megjelenésével azonban jelentős méretcsökkenés és teljesítménynövekedés ment végbe. Gordon Moore, az Intel alapítója az 1960-as évek közepén tette az azóta Moore-törvényként ismerté vált jóslatát, amely szerint egységnyi méretű integrált áramkörbe építhető tranzisztorok száma körülbelül kétévente megduplázódik. [1]

A megfigyelés a mai napig helytállónak bizonyult: a hardverfejlődés a Moore által közel ötven éve megjósolt ütemet követi. [1] Nyilvánvaló azonban, hogy a „törvény” véges, amelynek két fő oka van. Az egyik ok az, hogy a tranzisztor nem valósítható meg tetszőlegesen kis méretben, hiszen az egyes részeknek (emitterek, kollektorok) legalább egy atom nagyságúnak kell lenniük. A másik ok, amiért egy idő után nem folytatódhat tovább a Moore által megjósolt trend az az, hogy ahogy az eszközök mérete közelíti az atomi szintet (nanométeres nagyságrend), úgy „változik a fizika”: egyre inkább érvényesülnek a klasszikus fizikával nem leírható, úgynevezett kvantumjelenségek.

Több különböző becslés is létezik arról, hogy mikor veszti érvényét a törvény: vannak, akik szerint már 2020 körül [3], és vannak olyanok is, akik szerint 2030 táján [2]. A jóslatok azonban megegyeznek abban, hogy 2050-ig a tranzisztorok mérete eléri az elvi alsókorlátot.



1. ábra: A Moore törvény vége

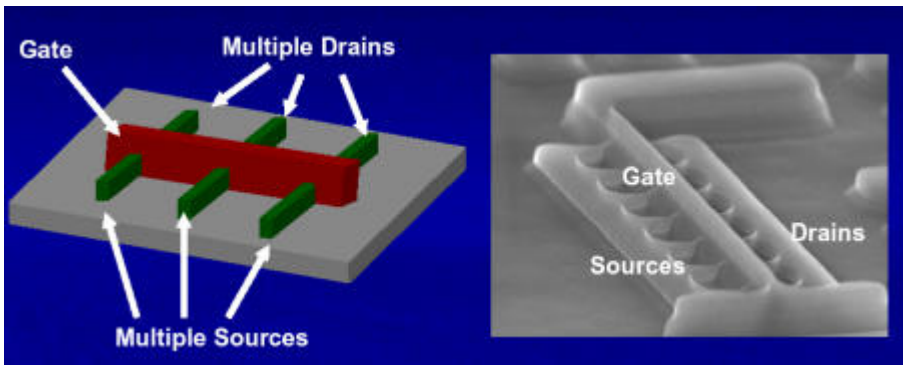
b) A Moore-törvény után

A világ számításigénye azonban folyamatosan növekvő, így a számítógépek fejlődése sem állhat meg: a számítástechnika története valószínűleg nem fejeződik be Moore

törvényével.

Több megközelítés létezik a probléma megoldására. Van, amelyik a tranzisztor továbbfejlesztését tűzte ki céljául (Intel – tri-gate). Vannak, amelyek a meglévő architektúrájú gépek felhasználásával alakítanak ki egy nagyobb rendszert (GRID, cloud), és van olyan megoldás, amelyik teljesen új architektúrát kínál: a kvantumszámítógép architektúráját.

Az Intel egy új technológia, az úgynevezett. 3-D Tri-Gate tranzisztorok fejlesztésével próbálja kitolni a Moore-törvény végét [4]. Az Intel eljárásának alapötlete az, hogy az eddigi sík (vagy más szóval planáris) helyett térbeli tranzisztorokat alkalmaznak. Az újítás eredményeképpen nem csak méret csökkenés érhető el, hanem energia megtakarítás is. Könnyen belátható, hogy –hasonlóan a hagyományos tranzisztorokhoz – ennek a technológiának is van méretbeli alsókorlátja. A kérdés az, hogy a technológia mikor éri ezt el.



2. ábra: 3D tranzisztor

Más irányzatok a meglévő hardverelemek felhasználásával építenek nagyobb kapacitású rendszereket. Mind a lazán, mind a szorosan csatolt rendszerek több CPU-t használnak a számítások elvégzéséhez.

A szorosan csatolt rendszerek egyre népszerűbbek az otthoni felhasználók körében: rengeteg hirdetéssel lehet találkozni, melyek dual-/multicore laptop vásárlására buzdítanak. Az ilyen gépekben egy alaplapon kettő vagy több processzor található. A több CPU-t a multitasking operációs rendszerek úgy használják ki, hogy a különböző feladatokat különböző processzoroknak osztják ki, amelynek eredménye a valódi párhuzamosság. (Természetesen valódi párhuzamosság csak akkor érhető el, ha a futtatandó processzek száma nem nagyobb a magok számánál. Az elv akkor is hasonló, ha több folyamat fut, mint ahány processzor van a rendszerben.) Szemben a klasszikus, egy processzoros rendszerekkel, ahol architekturális okok miatt csak a logikai szintű párhuzamosság valósulhat meg, hiszen ebben az esetben minden folyamat csak egy bizonyos időszeletig használhatja a rendszer erőforrásait. A többprocesszoros megoldás alkalmazása azonban látványos sebességnövekedéshez vezethet. Az eljárás teljesítménybeli korlátja az, hogy az egy alaplapra integrálható processzorok száma véges.

A lazán csatolt rendszer elnevezés olyan architektúrát jelöl, amelyben az egyes résztvevő számítógépek közösen, hálózaton kommunikálva oldanak meg feladatokat. Ilyen architektúra például az úgynevezett számítási felhő (cloud computing)[5] és a GRID [6]. Cloud computing esetén egy szolgáltató biztosít erőforrásokat (jellemzően tárhelyet), amelyet a felhasználók sajátjukként használhatnak, mintha a távoli mappa a gépük fájlrendszerében lenne. A GRID-nél a rendszerben résztvevő felhasználók osztják meg egymással az erőforrásokat (itt már jellemzően processzor időt és esetleg egyéb más hardvereket is, nem csak tárhelyet). Mindkét elv egyre népszerűbbé válik, azonban mindkettőnek meg vannak a maga gyengeségei is. Fontos, részben már megoldott, probléma a biztonság kérdése; az interneten, egymás számára teljesen idegen gépek között jóindulatú viselkedést feltételezni

naiv megközelítés volna. A másik alapvető probléma az, hogy lazán csatolt rendszerek esetén semmit nem tudunk a résztvevő gépek állapotáról: működnek-e, ott vannak-e egyáltalán. Ez a probléma az adatok, illetve az erőforrások rendelkezésre állása miatt kritikus.

c) A kvantumszámítógép

A hardvergyártás teljesen új megközelítése a *kvantumszámítógép*. Richard P. Feynman egy 1982-ben tartott előadásában azt a kérdést vizsgálta, hogyan számítógépen lehet szimulálni tetszőleges fizikai rendszert [7]. Arra a következtetésre jutott, hogy kvantumrendszer csak olyan számítógépen szimulálható, amely maga is kvantumjelenségeket használ a számításhoz. Ebben az előadásban merült fel először a kvantumszámítógép ötlete. Később világossá vált az is, hogy a kvantumszámítógép bizonyos klasszikus problémákat jelentős sebességnövekedés mellett oldana meg. A teljesítménynövekedés az úgynevezett kvantumjelenségek – mint az összefonódás vagy a szuperpozíció elve – használatából adódik. A szuperpozíció elve azt jelenti, hogy egy *qbit* (quantum bit: a kvantumszámítógépen tárolt egységnyi adat) nem logikai 0 vagy 1 értéket tárol, hanem ezek szuperpozícióját – azaz egyszerre mind a két értéket. Az összefonódás (entanglement) jelensége pedig azt jelenti, hogy kettő vagy több qbit együtt tárol egy értéket, azaz az összefonódásban szereplő bármelyik bit értékének ismeretében meghatározható, hogy a többi bit milyen értéket vesz fel. (A kvantum jelenségeket részletesen később ismertetem.)

A kvantumszámítógép forradalmi változásokat hozhat a számítástechnikában. Ennek szemléltetésére nézzünk meg néhány hatékony kvantumalgoritmust!

Híres eredmény a Lov Grover nevéhez köthető eljárás. A Grover-algoritmus $O(\sqrt{N})$ idő alatt keres meg egy értéket rendezetlen adatbázisban.[8] Klasszikus esetben a triviális megoldás (vagyis egyesével végignézni az adatbázis elemeit és egyezést keresni) a leghatékonyabb.

Másik jelentős eredmény Shor algoritmusa, amely kvantumszámítógépen polinom időben végez prímtényezős felbontást. A művelet időigénye: $O(ld^3(N))$.[9] Mivel a ma még nagy népszerűségnek örvendő RSA-kódolás alapötlete az, hogy egy nagy szám prímtényezős felbontása időigényes feladat, könnyű belátni, hogy a kvantumszámítógép megjelenése gyökeres változásokat hozna.[1] (Ugyanakkor fontos megjegyezni, hogy Shor algoritmusa illetve a prímtényezős felbontás nem csak kriptográfiai megfontolásokból nagyon lényeges).

A hatékony algoritmusok mellett a kvantumszámítógép paradigmaváltást is hozna magával. Hangsúlyoznunk kell, hogy nem csupán jobb tranzisztorról, vagy valamilyen más jól használható áramköri elemről van szó. A kvantumszámítógép és a klasszikus megoldás között magasabb absztrakciós szinten, logikailag is különbség van.

A fentebb említett elgondolások (párhuzamos programozás, 3D tranzisztorok, stb.) még mind alapvető implementációs problémákkal küzdenek. Nem kivétel ez alól a kvantumszámítógép sem.

A gyakorlatban még nem sikerült kvantumszámítógépet építeni. Az eddigi legnagyobb teljesítményű ilyen eszköz 4 qbit-es volt és Shor algoritmusa futott rajta (a tizenötöt bontotta fel prímtényezőkre). [10] A fizikai implementáció fő nehézsége a biteket megvalósító kvantumrendszerek izolálása, hiszen a bitek környezettel való összefonódása megmásítani az eredményt. Az eszközzel azonban valamilyen módon kommunikálni kell, ami az izolációval ellentétes folyamat.

Bizonyos vélekedések szerint a kvantumszámítógép létrehozása fizikai képtelenség.

[1] Megjegyzendő, hogy már külön tudományág foglalkozik olyan kódolási eljárások előállításával, amelyek kvantumszámítógéppel is csak lassan törhetők fel (kvantum-NP teljes problémák). Az új diszciplína neve: posztkvantum-kriptográfia.

Mindenesetre, ha egyértelmű választ kapnánk arra a kérdésre, hogy lehetséges-e a fizikai implementáció vagy nem, az hatalmas előrelépést jelentene a tudománynak.

2.2. Miért hasznos a kvantumszámítógép szimuláció?

Kvantumszámítógép szimulátorra azért van szükség, mert a jelenlegi technológiai fejlettség mellett csak korlátozott méretben lehetséges a fizikai implementáció. Annak ellenére, hogy valódi kvantumszámítógép még nem áll rendelkezésünkre, máris nagyon látványos eredmények születtek (mint Grover és Shor algoritmusa). Véleményem szerint további fejlődéshez vezethetne, ha hatékony szimulációs programok segítenék a kutatók munkáját. Hiszen sokszor akár egy pár bites szimuláció is szemléletesebb képet adhat, mint az elméleti megfontolások.

A szimulációs eszköz nemcsak a kutatásban, de mint demonstrációs eszköz az oktatásban is hasznosulhat.

3. A kvantuminformatika alapfogalmai

Az alábbi fejezetben a kvantuminformatika olyan alapvető fogalmait vezetem be, mint a kvantumbit, szuperpozíció, mérés stb. Az alapfogalmak ismertetése után a szimuláció által használt matematikai leírás bemutatása következik. A fejezet végén az egyes hardverarchitektúrák (CPU és GPU) felépítése és a szimuláció sebessége közötti összefüggéseket vizsgálom.

3.1. Kvantuminformatikai alapfogalmak

A szimuláció működésének megértéséhez szükségesnek tartom néhány fogalom bevezetését.

a) Kvantumbit (qbit), szuperpozíció [11]

Kvantumszámítógépen az információtárolás alapegysége a kvantumbit (angolul: quantum bit, vagy rövidítve qbit. Ez utóbbi elnevezés a magyar szakirodalomban is használatos).

Egy klasszikus bit a logikai 0 illetve 1 értékek tárolására alkalmas: egy adott pillanatban vagy az egyiket, vagy a másikat (de mindig csak az egyiket) tartalmazza. A tárolóegység kiolvasása szintén nagyon egyszerű: ha a bit egy adott logikai értéket tárol, akkor minden egyes kiolvasásnál az aktuálisan tárolt értéket kapjuk eredményül.

Ezzel szemben egy qbit a lehetséges logikai értékek szuperpozíciójában van, azaz a kvantum rendszer mindkét értéket egyszerre tárolja, mindegyiket különböző valószínűséggel. Az egy bites kvantumrendszer úgynevezet állapotvektoros leírása:

$$|\varphi\rangle = a|0\rangle + b|1\rangle$$

A $|\varphi\rangle$ (ejtsd: „ket fi”) rendszert $|a|^2$ valószínűséggel mérjük 0-s logikai állapotban, $|b|^2$ valószínűséggel pedig egynek. Az a és b együtthatókat a rendszer valószínűségi amplitúdójának nevezzük (a és b komplex számot alkotnak). Az együtthatóknak ki kell elégíteniük az alábbi egyenlőséget: $|a|^2+|b|^2 = 1$.

Mivel a qbit a két lehetséges logikai változó szuperpozíciójában van, ezért az aktuális érték meghatározásához meg kell mérni a rendszert.[2] (Később a kérdésről még részletesebben lesz szó)

Vegyük észre, hogy a kvantuminformatika speciális esete (elfajulása) a klasszikus, mégpedig akkor, hogy ha az egyik együttható egy, míg a másik nulla.

b) Összefonódás (entanglement) [12]

Nagyon fontos – és nagyon hatékonyan használható – kvantumos jelenség az összefonódás. A kvantummechanika idevágó posztulátuma szerint két rendszer együttes állapota meghatározható az egyes rendszerek tenzorszorzatával. Ebből következik, hogy bizonyos összetett rendszerek felbonthatóak két olyan kvantum rendszerre, amelyek tenzorszorzata maga az összetett rendszer. Például:

[2] A klasszikus és a kvantum bit közötti különbséget a pénzfeldobás analógiája ábrázolja szemléletesen: a klasszikus bitet pénzérmeként képzelhetjük el, amely a földre érkezve megállapodik az egyik „logikai értéken” (fej vagy írás). A kvantum bit ezzel szemben egy levegőben pörgő érme: mindkét lehetőséget tárolja egyszerre, kiolvasható értéket pedig „méréskor” nyer, amikor leérkezik a földre.

$$a|00\rangle + b|01\rangle = |0\rangle \otimes \left(a|0\rangle + b|1\rangle\right)$$

Ugyanakkor az alábbi rendszer felbontása ily módon nem lehetséges:

$$a|00\rangle + b|11\rangle$$

Látszik, hogy csupán egyetlen bit értékének ismeretében meg tudjuk határozni, hogy a rendszer melyik állapotban van. Ha az első bitnek nullát mérünk, akkor biztos, hogy a másik is nulla, hasonlóan, ha egyet mérünk, akkor biztos, hogy a másik értéke is egy. A „rejtélyes kapcsolat" akkor is fennáll az egyes bitek között, ha a rendszer két tagját távolabb viszik egymástól.

A különös jelenség, itt nem részletezendő okok miatt nem alkalmas a fénynél gyorsabb kommunikáció megvalósítására, azonban egyéb alkalmazásokban, mint a szupersűrűségű tömörítésben fontos szerepet játszik.

c) Műveletek [13]

A kvantummechanika második posztulátuma szerint minden zárt fizikai rendszer változása leírható unitér transzformációkkal. Egy transzformáció unitér, ha a lineáris algebrai reprezentációjára igaz, hogy

$$U^{\dagger} = U^{-1}.$$

Az unitér transzformációk több jól használható tulajdonsággal rendelkeznek: a mátrixuk négyzetes, az általuk leírt művelet pedig reverzibilis, ami azt jelenti, hogy a kapott eredményen a transzformáció inverzét elvégezve megkapjuk a kezdő állapotot.

A fentiekből következik, hogy minden logikai művelet („logikai kapu") leírható unitér transzformációkkal, azaz egy pontosan definiált matematika formula adódik az áramkör működésének reprezentálására.

Az unitér transzformációkkal történő műveletvégzés következménye, hogy míg klasszikusan az áramkör egyik elemét (logikai kapuját) annak igazságtáblájával adjuk meg, kvantumosan a mátrix reprezentáció a bevett mód.

d) Mérés [14]

Klasszikus információtárolás esetében egy adott tárolóegység tartalma tetszőleges számúszor kiolvasható, anélkül, hogy a tárolt érték megváltozna (tekintsünk el az igénybevétel okozta fizikai amortizációtól). Ezzel szemben egy kvantum rendszerben tárolt érték kiolvasása („mérése", measurement) megváltoztatja az adattárban tárolt értéket, azt úgynevezett „mérés utáni állapot"-ban (post measurement state) hagyja. A mérés tehát eltér az eddig megismert műveletektől, hiszen nem unitér transzformáció. Mégpedig azért nem, mert a második posztulátum (amely az unitér transzformációs leírást bevezeti) zárt fizikai rendszerről beszél. A mérés ezzel szemben átjárást biztosít a kvantumos és a klasszikus világ között.

Kvantummechanikai rendszerek mérése, illetve a tény, hogy egy rendszer a mérés után a mérés előttitől különböző állapotba kerül, a fizikán túlmutató kérdéseket vet fel.

Ilyen fizikán túlmutató kérdés, hogy mi történik a rendszerben két mérés között. Az egyik filozófiai iskola szerint a rendszerben a mérés eredményeképp történnek a változások, tehát ha nem vizsgáljuk a rendszert, akkor nem történik és nincs is ott semmi. Egy másik iskola szerint két mérés között sok-sok párhuzamos dimenzióban egyszerre változik a rendszer állapota (mindegyikben máshogy) és a mérés ekvivalens azzal, hogy a párhuzamos dimenziók közül kiválasztódik egy.

3.2. A kvantumszámítógép matematikai leírása

a) Állapotvektoros leírás

A fentebb leírtak szerint egy kvantumbit értékét egy művelet (mondjuk a Pauli-Z) elvégzése után úgy kaphatjuk meg, hogy a művelet mátrixának és a qbit-et leíró állapotvektor szorzatát vesszük. Például:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} a \\ -b \end{pmatrix} = a|0\rangle - |1\rangle$$

Működőképes (és matematikailag helyes) gondolat, ám több bites kvantumregiszterek esetén nehezen kezelhetővé válik. A problémára a sűrűségmátrixos leírás jelent megoldást.

b) Sűrűség mátrixos leírás [15]

Egy kvantum rendszer sűrűségmátrixát az alábbi képlet alapján számolhatjuk:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|,$$

ahol p_i az i-edik állapot előfordulásának klasszikus valószínűségét jelenti. Ha a rendszer úgynevezett tisztaállapotban (pure state) van, akkor csak egy értéket tárolhat, így a képlet az alábbira egyszerűsödik:

$$\rho = |\psi\rangle\langle\psi|$$

Egy rendszer tisztaállapotban van, ha nem két rendszer összefonódásából (entanglement) adódik, ellenkező esetben úgynevezett kevert állapotról (mixed state) beszélünk.

A sűrűségmátrixos leírás segítségével egyszerűen végezhetőek el a különböző unitér transzformációk:

$$\begin{aligned} U\sum_{i=1}^{n} p_i |\psi_i\rangle\langle\psi_i| U^\dagger &= \sum_{i=1}^{n} p_i U |\psi_i\rangle\langle\psi_i| U^\dagger \\ &= U\left(\sum_{i=1}^{n} p_i |\psi_i\rangle\langle\psi_i|\right) U^\dagger \\ &= U\rho U^\dagger, \end{aligned}$$

ahol ρ a rendszer állapotát leíró sűrűségmátrix, U pedig az unitér transzformáció mátrixa.

A mérések – a műveletekhez és a kvantumrendszerek állapotához hasonlóan – mátrixokkal adhatóak meg, egy mérést azonban több mátrix reprezentál. A mérést leíró mátrixoknak ortonormálisnak kell lenniük, az összegüknek pedig az identitás mátrixot kell adniuk.

A mérés első lépéseként véletlenszerűen kisorsolódik egy mátrix. Az egyes mérési mátrixok kisorsolásának valószínűségét az alábbi képlet adja meg:

$$\Pr\left[\ j\middle|\rho\right] = Tr\left(P_j\rho\right) = Tr\left(P_j^2\rho\right) = Tr\left(P_j\rho P_j^\dagger\right) = Tr\left(P_j\left[\sum_{i=1}^{n} p_i\left|\psi_i\right\rangle\left\langle\psi_i\right|\right]P_j^\dagger\right).$$

Mivel a mérés nem hagyja változatlanul a rendszert, ezért a mérés utáni állapot (post measurement state) számításához az alábbi műveletet kell elvégezni:

$$\rho_j = \frac{P_j\rho P_j}{Tr\left(P_j\rho P_j\right)} = \frac{P_j\left[\sum_{i=1}^{n} p_i\left|\psi_i\right\rangle\left\langle\psi_i\right|\right]P_j}{Tr\left(P_j\left[\sum_{i=1}^{n} p_i\left|\psi_i\right\rangle\left\langle\psi_i\right|\right]P_j\right)}$$

A nevezőben található „*Tr*” a trace, azaz nyomkövetés műveletére utal. A nyomkövetés az adott mátrix főátlóiban található értékek összege. Képlettel:

$$Tr\left(A\right) = a_{11} + a_{22} + \mathrm{K} + a_{nn} = \sum_{i=1}^{n} a_{ii}$$

A mérés folyamata befejeződött, hiszen ismerté válik a kvantumrendszer új állapota.

A sűrűségmátrixos leíráshoz kapcsolódik a fentieken túl a „partial trace” nevű művelet, amelynek segítségével egy rendszer alrendszerei vizsgálhatóak. (Magyarán a sűrűségmátrixos leírásból kinyerhetőek az egyes kvantumbitek értékei.)

3.3. Hardveres adottságok, a szimuláció megvalósíthatósága

a) Miért nehéz kvantumszámítógépet szimulálni?

A fentiekben olyan matematika leírást ismertünk meg, amely alkalmasnak tűnik arra, hogy klasszikus számítógépen műveleteket végezzünk vele. Érdemes megvizsgálnunk a szimulációval kapcsolatos nehézségeket, az okokat, amiért nem készülnek tucat számra több száz bitből álló kvantum rendszereket szimuláló eszközök asztali PC-re.

A fő nehézség, hogy exponenciálisan nő a tárolandó adatok száma, ahogy az a sűrűségmátrixok kiszámolását megadó képletből is kitűnik:

$$\rho = \left|\psi\right\rangle\left\langle\psi\right| = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}\begin{pmatrix}\alpha^* & \beta^*\end{pmatrix} = \begin{pmatrix}\alpha\alpha^* & \alpha\beta^*\\ \alpha^*\beta & \beta\beta^*\end{pmatrix} = \begin{pmatrix}\left|\alpha\right|^2 & \alpha\beta^*\\ \alpha^*\beta & \left|\beta\right|^2\end{pmatrix}$$

Egy n bites regiszter 2^n db értékkel írható le az eredeti leírásunk szerint. A sűrűségmátrix olyan négyzetes mátrix, amelyet egy 2^n méretű oszlop és egy ugyanekkora dimenziójú sorvektorból számolható ki, a sűrűség mátrix tehát 2^{2n} db értéket tartalmaz. (lásd a mellékletben „A szimuláció számításigénye” című táblázatot).

A táblázatból látszik, hogy milyen gyorsan, milyen mértékben növekszik (növekedne) a tárigény. (Megjegyzendő, hogy csupán az adatok tárolásáról tettünk eddig említést, a feldolgozásról még szó sem esett.)

A táblázatban kiszámolt adatok ismeretében világos, hogy miért komoly mérnöki probléma a kvantumszámítógép-szimuláció.

b) Egy másik kvantumszámítógép szimuláció [16]

A következőkben vizsgáljunk meg egy a Jülich Computer Centre által létrehozott szimulátort. A program, az alkotók állítása szerint, korának legnagyobb teljesítményű ilyen típusú alkalmazása volt. Maximálisan 42 bit szimulálására képes, amely jelentősen meghaladja a ma létező (egy-két bites) valódi kvantumszámítógépek teljesítményét.

Tanulságos megvizsgálni a rendszer hardverigényeit is: a program JUGENE nevű szuperszámítógépen fut, amely Európa leggyorsabb gépe. A számítógép közel 300000 processzort tartalmaz.

A tesztelés során Shor algoritmusát futtatták le a kutatók: a gép a 15707 számot bontotta fel 113 és 139 szorzatára.

A rendszer hardverkonfigurációjából is látszik, hogy a kvantumszámítógép szimuláció nem magától értetődő feladat, hiszen speciális hardveren háromszázezer processzorral sem lehet negyvenkét bitnél többet szimulálni.

c) Az én megoldásom célja, az alkalmazott ötletek

Egy számítógépes programmal szemben támasztott igények nagyban függnek attól, hogy kik a potenciális felhasználói. Az én munkám célja olyan szimulációs program vázának megalkotása, amely közönséges asztali számítógépen - tehát célhardver használata nélkül – futtatható. Értelemszerűen ez a kikötés csak tizenegy-néhány bites rendszerek szimulálását teszi lehetővé.

A kezdetektől világos volt, hogy a szimuláció nem valósítható meg CPU-n hatékonyan. Ezzel szemben GPU-n látványos sebesség növekedést sikerült elérni. A rendszermagját ezért egy olyan eljárás adja, amely a videókártyán hajtja végre a mátrixszorzást.

A grafikuskártya felhasználása nélkül az általam használt modell szimulálása a gyakorlatban nem lehetséges, mert CPU-n két nagyobb (>1000x1000) mátrix összeszorzása percekben mérhető időt vesz igénybe. Világos, hogy bonyolultabb áramkör szimulálása esetén műveletenként perces nagyságrendű időigény nem engedhető meg.

4. A program

Ebben a fejezetben a választott hardver architektúra (GPU) bemutatására kerül sor. Bemutatom röviden a hardver történetét, a működés elvét és azt, hogy a videókártya kedvező tulajdonságai hogyan használhatók fel a szimuláció során. Ismertetésre kerül a mátrixszorzást a GPU-n megvalósító algoritmus működése.

4.1. A program működése

a) GPU [17][18]

1) Mi a GPU?

Ahhoz, hogy világos legyen, hogy mi a GPU (graphics processing unit – grafikus feldolgozó egység) és mire alkalmazható, meg kell ismerkednünk a grafikus megjelenítés rövid történetével.

Az első időkben a grafikus kártya nem volt más, mint egyszerű csővezeték, amely a processzor felöl érkező adatokat továbbította a képernyőre. Mindaddig, amíg parancssoros vagy egyszerű grafikájú rendszereket kellett megjeleníteni, a pipeline elrendezés működőképesnek bizonyult. A megjelenítés fejlődésével azonban egyre nagyobb igények merültek fel, amelyeket a hardvergyártók egyre összetettebb áramköröket használó kártyákkal igyekeztek kielégíteni. A komplexitás növekedésével mind hangsúlyosabban fogalmazódott meg a programozható grafikus hardver iránti elvárás. Ennek eredménye a GPU.

A ma ismert GPU-k több száz feldolgozóegységet tartalmaznak, melyek lehetővé teszik a feldolgozás párhuzamosítását. Ebben rejlik a módszer ereje a CPU-khoz képest.

Egy grafikus kártyára írt eljárás koncepciója jellemzően a következő: megfogalmaz egy rutint, amely egy egyszerűbb műveletet ír le (például két szám összeszorzása), majd a GPU memóriába másolja az adatokat. (Az adatok mozgatása lassú művelet, ezért sebességkritikus alkalmazások esetén ezek számát minimalizálni kell.) Ha a grafikus kártya memóriájában rendelkezésre állnak az adatok, akkor a program több szálon („thread”) elindítja az egyszerű rutint, azonban mindegyik szálon más és más paraméterezéssel. A hardver felépítéséből adódóan az elindított threadek egy része párhuzamosan számítható, hiszen a futás során nem csak logikailag, de fizikailag is elkülönülnek.

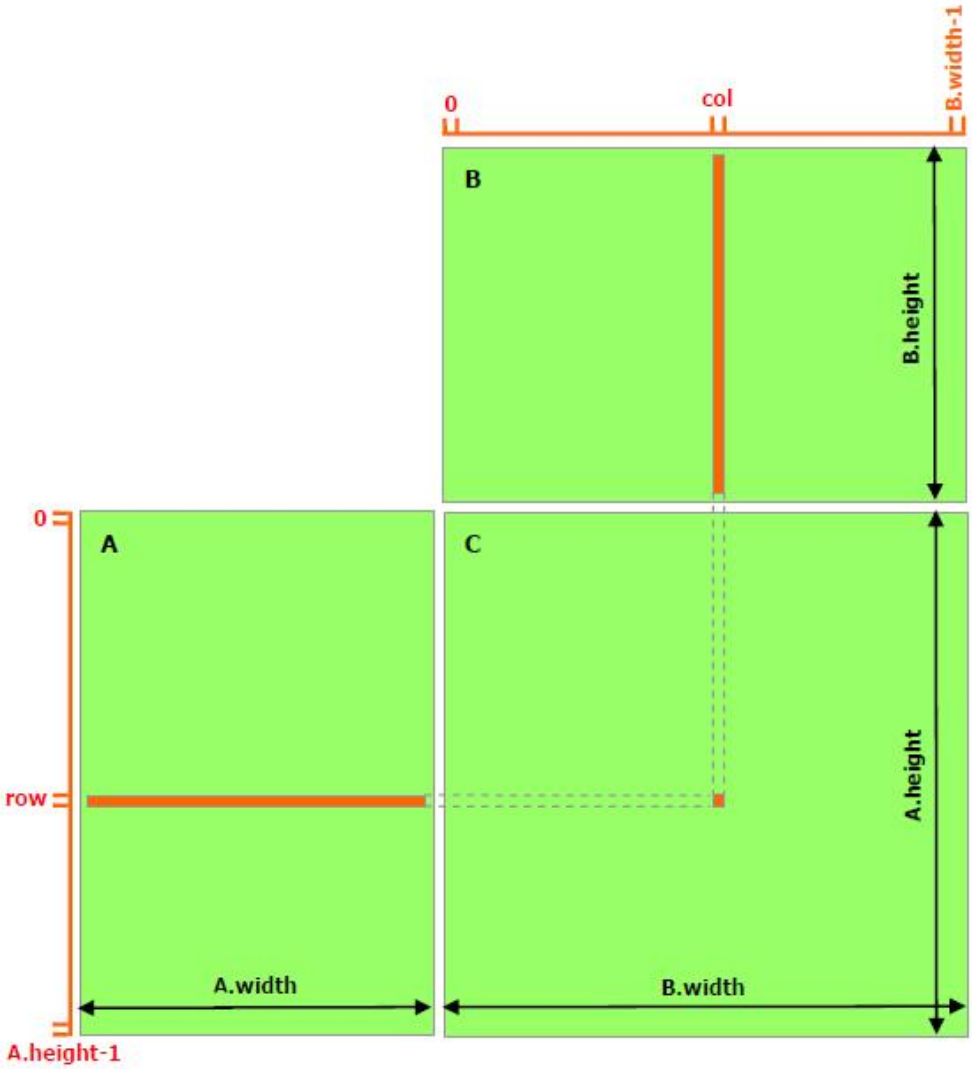
A grafikuskártyát jellemzően olyan feladatok megoldására használják, ahol sok adaton kell ugyanazokat a műveleteket elvégezni egymástól függetlenül (például kriptográfiai vagy orvosi alkalmazások esetén)

2) Hogyan alkalmazható a GPU kvantumrendszerek szimulálására?

Az ismertetett matematikai leírásból is kitűnik, hogy egy kvantumrendszer működése vizsgálható mátrixokkal és azokon végzett műveletekkel (sűrűségmátrixos leírás, unitér transzformáció, mérés, stb.)

A mátrixszorzás kiválóan párhuzamosítható művelet, hiszen az eredmény mátrix minden tagja egymástól függetlenül számítható. Vizsgáljuk meg, hogyan is történik a számítás!

Tegyük fel, hogy két darab 32X32-es mátrix összeszorzása a cél. Ekkor az eredmény is 32X32-es mátrix lesz, ami azt jelenti, hogy az eredmény mátrixnak összesen 1024 tagja van. A futás során az OpenCL 1024 szálat (threadet) indít el (ha a hardveres környezet lehetőséget ad erre, akkor egyszerre az összeset, ha nem, akkor annyit, amennyit lehet). Az egyes threadek az eredmény mátrixban elfoglalt helyüknek megfelelő értékek kiszámítását végzik.



3. ábra: A mátrixszorzás elvi vázlata

A 3. ábra illusztrálja a mátrixszorzás műveletét. A C mátrixban jelölt narancssárga téglalap az eredménymátrix aktuális eleme, melynek kiszámítása az „A” és „B” mátrixok jelölt oszlopai illetve sorai alapján történik. A GPU-n való számolás lényege pedig az, hogy mindegyik narancssárga pontnak megfelelő egységet egy-egy thread párhuzamosan számol ki.

3) Implementációs kérdések

A GPU-n futtatandó logikát OpenCL-ben írtam meg.[3] Annak, hogy erre a keretrendszerre esett a választásom, több oka volt.

Az egyik ok a keretrendszer platformfüggetlensége: [18] CUDA-ban csak olyan programokat lehet írni, amelyek az Nvidia bizonyos termékein működőképesek, de más platformon nem. Ezzel szemben az OpenCL – újrafordítás után – több gyártó kártyáin is fut.

Fontos tulajdonsága az OpenCL-nek, hogy támogatja az úgynevezett heterogén környezet létrehozását is. [19] Ez a funkció lehetővé teszi olyan eszköz használatát, amelyben több GPU (akár különböző típusúak is) végzi a számításokat. A használatot egyszerűsíti, hogy a keretrendszer elrejti a fizikai réteget a fejlesztő elől. Az inicializálási szakaszban a programozó kialakítja a heterogén környezetet: létrehoz egy úgynevezett kontextust, amihez hozzáadja a kívánt feldolgozó egységeket. Ezt követően a programozás úgy történik, mintha

[3] http://www.khronos.org/opencl/, 2011-10-20

egyetlen, a megadott részekből összeadott hardveren folyna a fejlesztés.

Ha tehát valaki egy GPU helyett négyet használ, akkor például 512 szál helyett 4x512-t, azaz kétezer negyvennyolcat futtathat egyszerre. Látható, hogy több egység használata jelentős sebességnövekedést eredményezhet az egy GPU-s rendszerhez képest. Megjegyzendő, hogy nem csak több GPU-t, de akár CPU-t (vagy CPU-kat) is adhatunk a rendszerhez.

A projekt továbbfejlesztése során hasznos lehet az, hogy az OpenCL kezeli a heterogén kontextust, én azonban nem használtam ki ezt a tulajdonságát, mégpedig a számításigény növekedésének üteme miatt nem. Tizenhárom bites rendszer szimulálható egy 1 GB memóriájú videókártyán. Nagyobb rendszer esetében azonban nem megoldható, hogy egy műveletet a mátrixok tördelése nélkül elvégezzünk. Ebből az következik, hogy a mátrixok részeit a háttértárról kell beolvasni, amely a működés jelentős lassulását eredményezi. Megfontolandó, hogy érdemes-e megvalósítani a ezt változtatást, hiszen egy-két bit növekedésért cserébe a program bonyolultabbá válik, sebessége pedig csökken.

Az én választásom az egyszerűbb megoldásra esett, mert úgy gondolom, hogy fontosabb, hogy az alkalmazás kisebb, gyorsabb legyen, mint az, hogy egy-két bittel nagyobb rendszer szimulálására legyen képes.

4.2. A program felhasználási lehetőségei

Mint arra már a bevezetőben is utaltam, a kvantumszámítógép forradalmi változásokat hozna az informatika és az élet egyéb területein. A forradalmi változásokra azonban várni kell, hiszen az architektúra fizikai implementációja még sok megoldatlan problémával küzd. A konkrét megvalósítás hiánya ellenére sok, nagyon látványos kvantum algoritmus vált ismerté az elmúlt időkben, amelyek tovább inspirálják a diszciplína fejlődését (mind hardver, mind szoftver tekintetében).

Addig azonban, amíg a fizikai implementáció várat magára, a kvantuminformatika eredményei nehezen kommunikálhatóak, megfoghatatlanok maradnak. Egy mindenki által elérhető szimulációs környezet azzal, hogy közelebb hozza a szélesebb kutatói, fejlesztői rétegekhez az új paradigmát, megoldja ezt a problémát, amely további fejlődéshez vezetne. Közismert például, hogy a számítógépek modern történetében fontos áttörést jelentett, amikor szabadon programozhatóvá váltak az eszközök, így a programozás feladata kiszabadulhatott a laboratóriumokból és megszűnt pár fejlesztő privilégiuma lenni.

5. Implementációs kérdések

A következő fejezetben bemutatom és értékelem a mérési eredményeket. Megvizsgálom, hogy konkrét megvalósítás esetén milyen arányú sebességkülönbség van a GPU és a CPU architektúra között. Kitérek arra is, hogy a GPU látványos gyorsasága ellenére is csak korlátos méretű kvantumrendszerek szimulálására alkalmas

5.1. Mátrixszorzás CPU-n, mátrixszorzás GPU-n

A következőkben implementációs kérdésekről lesz szó: megvizsgálom, hogy konkrét esetekben milyen mértékű teljesítménynövekedést érhetünk el GPU használatával.

A szimulációban a műveletvégzést unitér transzformációk reprezentálják; a műveletvégzés képlete:

$$\begin{aligned} U\sum_{i=1}^{n} p_i \left|\psi_i\right\rangle\left\langle\psi_i\right| U^{\dagger} &= \sum_{i=1}^{n} p_i U \left|\psi_i\right\rangle\left\langle\psi_i\right| U^{\dagger} \\ &= U\left(\sum_{i=1}^{n} p_i \left|\psi_i\right\rangle\left\langle\psi_i\right|\right) U^{\dagger} \\ &= U\rho U^{\dagger}, \end{aligned}$$

Ahol U az unitér transzformációt, ρ pedig a rendszer sűrűségmátrixát jelenti. A fenti képletből is kitűnik, hogy a kvantumrendszer időbeli változása mátrixszorzásokkal leírható. (A továbbiakban – a képlettől eltérően – azzal az egyszerűsítő feltételezéssel élek, hogy három helyett két mátrix összeszorzása megy végbe. Ennek előnye, hogy pontosabb mérések végezhetőek, ugyanakkor a levonható következtetések nem változnak)

A mátrixszorzás CPU-n a soros műveletvégzés miatt lassú művelet, ugyanakkor nagyon jól párhuzamosítható, hiszen az eredménymátrix elemei egymástól függetlenül számíthatóak. A GPU a probléma párhuzamosíthatósága miatt lesz a CPU hatékony kiváltója.

Mielőtt a mérést, illetve a mérési adatokat ismertetném, vizsgáljuk meg azt a kérdést, hogy architektúra váltás helyett nem lehetne-e valahogy máshogy – például egy ötletes algoritmussal – orvosolni a problémát!

Bizonyos matematikai problémáknál úgynevezett ritkamátrixok alkalmazása lehetséges. A ritkamátrixban nagyon sok zérus elem van; ilyen esetben a műveletvégzés triviálisan meggyorsítható. Kvantummechanikai rendszer leírásánál ez a koncepció nem alkalmazható, hiszen mind az unitér transzformációt megadó mátrix, mind a sűrűségmátrix az esetek döntő többségében „érvényes” (nem nulla) adatokkal töltött.

Célszerűnek látszik tehát a mátrixszorzáshoz a GPU szolgáltatásait igénybe venni.

5.2. Mérések, eredmények

A mérések az alábbi konfigurációjú számítógépen történtek: Intel Core2 Duo (kétprocesszoros rendszer), az egyes CPU-k sebessége egyenként 3,00 GHz. 4 GB memória állt rendelkezésre, operációs rendszer: Windows 7.

A GPU műveleteket NVIDIA GeForce GTX 560-as kártyán[4] teszteltem. A kártya memóriája 1024 MB.

a) Konfiguráció, az idő mérése

Egy program futási idejének pontos mérése nem egyszerű feladat, hiszen a mérést

[4] A konfiguráció részletes leírása: http://www.geforce.com/Hardware/GPUs/geforce-gtx-560/specifications

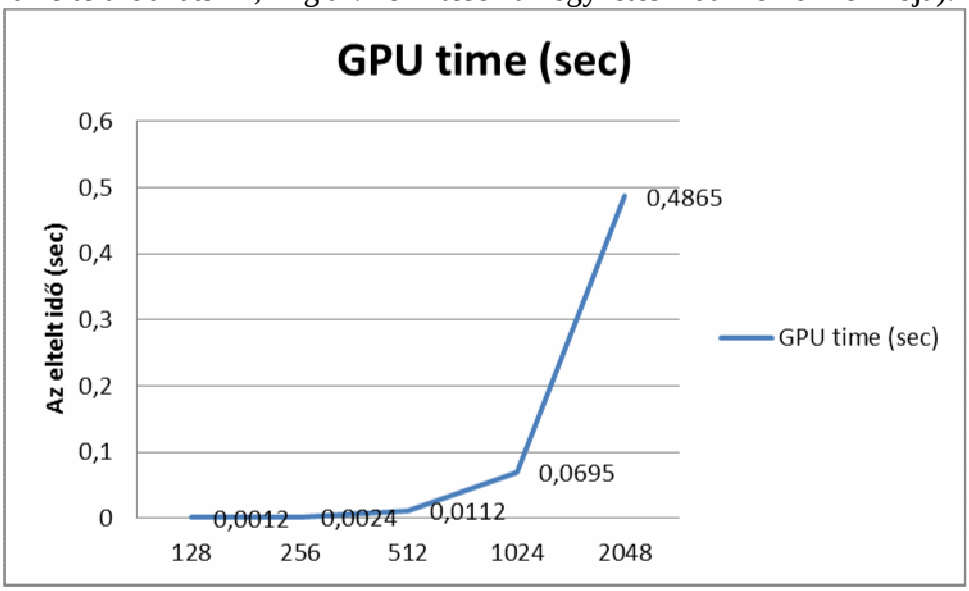
végző programrész ugyanazon a processzoron fut, mint amelyik magát a vizsgált programot menedzseli. Multitasking operációs rendszereknél pedig szinte biztos, hogy ezeken kívül fut másik alkalmazás is a CPU-n, amely tovább rontja a mérés pontosságát. Annak ellenére azonban, hogy a mért adatok a futás valódi hosszának csupán a közelítését adják, látványos eltérések tapasztalhatóak a GPU-n illetlve a CPU-n futtatott mérési adatok között.

A méréshez az ANSI C++ clock() eljárását alkalmaztam, ami az általam használt platformon az úgynevezett.wall-time-ot méri, azaz a futás kezdetétől eltelt időt. Bizonyos platformokon ugyanez a parancs csak azt az időt méri, amelyet a program a processzoron való futással tölt. Például a Sleep(1000) parancs esetén a processzoron töltött idő ~0 ms, míg a program futásának ideje ~1000 ms. Látható, hogy számunkra a wall-time (tehát a program futásának az ideje) a hasznos információ, hiszen a másik esetben a GPU számítást nullának mérnénk.

Az eredmények tíz mérés átlagából születtek.

b) A mért eredmények

A GPU-n futtatott mérések eredményét az alábbi grafikon szemlélteti (a függőleges tengelyen az eltelt idő látszik, míg a vízszintesen a négyzetes mátrixok dimenziója):

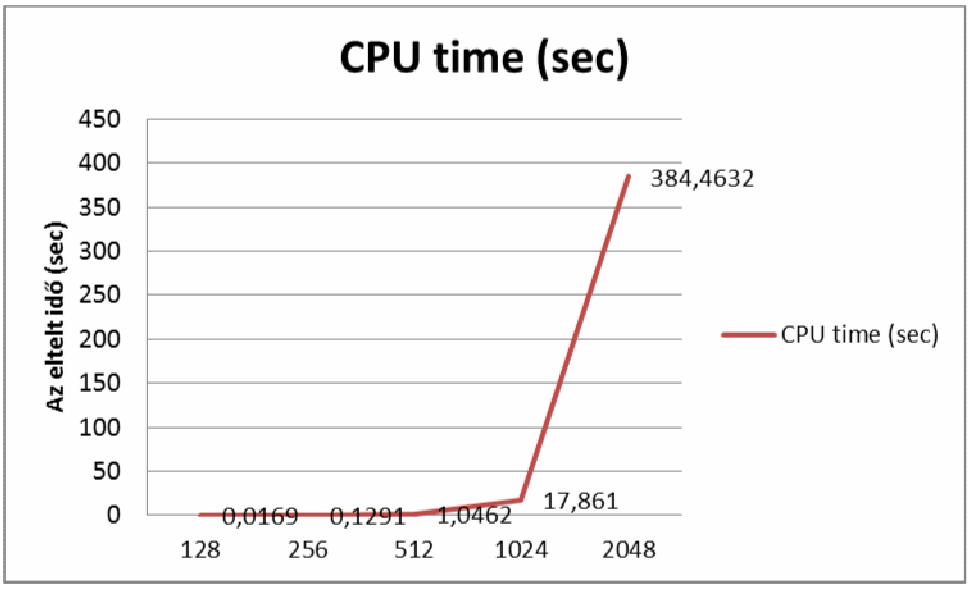


4. ábra: Mátrixszorzás GPU-n

A mérések során két négyzetes mátrix szorzását vizsgáltam, hiszen a kvantumrendszerekben is ezek játsszák a főszerepet (a sűrűségmátrix is négyzetes, az unitér transzformációk mátrixa is négyzetes, valamint a mérési mátrixok is négyzetesek).

A 4. ábrán megfigyelhető az exponenciálisnövekedés, ami nem meglepő, hiszen a kvadratikusmátrixok dimenziójának exponenciális függvénye a mátrix elemszáma. (Egy n dimenziós mátrixnak n x n eleme van.) Ez tehát azt jelenti, hogy a GPU-n nxn szál indul egyszerre, amelynek mindegyike n darab szorzást hajt végre.

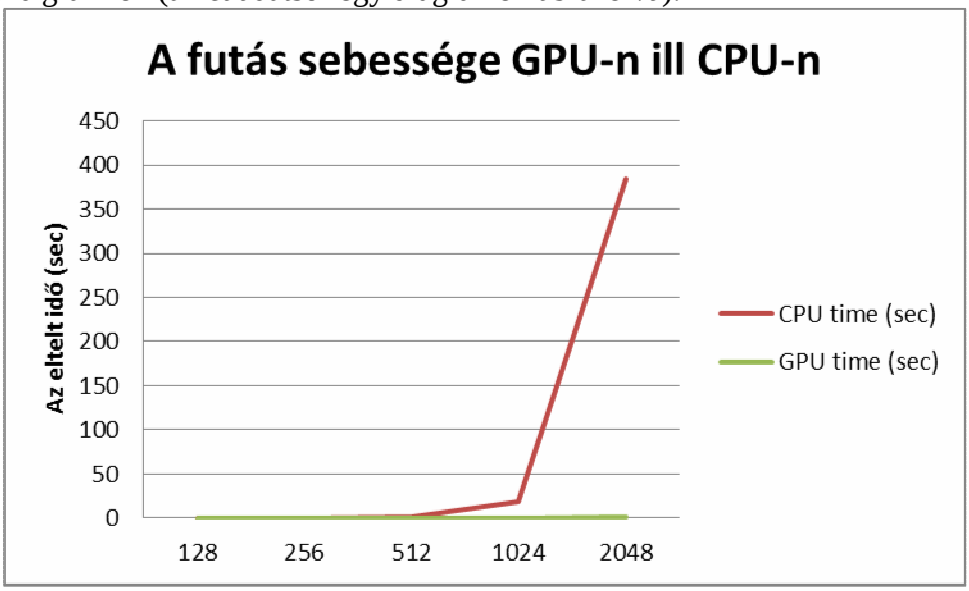
A CPU-n futtatott mérések eredményeit a következő grafikon szemlélteti (a tengelyek értelmezése az előzőhöz hasonló):



5. ábra: Mátrixszorzás CPU-n

Az 4. ábrán látható grafikonhoz hasonlóan itt is exponenciális növekedés figyelhető meg, amely azonban gyorsabb ütemű mint a GPU-n vizsgált esetben. Ennek oka a processzor sorosvégrehajtási modelljében keresendő: mindkét esetben ugyanannyi feladatot kell elvégezni, azonban GPU-n a szálak párhuzamosíthatóak (a végrehajtási idő tehát nem függ a threadek számától, csak a hosszuktól).

A két grafikon között a különbség a függőleges tengelyen látszik: míg a GPU-n a mért értékek maximuma sem éri el az 500 milliszekundumot (fél másodperc), addig a CPU-n a maximális érték 380 másodperc (~6 és fél perc) körüli. Ezt a jelentős különbséget illusztrálja a következő grafikon (a két adatsor egy diagramon ábrázolva):



6. ábra: Az adatok összevetése

c) Az eredmények értelmezése

A fenti grafikonok alapján jól látható a két architektúra közötti sebesség különbség: míg CPU-n egyetlen mátrixszorzás hat és fél percet vesz igénybe, addig GPU-n fél másodpercet. A gyakorlatban ritka az olyan áramkör, melynek működése egy elemi művelettel leírható lenne. A szimuláció műveletvégzési ideje a szimulált áramköri elemek

számával lineárisan nő. Könnyen kiszámolható, hogy a GPU-n futó program futása akár ~790-szer gyorsabb is lehet, mint a CPU-s változaté.

Felmerül a kérdés, hogy vajon a GPU-n való szimulációnak mi a korlátja? Vajon nagyobb (> 11 qbit) rendszerek villámgyors futtatására is képes a grafikuskártya? Az utóbbi kérdésre egyértelmű nem a válasz. A videókártya (ill. a párhuzamosan futtatható szálak ötlete) csak az exponenciálisan növekvő feladatok számára ad megoldást, az exponenciális ütemben növekvő tárigény kérdésére azonban nem. A felsőkorlátot a rendelkezésre álló memória jelenti.

Vizsgáljuk meg a bővíthetőség kérdését! Érdemes feldarabolni a mátrixokat és részenként elvégezni a műveletet. Tegyük fel, hogy a tárigény mátrixonként 4 GB (15 bites rendszer). A művelet memóriaigénye 8 GB (hiszen a feladat két mátrix összeszorzása), amely a mai technológiai fejlettség mellett jó eséllyel nem fér el a memóriában. Ha nem fér el a memóriában, ki lehetne olvasni a háttértárról, ami azonban látványosan lelassítaná a szimulációt. Hosszú másodperceket vesz igénybe egy gigabájtos szeletekben beolvasni egy binárisfájlt.

A háttértár lassú elérése ellenére akár alternatíva is lehetne a mátrixok felszeletelésének ötlete. De a tizenöt bites rendszert elég csupán négy bittel bővíteni ahhoz, hogy a rendszert leíró sűrűség mátrix elérje az egy terrabájtos nagyságot, ami már a háttértáron sem (vagy alig) fér el.

A fenti gondolatmenet miatt nem érdemes bonyolítani az algoritmust: mindegyik továbbfejlesztési ötlet egy-egy bitet adna hozzá a rendszer méretéhez, ugyanakkor nagyban megnehezítené a megvalósítást.

6. Következtetések, fejlesztési lehetőségek

Ebben a fejezetben összegzem az eddig leírtakat és megvizsgálom, hogy milyen további irányai lehetnek a fejlesztésnek, érdemes-e más architektúrára átírni a programot.

6.1. Következtetések

Hamar világossá vált a fejlesztés során, hogy a GPU használata, habár látványos sebesség növekedést eredményez, nem oldja meg a szimulálhatóság problémáit. A grafikushardver nyújtotta párhuzamosíthatóság eredményeképpen a műveletvégzés gyorsasága jelentősen nőtt, ugyanakkor az exponenciális tárigény növekedés okozta problémákat – értelemszerűen - nem küszöböli ki.

Asztali számítógépen, az ismertetett matematika leírást használva, célhardver nélkül a jelenleg elérhető technológia mellett 10-12 bites kvantumrendszerek szimulálása tűnik reálisnak.

6.2. Fejlesztési lehetőségek

Az előző fejezetekben felvázolt implementációs problémákon túl felmerülnek egyéb elvi kérdések is, például a teljesítmény növelésével kapcsolatban. Esetleg milyen más architektúrára lehet vagy érdemes továbbfejleszteni a programot? Az elsődleges felhasználáson (kvantum logikai áramkörök szimulálása) mi másra lehet még használni egy ilyen jellegű alkalmazást?

a) Teljesítménynövekedés más architektúrán

Érdemes megvizsgálni, hogy más platformokon milyen kilátásai vannak egy ilyen jellegű szimulációnak. Az exponenciális kapacitásigény növekedés és a kézenfekvő párhuzamosítási lehetőségek miatt érdemes lehet szuperszámítógépen vagy GRID-en vizsgálódni. Nyilvánvaló, hogy az általam vizsgált matematikai leírás használatával más architektúrán is exponenciális lesz a tárigény; nézzük meg, hogy ez mire lehet elég!

A HunGRID [20] projektben 2011. októberében 260 CPU vett részt, a rendszer tárolókapacitása pedig összesen 4816 GB. Éljünk azzal az egyszerűsítő feltevéssel, hogy minden CPU számára azonos nagyságú háttértár érhető el (esetünkben ~18,5 GB). Ha a mátrixok tördelése nélkül akarjuk megoldani a feladatot, akkor a maximális méretű rendszer, amelyet szimulálhatunk, tizenhat bites. (Lásd az erőforrásigény növekedését bemutató táblázatot.)

A fenti gondolatsor csupán a tetemes tárolási kapacitással foglalkozik, a számítás időigényével (amely a GPU- CPU-val való helyettesítésével nőne) nem.

Látható, hogy akár egy GRID-es, akár egy szuperszámítógépes megvalósítás teljesen különbözne az eredeti céloktól, hiszen egy ilyen program már nem az otthoni felhasználókat célozná meg. Tudományos felhasználás esetén nem feltétlen engedhetők meg olyan kompromisszumok, mint amilyeneket a prezentált program megkötött (nem olyan nagy teljesítményű, de cserébe nem igényel célhardvert).

b) Mire lehet még használni egy ilyen programot?

A triviális felhasználáson túl – kvantum logikai áramkörök szimulálása – egy ilyen alkalmazás alkalmas lehet további feladatok elvégzésére is.

Nagyon érdekes ötlet a már meglévő szimulációhoz interpretert készíteni, amely képes egy kvantum programozási nyelv parancsait értelmezni és futtatni, a bemutatott alkalmazást mintegy virtuális gépként alkalmazva. Természetesen egy ilyen projekt megvalósítása rengeteg további kérdést vetne fel. Az sem kizárt, hogy egy hatékony programozási nyelvvel olyan áramkörök tervezhetők, amelyek – még ilyen kevés bit szimulálása mellett is –

használhatatlanul lassan futnának le. Ugyanakkor valószínűsíthető, hogy egy interpreter megvalósítása segítené a kvantum programozási nyelvek fejlődését.

A különböző speciális architektúrákhoz kapcsolódó gondolatmenetből is kitűnik, hogy a probléma megoldása egyáltalán nem magától értetődő, hiszen még a különlegesen nagy teljesítményű gépek sem hoznak áttörést a kvantumszámítógépek szimulálásában. Ugyanakkor belátható az is, hogy hasznos lenne egy nagy kapacitású szimulációs eszköz létrehozása, arra az időre, amíg nem áll rendelkezésünkre a valódi kvantumszámítógépet.

A projekt sikeresnek tekinthető, hiszen a kitűzött célt elérte. Sikerült olyan programot írni, amelynek segítségével célhardver nélkül, de mégis gyorsan szimulálhatóak 10-12 bites kvantumrendszerek. A cél elérésében fontos szerepet játszott a grafikuskártyán való számítás végzés, hiszen videókártya nélkül a program futása meglehetősen lassú.

Irodalomjegyzék

[1.] Intel papers: Moore's law raising the bar - online például az alábbi címen érhető el: http://download.intel.com/museum/Moores_Law/Printed_Materials/Moores_Law_Backgrounder.pdf

[2.] Joan Vaccaro, Griffith University, Australia: http://www.ict.griffith.edu.au/joan/qucomp/intro.php

[3.] Moore's Law Ends in 2020, Cadence CTO predicts (video) **http://www.uberpulse.com/us/2008/02/moores_law_ends_in_2010_cadence_cto_predicts.php**

[4.] Intel 22nm 3-D Tri-Gate Transistor Technology **http://newsroom.intel.com/docs/DOC-2032**

[5.] Cloud Computing: **http://en.wikipedia.org/wiki/Cloud_computing**

[6.] GridCafe **http://www.gridcafe.org/**

[7.] Richard P. Feynman: Simulating physics with computers - elektronikusan például innen érhető el: **http://www.phy.mtu.edu/~sgowtham/PH4390/Week_02/IJTP_v21_p467_y1982.pdf**

[8.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 7.1. The basic Grover algorithm

[9.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 6.3 Order finding and factoring – Shor algorithm

[10.] Quantum chip helps crack code: http://spectrum.ieee.org/computing/hardware/chip-does-part-of-codecracking-quantum-algorithm

[11.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 2.3 QBits and qregisters

[12.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 2.6. Entanglement

[13.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 2.2 the postulates of quantum mechanics

[14.] Sándor Imre, Ferenc Balázs: Quantum Computing and Communications – An engineering approach, John Wiley & Sons. Ltd. 2005: 3. measurements

[15.] Sándor Imre: Advanced Quantum Communications – An Engineering Approach, Wiley – [előkészületben]: 2.2. Basic definitions and formulas

[16.] World record: Julich supercomputer simulates quantum computer **http://www.eurekalert.org/pub_releases/2010-03/haog-wrj033010.php**

[17.] What is GPU computing? http://www.nvidia.com/object/GPU_Computing.html

[18.] NVidia - Cuda C programming guide – elektronikusan például az alábbi címről érhető el: http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf

[19.] OpenCL Overview – elektronikusan például innen elérhető: http://gpgpu.org/wp/wp-content/uploads/2009/06/05-OpenCLIntroduction.pdf

[20.] A HunGROD project hivatalos honlapja, http://grid.kfki.hu/hungrid/services.html

Ábrák jegyzéke

I. Melléklet - A szimuláció számításigénye

kvantum bitek száma	a mátrixban lévő elemek száma	tárolandó adat mérete MB-ban	tárolandó adat mérete GB-ban	tárolandó adat mérete TB
1	4	1,52588E-05	1,49012E-08	1,45519E-11
2	16	6,10352E-05	5,96046E-08	5,82077E-11
3	64	0,000244141	2,38419E-07	2,32831E-10
4	256	0,000976563	9,53674E-07	9,31323E-10
5	1024	0,00390625	3,8147E-06	3,72529E-09
6	4096	0,015625	1,52588E-05	1,49012E-08
7	16384	0,0625	6,10352E-05	5,96046E-08
8	65536	0,25	0,000244141	2,38419E-07
9	262144	1	0,000976563	9,53674E-07
10	1048576	4	0,00390625	3,8147E-06
11	4194304	16	0,015625	1,52588E-05
12	16777216	64	0,0625	6,10352E-05
13	67108864	256	0,25	0,000244141
14	268435456	1024	1	0,000976563
15	1073741824	4096	4	0,00390625
16	4294967296	16384	16	0,015625
17	17179869184	65536	64	0,0625
18	68719476736	262144	256	0,25
19	2,74878E+11	1048576	1024	1
20	1,09951E+12	4194304	4096	4
21	4,39805E+12	16777216	16384	16
22	1,75922E+13	67108864	65536	64
23	7,03687E+13	268435456	262144	256
24	2,81475E+14	1073741824	1048576	1024
25	1,1259E+15	4294967296	4194304	4096
26	4,5036E+15	17179869184	16777216	16384
27	1,80144E+16	68719476736	67108864	65536
28	7,20576E+16	2,74878E+11	268435456	262144
29	2,8823E+17	1,09951E+12	1073741824	1048576

kvantum bitek száma	a mátrixban lévő elemek száma	tárolandó adat mérete MB-ban	tárolandó adat mérete GB-ban	tárolandó adat mérete TB
31	4,61169E+18	1,75922E+13	17179869184	16777216
32	1,84467E+19	7,03687E+13	68719476736	67108864
33	7,3787E+19	2,81475E+14	2,74878E+11	268435456
34	2,95148E+20	1,1259E+15	1,09951E+12	1073741824
35	1,18059E+21	4,5036E+15	4,39805E+12	4294967296
36	4,72237E+21	1,80144E+16	1,75922E+13	17179869184
37	1,88895E+22	7,20576E+16	7,03687E+13	68719476736
38	7,55579E+22	2,8823E+17	2,81475E+14	2,74878E+11
39	3,02231E+23	1,15292E+18	1,1259E+15	1,09951E+12
40	1,20893E+24	4,61169E+18	4,5036E+15	4,39805E+12
41	4,8357E+24	1,84467E+19	1,80144E+16	1,75922E+13
42	1,93428E+25	7,3787E+19	7,20576E+16	7,03687E+13
43	7,73713E+25	2,95148E+20	2,8823E+17	2,81475E+14
44	3,09485E+26	1,18059E+21	1,15292E+18	1,1259E+15
45	1,23794E+27	4,72237E+21	4,61169E+18	4,5036E+15
46	4,95176E+27	1,88895E+22	1,84467E+19	1,80144E+16
47	1,9807E+28	7,55579E+22	7,3787E+19	7,20576E+16
48	7,92282E+28	3,02231E+23	2,95148E+20	2,8823E+17
49	3,16913E+29	1,20893E+24	1,18059E+21	1,15292E+18
50	1,26765E+30	4,8357E+24	4,72237E+21	4,61169E+18