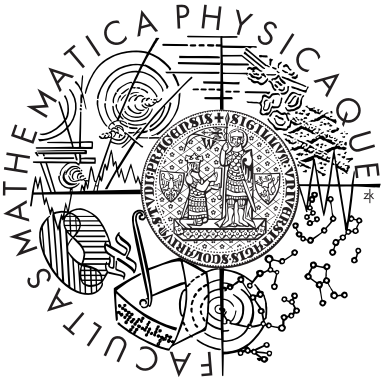
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Škoda

Kvantové algoritmy

Ústav částicové a jaderné fyziky
Vedoucí bakalářské práce: Doc. RNDr. Pavel Cejnar, Dr.

Studijní program: obecná fyzika

2007

Chtěl bych poděkovat Pavlovi Cejnarovi za trpělivou spolupráci, důkladné čtení prvních verzí práce a hlavně za obrovskou podporu.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne

Obsah

Název práce: Kvantové algoritmy
Autor: Petr Škoda
Katedra (ústav): Ústav částicové a jaderné fyziky
Vedoucí bakalářské práce: Doc. RNDr. Pavel Cejnar, Dr.
e-mail vedoucího: cejnar@ipnp.troja.mff.cuni.cz

Abstrakt: Cílem bakalářské práce je seznámit čtenáře se základními principy kvantového počítání. Na zřejmě nejznámějším kvantovém algoritmu, Shorově algoritmu na faktorizaci čísel, jsou předvedeny postupy budování kvantových algoritmů, jejich problémy a nedostatky. Přirozeným prostředkem pro zápis algoritmů jsou programovací jazyky. Jedním z jazyků pro kvantové algoritmy je Quantum Computation Language (QCL), ve kterém je jako součást práce implementován Shorův algoritmus. Kvantové algoritmy jsou v současné době ve středu pozornosti hlavně díky potenciálnímu zrychlení některých klasických algoritmů, proto se práce soustředí i na srovnání známých klasických a kvantových algoritmů.

Klíčová slova: kvantový algoritmus, Shor, QCL

Title: Quantum Algorithms
Author: Petr Škoda
Department: Institute of Particle and Nuclear Physics
Supervisor: Doc. RNDr. Pavel Cejnar, Dr.
Supervisor's e-mail address: cejnar@ipnp.troja.mff.cuni.cz

Abstract: The goal of the bachelor work is to explain the basic principles of the quantum computation. The Shor's algorithm for the number factorization, one of the best known quantum algorithms, represents a good example where the techniques of building quantum algorithms, common problems, and disadvantages can be shown. Using a programming language is the natural way how to describe an algorithm. Quantum Computation Language (QCL) is a programming language for quantum algorithms. As a part of the work, the Shor's algorithm is written in QCL. One of the reasons why the quantum algorithms are studied in present is the potencial speed up of some classical algorithms. Therefore, the work concentrates on the comparing classical and quantum algorithms.

Keywords: quantum algorithm, Shor, QCL

1 Úvod

Teorie kvantové mechaniky vznikla na počátku 20. století, aby popsala jevy jako kvantování energií emitovaných fotonů či stabilní dráhy elektronů kolem atomových jader, které klasická fyzika vysvětlit nedovedla. Během 20. století se pak rozvinula v jednu z nejdůležitějších současných teorií. Kvantová teorie ovšem vyniká nad ostatní teorie svojí neintuitivností a mnoho známých fyziků se snažilo prokázat její neúplnost. Přesto všechny dosavadní experimenty potvrzují důsledky kvantové mechaniky.

Jedním z mnoha oborů kvantové mechaniky je kvantové počítání, které se zaměřuje na využití kvantových systémů k výpočtu algoritmických úloh. Jedním z modelů pro kvantové počítání jsou kvantové obvody. Kvantový obvod se skládá z kvantových bitů (q-bitů), které jsou analogií klasických bitů, a kvantových hradel, pomocí nichž probíhá kvantový výpočet. Důležitým prvkem výpočtu v kvantovém systému je měření, pomocí kterého čteme hodnoty q-bitů.

Hlavní myšlenkou kvantového počítání je paralelizace výpočtu. Zatímco klasický počítač je vždy v jednom konkrétním stavu daném hodnotami jeho bitů, kvantový počítač může být v libovolné superpozici možných stavů. Výpočet tedy může probíhat pro více (až 2^n, kde n je počet q-bitů) hodnot najednou. Potud to vypadá jednoduše, problém ovšem nastává, jak tyto výsledky získat. Měřením stavu počítače získáme pouze jednu náhodnou hodnotu. Známé kvantové algoritmy, které jsou rychlejší než analogické klasické, pracují zcela jiným způsobem a využívají právě vlastností kvantového systému.

Práce je rozdělena do čtyř na sebe navazujících částí. V části 2 jsou popsány základní kvantová hradla a vysvětleny principy kvantového počítání. Důležitým prvkem kvantových algoritmu je diskrétní Fourierova transformace, která je popsána v části 3. V části 4 se zaměřím na Shorův algoritmus. Poslední část 5 je věnována implementaci Shorova algoritmu v jazyce Quantum Computer Language.

2 Elementy kvantového počítání

V této sekci zavedu základní prostředky kvantového počítání, konkrétně kvantové bity a kvantová hradla. Předpokládám základaní znalosti kvantových systémů například z Formánka [1]. Předkládané informace o kvantovém počítání a kvantových algoritmech jsem čerpal z knihy Nielsena a Chuanga [2].

Od kvantového bitu očekáváme, že v analogii s klasickým bitem bude nabývat dvou hodnot 0 a 1. Zatímco klasický bit je vždy pravě v jednom z těchto možných stavů, kvantový bit může být „v obou zároveň“.

2.1 Kvantové bity

Kvantový bit Q, neboli *q-bit*, je takový kvantově mechanický systém, který lze plně popsat jako superpozici dvou ortonormálních bázových vektorů, které označíme $|0\rangle$ a $|1\rangle$. Stavový prostor je Hilbertův prostor $Q = \mathbb{C}^2$ s bazí $B = \{|0\rangle, |1\rangle\}$, které říkáme *výpočetní báze*.

Q-Bit se může během výpočtu nacházet v libovolném stavu $|\psi\rangle$, který můžeme vyjádřit jako lineární kombinaci bázových vektorů

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \ . \tag{1}$$

Při měření q-bitu hodnoty $|\alpha|^2$ a $|\beta|^2$ určují pravděpodobnost nalezení q-bitu ve stavu $|0\rangle$, resp. $|1\rangle$. Proto požadujeme, aby platila normovací podmínka

$$|\alpha|^2 + |\beta|^2 = 1 \ . \tag{2}$$

V reálném světě je mnoho kvantově mechanických systémů, které se chovají jako kvantové bity. Polarizace fotonu, spin elektronu či dvouhladinový atom jsou běžné příklady.

Skupině n q-bitů říkáme *registr* R o velikosti n. Je to kvantově-mechanickým systém popsatelný Hilbertovým prostorem $R = \mathbb{C}^{2^n}$ dimenze 2^n. Jako přirozenou výpočetní bázi volíme vektory z $B_1 \times B_2 \times \cdots \times B_n$, kde B_i je báze i-tého q-bitu, a značíme je $|k_1, \ldots, k_n\rangle = |k_1\rangle|k_2\rangle \cdots |k_n\rangle$, kde $|k_i\rangle$ je bázový vektor i-tého q-bitu. Pokud posloupnost bitů $k_1, \ldots, k_n$ chápu jako binární zápis čísla $k = \sum_{i=1}^{n} k_i 2^{i-1}$, můžeme psát $|k_1, \ldots, k_n\rangle \equiv |k\rangle$. Bázových vektorů je celkem $N \equiv 2^n$ a obecný stav $|\psi\rangle$ systému lze zapsat jako jejich superpozici

$$|\psi\rangle = \sum_{k=0}^{N-1} \psi_k |k\rangle \ . \tag{3}$$

Uvědomte si, že pokud bychom chtěli tento stav uložit v počítači potřebovali bychom uložit přibližně $N \equiv 2^n$ komplexních čísel s dostatečnou přesností. V kvantovém počítači nám na to stačí n q-bitů.

Registry nemají v přírodě přirozenou reprezentaci. Důvodem je u velkých kvantových systémů velmi rychlá dekoherence, tedy interakce systému s prostředím nebo mezi částmi systému, při které systém přechází z čistého stavu do smíšeného stavu, tedy do statistické směsi čistých stavů. S tímto problémem se zatím potýkají všechny pokusy o výrobu kvantového počítače.

2.2 Kvantová hradla

Primitivní operace nad q-bity provádějí *kvantová hradla*. Kvantové hradlo je reprezentováno unitárním operátorem U, který změní kvantový stav $|\psi\rangle$ na stav $U|\psi\rangle$. Připomeňme, že unitární operátor se dá chápat jako rotace vektorů v

Hilbertově prostoru, protože se při aplikaci operátoru nemění velikost ani úhel (resp. skalární součin) svíraný zobrazenými vektory. Kvantový výpočet je pak složen z posloupnosti kvantových hradel, kterou můžeme reprezentovat jedinným operátorem $U = U_n U_{n-1} \dots U_2 U_1$, kde U_i jsou operátory příslušné jednotlivým hradlům. Nejprve si všimněme důležitého rozdílu mezi kvantovým a klasickým výpočtem: Protože operátory jsou unitární, každý výpočet na kvantovém stroji je z definice reversibilní. Nevratnost vnese do výpočtu až měření hodnot q-bitů.

Ukázalo se, že většina jednoduchých hradel lze implementovat v reálných kvantově mechanických systémech. Několik základních hradel si nyní popíšeme. Libovolný stav registru lze ve výpočetní bázi vyjádřit jako vektor z Hilbertova prostoru $\mathbb{C}^{2^n}$. Mějme obecný stav $|\psi\rangle$ daný superpozicí (3), pak příslušný vektor ψ má složky ψ_i. Hradla pak budeme reprezentovat unitárními maticemi $2^n \times 2^n$ z prostoru $\mathbb{C}^{2^n \cdot 2^n}$, kde n je počet q-bitů, na kterých hradlo pracuje. Aplikace hradla na stav systému je nyní ekvivalentní násobení vektoru stavu maticí příslušné hradlu: $U|\psi\rangle \to \mathbf{U}\psi$.

Podívejme se na základní jednobitová hradla, kterými jsou Pauliho matice:

$$\mathbf{X} \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \ \mathbf{Y} \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \ \mathbf{Z} \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{4}$$

Hradlo X je známější jako operátor bitové negace NOT, který stav $|0\rangle$ převede na stav $|1\rangle$ a naopak.

Následující tři hradla hrají velkou roli při skládání složitějších hradel.

$$\mathbf{H} \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \ \mathbf{S} \equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \ \mathbf{T} \equiv \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}. \tag{5}$$

Hadamardovo hradlo H se používá například při přípravě stavu tvořeného superpozicí všech stavů. Vezměme q-bit ve stavu $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Po aplikaci Hadamardova hradla dostaneme stav

$$\mathbf{H} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \ .$$

Pokud máme n-bitový registr v základním stavu $|0\rangle \equiv |0 \dots 0\rangle$, pak aplikací Hadamardova hradla na všechny jeho bity dostaneme stav

$$\frac{1}{2^{N/2}} \sum_{k=0}^{N-1} |k\rangle \ , \tag{6}$$

což je superpozice všech možných stavů registru, kde každý stav má stejnou pravděpodobnost.

Fázové hradlo S a $\pi/8$ hradlo T se používají například při konstrukci obvodu pro diskrétní Fourierovu transformaci.

Obrázek 1: Obvod s jedním bitem, na který je aplikováno Hadamardovo hradlo.

Obrázek 2: Obvod s CNOT hradlem. Černá tečka značí kontrolní bit, bílá tečka negovaný bit.

Je zřejmé, že pouze s jednobitovými hradly příliš zajímavých operátorů nezkonstruujeme, potřebujeme k tomu i hradla dvoubitová. Základním prvkem klasického výpočtu je větvení výpočtu pomocí běžné konstrukce `if...then...else`. K sestrojení analogického podmíněného výpočtu v kvantovém obvodu se používá hradlo controlled-NOT (CNOT). Je to dvoubitové hradlo, které provede negaci druhého bitu, pouze pokud je první bit ve stavu $|1\rangle$. CNOT lze ve výpočetní bázi popsat maticí

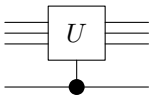
$$\mathbf{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{7}$$

Pomocí něho lze vytvořit z libovolné unitární operace U operaci controlled-U, která se provede pouze v případě, že je nastaven kontrolní bit.

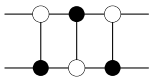
Nyní vyvstává otázka: Kolik druhů hradel potřebujeme, abychom z nich dokázali složit libovolný unitární operátor? Odpověď není vůbec lehká, Chuang a Nielsen [2] ukazují, že libovolný unitární operátor lze složit z jednobitových hradel a CNOT hradel. Pokud bychom ovšem chtěli najít pouze konečnou množinu hradel, nelze již všechny unitární operátory složit přesně, protože jich je nekonečně mnoho. Proto se přistupuje k aproximaci operátorů, kterou se zabývá celá teorie kvantových samoopravných kódů.

Kvantová hradla sestavujeme do kvantových *obvodů*. Jako u klasických obvodů jsou jednotlivé bity vedeny po „drátech“ zleva do hradla a zprava vede výstup operace. Pro obvod s jedním q-bitem a jedním jednobitovým Hadamardovým hradlem vypadá obvod jako na obrázku 1. Protože hradla pouze mění stav q-bitů, nad kterými pracují, počet vstupů a výstupů je vždy stejný. Pro controlled-NOT hradlo v obvodu máme speciální značení znázorněné na obrázku 2. Černou tečkou značíme kontrolní bit, bílou tečkou negovaný bit. Podobným způsobem značíme controlled-U operaci (viz. obrázek 3). Svazkem čar značíme, že hradlo U může pracovat na více bitech.

Prohození stavů dvou q-bitů lze provést pomocí jednoduchého obvodu *swap*, který použijeme při konstrukci obvodu pro kvantovou Fourierovu transformaci.



Obrázek 3: Obvod s obecným controlled-U hradlem. Skupina čar značí, že hradlo U může pracovat na více q-bitech.



Obrázek 4: Obvod swap, který prohazuje kvantové stavy dvou bitů.

Obvod se skladá ze tří hradel CNOT zapojených podle obrázku 4.

2.3 Asymptotická složitost

Na závěr sekce se ještě podíváme na měření efektivity algoritmů. Na klasických počítačích se zábýváme časovou složitostí algoritmů, což je funkce $T(n)$ nabývající pro každou velikost vstupu n největší počet elementárních kroků algoritmu, které provede nad nějakým vstupem o velikosti n. Protože je ale funkce $T(n)$ často složitá a nás zajímá hlavně její základní chování, zavadí se *asymptotická* složitost.

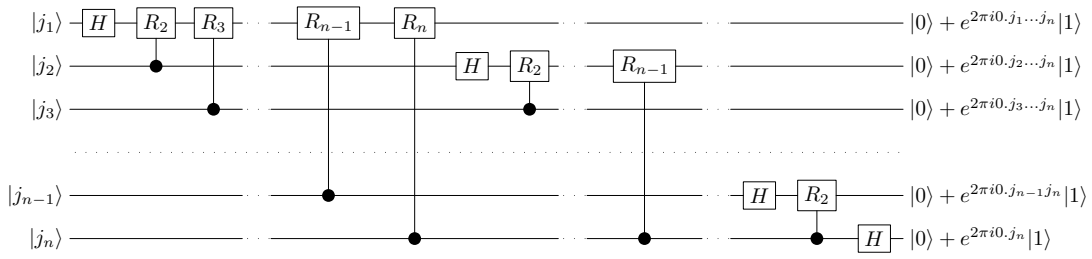
Říkáme, že $T(n)$ je *asymptoticky menší nebo rovno* $f(n)$ a píšeme $T(n) = O(f(n))$, pokud existují pevné kladné konstanty c a n_0, že pro každé n větší než n_0 je $T(n) < c \cdot f(n)$. Znamená to, že funkce $T(n)$ roste do nekonečna pomaleji nebo stejně rychle jako $f(n)$.

Říkáme, že $T(n)$ je *asympoticky rovné* $f(n)$ ($T(n)$ se chová *řádově* jako $f(n)$) a píšeme $T(n) = \Theta(f(n))$, pokud existují dvě kladné konstanty c_1, c_2, že pro $n > n_0$ platí $c_1 f(n) < T(n) < c_2 f(n)$. Jinými slovy funkce $T(n)$ roste do nekonečna stejně rychle jako $f(n)$.

Na kvantových počítačích bude elementárním krokem algoritmu jedno elementární hradlo, a proto nás bude zajímat, kolik elementárních hradel obsahují obvody, které budujeme. Podobně jako u klasických algoritmů nás bude počet hradel zajímat jen asymptoticky.

3 Kvantová Fourierova transformace

Diskrétní Fourierova transformace má vysoké uplatnění v matematice i v praxi, proto by zrychlení výpočtu transformace mělo velký přínos samo o sobě. Kvantová Fourierova transformace je základem pro několik algoritmů, mezi nimiž je i Shorův algoritmus.



Obrázek 5: Obvod pro kvantovou Fourierovu transformaci.

Diskrétní Fourierova transformace je zobrazení, které komplexnímu vektoru se složkami $x_0, \ldots, x_{N-1}$, kde N je pevná velikost vektoru, přiřadí vektor o složkách $y_0, \ldots, y_{N-1}$, kde hodnoty y_k jsou dány následovně:

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N} \tag{8}$$

Kvantová Fourierova transformace je stejná transformace, která je provedena nad vektorem reprezentujícím stav systému. Transformace obecného stavu je následující:

$$\sum_{j=0}^{N-1} x_j |j\rangle \longrightarrow \sum_{k=0}^{N-1} y_k |k\rangle \tag{9}$$

kde koeficienty y_k jsou výsledkem diskrétní Fourierovy transformace na koeficientech x_j.

Nyní jsme popsali kvantovou Fourierovu transformaci jako výsledek transformace obecného stavu. Aby měla transformace fyzikální smysl, musíme ovšem ukázat, že tato transformace je unitární operátor. Unitarita vyplyne z konstrukce obvodu pro výpočet transformace, ale dá se ukázat i přímo.

Pro konstrukci obvodu transformace se podíváme, jak působí na bázový stav $|j\rangle$. Díky principu superpozice se pak obecný stav transformuje na součet transformovaných bázových stavů.

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle \tag{10}$$

Rovnici si ještě trochu upravíme. K tomu bude užitečné pracovat se stavem $|j\rangle$ v jeho binární podobě $|j_1, \ldots, j_n\rangle \equiv |j_1\rangle|j_2\rangle \cdots |j_n\rangle$. Podobně zavedeme značení $0.j_l j_{l+1} \ldots j_m$ jako zápis binárního zlomku $j_l/2 + j_{l+1}/4 + \cdots + j_m/2^{m-l+1}$.

$$|j\rangle \longrightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} |k\rangle \tag{11}$$

$$
\begin{aligned}
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j \left(\sum_{l=1}^{n} k_l 2^{-l}\right)} |k_1, \ldots, k_n\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \\
&= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\
&= \frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \\
&= \frac{1}{2^{n/2}} \left(|0\rangle + e^{2\pi i (0.j_n)} |1\rangle \right) \left(|0\rangle + e^{2\pi i (0.j_{n-1} j_n)} |1\rangle \right) \cdots \\
&\quad \cdots \left(|0\rangle + e^{2\pi i (0.j_1 j_2 \ldots j_n)} |1\rangle \right)
\end{aligned}
$$

Nyní už vytvoříme obvod pro kvantovou Fourierovu transformaci přímočaře. Budeme k tomu potřebovat hradla R_k

$$
\mathbf{R}_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{bmatrix} \tag{12}
$$

Všimněte si, že R_1 a R_2 jsou ve skutečnosti dříve popsaná hradla S a T. Hlavní část obvodu pro kvantovou Fourierovu transformaci je znázorněna na schématu 5. Popišme, co dělá tento obvod pro vstup $|j_1, \ldots, j_n\rangle$. Aplikací Hadamardova hradla na první bit dostaneme stav

$$
\frac{1}{2^{1/2}} \left(|0\rangle + e^{2\pi i (0.j_1)} |1\rangle \right) |j_2, \ldots, j_n\rangle,
$$

protože $e^{2\pi i (0.j_1)}$ je rovno -1, pokud $j_1 = 1$ a $+1$, pokud $j_1 = 0$. Po provedení controlled-R_2 hradla, dostaneme stav

$$
\frac{1}{2^{1/2}} \left(|0\rangle + e^{2\pi i (0.j_1 j_2)} |1\rangle \right) |j_2, \ldots, j_n\rangle
$$

a po provedení controlled-R_3 až controlled-R_n hradel dokončíme generovaní stavu prvního bitu ve stavu

$$
\frac{1}{2^{1/2}} \left(|0\rangle + e^{2\pi i (0.j_1 j_2 \cdots j_n)} |1\rangle \right) |j_2, \ldots, j_n\rangle \ .
$$

Podobným způsobem získáme výsledný stav na druhém bitu

$$
\frac{1}{2^{2/2}} \left(|0\rangle + e^{2\pi i (0.j_1 \cdots j_n)} |1\rangle \right) \left(|0\rangle + e^{2\pi i (0.j_2 \cdots j_n)} |1\rangle \right) |j_3, \ldots, j_n\rangle \ .
$$

Na konci obvodu jsme získali stav

$$
\frac{1}{2^{n/2}} \left(|0\rangle + e^{2\pi i (0.j_1 \cdots j_n)} |1\rangle \right) \left(|0\rangle + e^{2\pi i (0.j_2 \cdots j_n)} |1\rangle \right) \cdots \left(|0\rangle + e^{2\pi i (0.j_n)} |1\rangle \right) \ , \tag{13}
$$

který se liší od vzorce (11) v pořadí stavů na jednotlivých bitech. Prohození stavů dvou kvantových bitů provedeme pomocí jednoduchého *swap* obvodu z předchozí sekce. Stačí nám tedy $n/2$ prohození, abychom získali stav v žádaném tvaru (11).

Všimněte si, že všechna použitá hradla jsou unitární, a proto je i kvantová Fourierova transformace unitární. Podívejme se kolik hradel tento obvod používá. Na prvním bit je aplikováno Hadamardovo hradlo a $n-1$ R_i hradel, na druhý bit o jedno hradlo méně, až na n-tý bit pouze jedno hradlo. Hlavní část obvodu používá $n+(n-1)+\cdots+1 = n(n+1)/2$ hradel. Operací swap je nejvýše $n/2$ a každá lze reprezentovat pomocí tří controlled-NOT hradel. Celkový počet hradel je tedy $\Theta(n^2)$.

Abychom dokázali zhodnotit rychlost kvantové Fourierovy transformace, srovnejme ji s klasickým počítačem. Nejrychlejší algoritmy jako je *rychlá Fourierova transformace*, která běží v čase $\Theta(n2^n)$, jsou exponenciálně pomalejší. Můžeme tedy použít kvantovou Fourierovu transformaci k počítání diskrétní Fourierovy transformace v praktických aplikacích jako je komprese dat nebo zpracování obrazu? Odpověď je bohužel záporná. Neznáme totiž žádný způsob, jak změřit amplitudy transformovaného stavu, dokonce ani nedokážeme efektivně připravit konkrétní vstupní stav. Přesto je na kvantové Fourierově transformaci postavena celá skupina algoritmů.

4 Shorův algoritmus

Kvantový algoritmus na faktorizaci čísel publikoval Peter Shor [3] ve svém článku z roku 1994, který poté vyšel znovu v roce 1997 v SIAMu. Než se dostaneme k Shorově algoritmu, budeme muset projít ještě dlouhou cestu. Nejprve si ukážeme obvod na zjištění vlastních čísel libovolného unitárního operátoru (*phase estimation*). Ten pak využijeme k vytvoření obvodu pro výpočet řádu grupy (*order-finding problem*). Ukážeme, že faktorizace se dá převést na problém nalezení řádu konkrétní multiplikativní grupy a tento řád najdeme jako fázi speciálního operátoru. Tím dostaneme známý Shorův algoritmus.

Pro vlastní číslo λ unitárního operátoru U platí $|\lambda| = 1$, což odvodíme pro vlastní vektor x operátoru U příslušný vlastnímu číslu λ. Použijeme k tomu definici unitárního operátoru $UU^* = Id$.

$$\begin{aligned} Ux &= \lambda x \\ x^*U^* &= \bar{\lambda}x^* \\ x^*U^*Ux &= |\lambda|^2 x^*x \\ x^*x &= |\lambda|^2 x^*x \\ |\lambda|^2 &= 1 \\ |\lambda| &= 1 \end{aligned}$$

Lze ukázat, že unitární operace zachovává velikost vektoru, můžeme se na ni dívat jako na otočení kolem počátku.

4.1 Určení fáze

Jak jsme právě ukázali, vlastní číslo λ unitárního operátoru je komplexní číslo o velikosti 1, a proto ho můžeme zapsat ve tvaru $\lambda = e^{2\pi i\varphi}$. Naším cílem bude zjistit hodnotu fáze φ. K tomu vytvoříme obvod se dvěmi registry. Počet q-bitů prvního *cílového* registru označíme t a závisí na přesnosti s jakou chceme fázi φ změřit. Druhý *operátorový* registr má stejný počet q-bitů jako unitární operátor U, jehož fázi měříme. První část obvodu je znázorněna na obrázku 6. Operátorový registr je na začátku ve stavu $|u\rangle$, kde $|u\rangle$ je vlastní vektor operátoru U příslušný vlastnímu číslu λ. Cílový registr je na začátku ve stavu $|0\rangle$, ale v prvním kroku ho připravíme do superpozice všech stavů pomocí Hadamardových hradel. K zápisu použijeme tvar součinu jako v sekci 3.

$$\begin{aligned} & \frac{1}{2^{t/2}} \left(\sum_{k=0}^{2^t-1} |k\rangle \right) |u\rangle \\ = \quad & \frac{1}{2^{t/2}} \left(|0\rangle + |1\rangle\right) \left(|0\rangle + |1\rangle\right) \cdots \left(|0\rangle + |1\rangle\right) |u\rangle \end{aligned} \tag{14}$$

Poté provedeme controlled-U^{2^i} operace pro $i = 0, \ldots, t-1$, kde kontrolovací q-bit je $(i+1)$-ní nejvyšší q-bit v druhém registru. Po prvním kroku ($i = 0$) bude $|0\rangle + e^{2\pi i\varphi}|1\rangle$ stav nejvyššího q-bitu cílového registru. V i-tém kroku nastavíme stav $(i+1)$-ního nejvyššího q-bitu cílového registru na $|0\rangle + e^{2^{i+1}\pi i\varphi}|1\rangle$. Po provedení všech kontrolovaných operací máme systém ve stavu

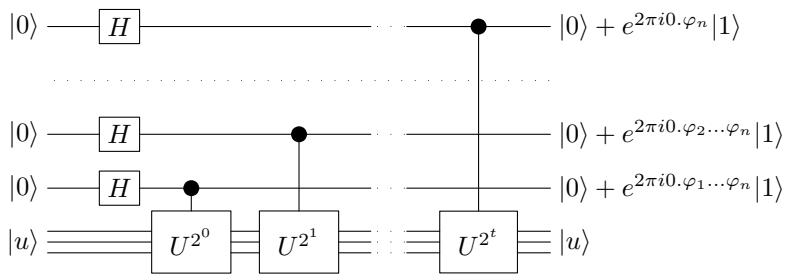
$$\begin{aligned} & \frac{1}{2^{t/2}} \left(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle\right) \left(|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle\right) |u\rangle \\ = \quad & \frac{1}{2^{t/2}} \sum_{k=0}^{2^n-1} e^{2\pi i\varphi k}|k\rangle \ . \end{aligned} \tag{15}$$

Na výsledný stav cílového registru provedeme zpětnou Fourierovu transformaci. Aproximaci hledané fáze φ získáme jako výsledek měření cílového registru.

Pro lepší představu o fungování obvodu předpokládejme, že fáze φ lze zapsat jako binární desetinné číslo o t cifrách. Použijeme stejný zápis jako v sekci 3, $\varphi = 0.\varphi_1\varphi_2\ldots\varphi_t$. Po provedení všech kontrolovaných operací máme systém ve stavu

$$\frac{1}{2^{t/2}} \left(|0\rangle + e^{2\pi i(0.\varphi_t)}|1\rangle\right) \left(|0\rangle + e^{2\pi i(0.\varphi_{t-1}\varphi_t)}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i(0.\varphi_1\ldots\varphi_t)}|1\rangle\right) |u\rangle \ . \tag{16}$$

Tento stav jsem již viděli v (11). Je to stav kvantové Fourierovy transformace vektoru o složkách φ_i. Provedeme inverzní kvantovou Fourierovu transformaci na



Obrázek 6: První část obvodu pro určení fáze vlastního čísla unitárního operátoru.

cílový registr a dostaneme stav $|\varphi\rangle|u\rangle$. Výsledkem měření operátorového registru je požadovaná fáze φ přesně.

Pokud má φ méně než t desetinných cifer, dokážeme změřit φ přesně. V obecném případě se ale musíme spokojit s tím, že s velkou pravděpodobností se změřený odhad $\widetilde{\varphi}$ nebude od skutečné fáze příliš lišit. V knize Nielsena a Chuanga [2] je dokázán následující odhad. K tomu, abychom změřili přesně φ na n bitů s pravděpodobností alespoň $1 - \varepsilon$, stačí zvolit velikost cílového registru

$$t = n + \left\lceil \log\left(2 + \frac{1}{2\varepsilon}\right)\right\rceil \ . \tag{17}$$

Zatím jsme předpokládali, že známe vlastní vektor operátoru U a hlavně že dokážeme druhý registr do vlastního stavu $|u\rangle$ připravit. My ovšem můžeme tento předpoklad obejít tím, že druhý registr připravíme do nějakého obecného stavu $|\psi\rangle = \sum_u c_u|u\rangle$. Pak, po provedení algoritmu Určení fáze, dostaneme stav $\sum c_u|\widetilde{\varphi}_u\rangle|u\rangle$. S pravděpodobností $|c_u|^2$ tedy dostaneme vlastní číslo příslušné vlastnímu vektoru $|u\rangle$.

4.2 Hledání řádu

Nyní si ukážeme si ukážeme, jak využít algoritmu Určení fáze k vyřešení problému Hledání řádu, který je zcela odlišný. Mějme dvě přirozená nesoudělná čísla x a N, $1 < x < N$. Pak *řád* x modulo N je nejmenší přirozené číslo r takové, že platí kongruentní rovnice $x^r \equiv 1 \pmod N$, kde $(\mathrm{mod}\ N)$ znamená, že výpočet probíhá nad tělesem $\mathbb{Z}_n$. Problém Hledání řádu požaduje pro dané x a N nalézt řád x modulo N.

Řád x modulo N nalezneme trikem. Najdeme speciální unitární operátor U, z jehož vlastního čísla již tento řád spočítáme. Operátor U pracuje na bázových stavech podle formule

$$U|y\rangle \ \rightarrow \ |xy \ (\mathrm{mod}\ N)\rangle \tag{18}$$

Vlastními stavy U jsou

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(\frac{-2\pi isk}{r}\right) |x^k \pmod N\rangle \tag{19}$$

pro $0 \leq s \leq r-1$, jak ukazuje následující rovnice

$$\begin{aligned} U|u_s\rangle &\equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(\frac{-2\pi isk}{r}\right) |x^{k+1} \pmod N\rangle \\ &\equiv \exp\left(\frac{-2\pi is}{r}\right) |u_s\rangle \end{aligned} \tag{20}$$

Vlastními čísly U jsou čísla $\exp(2\pi is/r)$. Použijeme algoritmus Určení fáze na operátor U jehož výsledkem bude pro vlastní stav $|u_s\rangle$ fáze s/r, ze které se nám již r podaří získat, jak ukážeme později.

Vyvstávají dva základní problémy. Prvním problémem je, jak zkonstuovat příslušný unitární operátor U a jeho mocniny. Tento problém se řeší pomocí *modulárního mocnění*, které je stručně popsáno v sekci Implementace. Druhým problémem je, jak vytvořit vlastní stav $|u_s\rangle$ operátoru U. To se zatím efektivně neumí, ale problém lze obejít jednoduchým trikem, když si všimneme, že platí

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle. \tag{21}$$

Takto můžeme připravit druhý registr do stavu $|1\rangle$ místo do vlastního stavu operátoru U. Po provedení obvodu určení fáze dostaneme v prvním registru superpozici všech vlastních čísel, ze kterých měřením dostaneme jedno náhodné.

Podívejme se nyní, jak z podílu s/r získat řád r. První podmínkou je, aby s bylo nesoudělné s r. Naštěstí tato podmínka je splněna s pravděpodobností alespoň $1/(2\log N)$, viz. [2]. Opakováním měření $2\log N$ krát dostaneme s dobrou pravděpodobností nesoudělný podíl s/r. Lze použít i chytřejších metod z [2], kterým stačí pouze konstantní počet opakování měření.

Jiným problémem je přesnost změření s/r. Změřená fáze ve tvaru $a/2^t$, kde t je počet bitů prvního registru, se od zlomku s/r trochu liší. Použijeme metodu na zjištění nejbližšího zlomku ve tvaru p/q k $a/2^t$, kde $q < N$. Jmenovatel q pak bude dobrým odhadem r, protože víme, že řád r je menší než N. Této metodě se říká *rozvoj do řetězového zlomku* (the continued fraction expansion). Podíl $a/2^t$ převedeme do tvaru řetězového zlomku

$$[a_0, \ldots, a_M] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\ldots + \frac{1}{a_M}}}} \tag{22}$$

jednoduchým rekurzivním algoritmem, který je popsán v sekci Implementace. Všimněte si, že zápis do řetězového zlomku je jednoznačný a pro každé racionální

číslo konečný. Posloupnost $[a_0, \ldots, a_k]$ značí příslušný řetězový zlomek, u kterého zapomeneme na koeficienty $a_{k+1}, \ldots a_M$. Nyní vezmeme největší k takové, aby zlomek $s/r = [a_0, \ldots, a_k]$ odpovídající prvním k členů posloupnosti měl $r < N$. Získaný řád r je nejlepším odhadem řádu a výstupem algoritmu.

4.3 Faktorizace

Faktorizace je rozklad čísla na jeho prvočinitele. Problém je nalézt k danému složenému číslu N jednoho jeho dělitele. Dosud není znám žádný klasický algoritmus, který by k tomu nepotřeboval řádově exponenciální počet kroků v počtu bitů N. Toho se využívá v moderní kryptografii a jsou na tom založeny některé protokoly pro šifrování s veřejnými klíči (např. RSA).

Pokračujme dále v převádění jednoho problému na druhý. Nyní si ukážeme, jak převést faktorizaci na hledání řádu. Označme si N faktorizované číslo. Pokud bychom měli netriviální řešení rovnice $x^2 = 1 \pmod N$, $x \neq \pm 1 \pmod N$, pak už nalezneme dělitele N, protože

$$x^2 - 1 \;\; = \;\; (x+1)(x-1) = 0 \pmod N, \tag{23}$$

kde lze předpokládat, že $1 < x < N-1$, a tedy alespoň jedno číslo z $x+1$ a $x-1$ je netriviálním dělitelem N.

Jak nalezneme takové x? Pro číslo $y < N$ nesoudělné s N z definice platí $y^r = 1 \pmod N$, kde r je řád y modulo N. Pokud je navíc r sudé a $y^{r/2} \neq \pm 1 \pmod N$, pak $x \equiv y^{r/2} \pmod N$ je námi hledané x. Číslo y můžeme vybrat náhodně. Dle [2] platí, že pokud má číslo alespoň dva různé prvočíselné dělitele, je pravděpodobnost $P[r \text{ je sudé}, y^{r/2} \neq \pm 1 \pmod N] \geq 3/4$.

4.4 Algoritmus

Shorův algoritmus se skládá z mnoha oddělených kroků, které je nutné spojit do jednoho algoritmu. Pokusím se nyní celý algoritmus zkráceně zapsat:

Faktorizace:

- Vstup: celé složené číslo N, N není mocnina prvočísla
- Výstup: dělitel čísla N
- Algoritmus:
  1. Náhodně vybereme číslo $1 < x < N$ nesoudělné s N. Pokud by bylo x soudělné s N, pak x je hledaný dělitel N a algoritmus ukončíme.
  2. Pro číslo x vyvoříme operátor $U : |y\rangle \rightarrow |yx \pmod N\rangle$ s vlastními čísly $e^{2\pi i s/r}$, příslušných vlastním vektorům $|u_s\rangle$, $s = 0, \ldots, r-1$. Použijeme

obvod určení fáze operátoru U na superpozici vlastních vektorů $|1\rangle = \sum_{s=0}^{r-1} |u_s\rangle$. Dostaneme superpozici vlastních čísel, ze kterých měřením vybereme náhodné z nich.

3. Z poměru s/r získáme odhad $\widetilde{r}$ řádu r pomocí převodu na řetězový zlomek. Tento odhad $\widetilde{r}$ je roven r, pokud s a r jsou nesoudělné a pokud byl podíl s/r změřen dostatečně přesně. V tom případě platí $x^{\widetilde{r}} = 1$.
4. Pokud je r sudé a platí $x^{r/2} \neq \pm 1 \pmod{N}$, pak získáme dělitele N jako jedno z čísel $\gcd(x^{r/2}+1, N)$ a $\gcd(x^{r/2}-1, N)$, kde gcd značí největšího společného dělitele.

5 Implementace

Pro psaní a testování programů na kvantové počítače napsal Ömer [4] jako svoji diplomovou práci programovací jazyk QCL pro kvantové počítače. Jazyk QCL je podobný programovacímu jazyku C, ale používá pouze základní konstrukce jazyka a jednoduché datové typy. Navíc definuje prostředky pro tvorbu obvodů na kvantových počítačích jako jsou kvantové registry, operátory a kvantové funkce. K jazyku QCL je k dispozici i jeho interpretr, který simuluje kvantové výpočty na klasickém počítači. Dokumentace jazyka a interpretr je veřejně ke stáhnutí na internetové adrese *http://tph.tuwien.ac.at/∼oemer/qcl.html*.

V jazyku QCL jsem implementoval Shorův algoritmus. Popíši některé důležité části programu, celý program je přiložen v dodatku.

5.1 Modulární umocňování

Jádrem algoritmu je určení fáze operátoru $U : |y\rangle \to |xy \pmod{N}\rangle$. Připomeňme jak vypadá obvod pro určení fáze ze sekce 4.1. Po připravení cílového registru do superpozice všech stavů následuje blok controlled-U^{2^i} hradel. Ovšem vytvořit hradlo pro operátor U^{2^i} je těžké, ale naštěstí to není potřeba. Řešení spočívá v *modulárním umocňování*, které vypadá následovně. Představme si, že cílový registr není v superpozici všech stavů, ale pouze v jednom konkrétním stavu $|z\rangle$. Z linearity plyne, že působení operátoru na superpozici stavů je stejné jako kdybychom aplikovali operátor na jednotlivé stavy a vzali superpozici výsledku. Provedením controlled-U^{2^i} operací na stav $|z\rangle$ dostaneme

$$\begin{aligned} |z\rangle|y\rangle &\to |z\rangle U^{z_t 2^{t-1}} \cdots U^{z_1 2^0}|y\rangle \\ &= |z\rangle|x^{z_t 2^{t-1}} \times \cdots \times x^{z_1 2^0} y \pmod{N}\rangle \\ &= |z\rangle|x^z y \pmod{N}\rangle. \end{aligned} \tag{24}$$

Proto se tomuto obvodu říká modulární umocňování. Budeme k němu potřebovat postavit aritmetická hradla. Začneme sčítáním, které použijeme na násobení a nakonec na umocňování. V dodatku jsou hradla popsána v jazyce QCL.

V kvantovém počítání se často používá chytrá myšlenka, které se říká *vratné počítání*. Mějme čtyři kvantové registry x, y, z a w, jejich hodnoty budeme značit (x, y, z, w). Chceme spočítat složitou funkci $f(x)$, kde x je kvantový registr a přitom použijeme registry y a z k pomocným výpočtům. Aby byl výpočet kvantový unitární operátor, musí být reversibilní, nemůžeme tedy hodnotu registru přepsat jinou hodnotou, protože by výpočet nebyl reversibilní. Můžeme si ale připravit pomocné registry y a z do stavu $|0\rangle$. Pak provedením výpočtu dostaneme

$$(x, 0, 0, w) \to (x, g(x), f(x), w),$$

kde $g(x)$ je výsledek nějakého pomocného výpočtu. Nyní přijde hlavní myšlenka: Binárně přičteme vypočtenou hodnotu do registru w a poté provedeme inverzní výpočet, který po sobě uklidí použité registry.

$$(x, 0, 0, w) \to (x, g(x), f(x), w \oplus f(x)) \to (x, 0, 0, w \oplus f(x))$$

Často pak pomocné registry vynecháváme a píšeme pouze $(x, w) \to (x, w \oplus f(x))$.

Jako příklad uvedeme převod kvantového mocnění na kvantové násobení.

$$|x^z \pmod N\rangle \;=\; \left(x^{z_t 2^{t-1}} \pmod N\right)\left(x^{z_{t-1}2^{t-2}} \pmod N\right)\cdots\left(x^{z_1 2^0} \pmod N\right)$$

Je-li obvod ve stavu $|z\rangle|y\rangle$, pak kvantovým vynásobením druhého registru čísly $x^{2^{i-1}} \pmod N$ za podmínky, že $z_i = 1$, dává obvod pro kvantové modulární mocnění.

5.2 Rozvoj do řetězového zlomku

Podívejme se ještě na algoritmus pro rozvoj do řetězového zlomku, který se využívá při odhadu jmenovatele zlomku. Připomeňme, že cílem je najít pro zlomek a/b a číslo m nejbližší zlomek p/q takový, že $q \leq m$. Myšlenka algoritmu je vytvářet postupně rozvoj a/b do řetězového zlomku a přitom počítat jmenovatele řetězového zlomku. Jakmile překročíme mez m vrátíme posledního jmenovatele menšího nebo rovného m.

Algoritmus bude pracovat po krocích. V k-tém kroku budeme mít zlomek a/b rozvinutý na prvních k členů řetězového zlomku. Označme x_k a y_k ještě nerozvinutou část zlomku z rozvoje

$$[a_0, \ldots, a_k](y_k/x_k) \;=\; a_0 + \cfrac{1}{\ldots + \cfrac{1}{a_k + \frac{y_k}{x_k}}}\,, \tag{25}$$

pak $a_{k+1} = \lfloor x_k/y_k \rfloor$, $x_{k+1} = y_k$ a $y_{k+1} = x_k \pmod y_k$. Nechť se dá jmenovatel zlomku $[a_0, \ldots, a_k](r_k)$ zapsat ve tvaru $p_k \cdot r_k + q_k$. Pokud je r_k ve tvaru $r_k = 1/(a_{k+1} + r_{k+1})$, pak se dá jmenovatel zlomku $[a_0, \ldots, a_{k+1}](r_{k+1})$ zapsat ve tvaru $q_k a_{k+1} + p_k + q_k \cdot r_{k+1}$ a tedy $q_{k+1} = q_k a_{k+1} + p_k$ a $p_{k+1} = q_k$. Nyní algoritmus formálně zapíšeme.

- Vstup:
  Celá čísla a, b, m
- Inicializace:
  $x_0 = a$, $y_0 = b$, $p_0 = 0$, $q_0 = 1$, $i = 1$
- Výstup:
  Jmenovatel q nejbližšího zlomku p/q ke zlomku a/b, kde $q \leq m$
- Algoritmus:
  1. pokud $x_{i-1} \bmod y_{i-1} = 0$ pak vrať q_{i-1}
  2. $x_i = y_{i-1}$
     $y_i = x_{i-1} \bmod y_{i-1}$
  3. $p_i = q_{i-1}$
     $q_i = \lfloor x_i/y_i \rfloor \cdot q_{i-1} + p_{i-1}$
  4. pokud $q_i > m$ pak vrať q_{i-1}
     jinak $i := i + 1$ a pokračuj od prvního bodu

Hodnota y_i se v každém kroku zmenší alespoň na polovinu, takže algorimtus běží v čase $O(\log b)$. Tento algoritmus je implementován ve funkci `denominator`.

6 Shrnutí

Práce se snaží popsat aparát kvantového počítání, následně postavit Shorův algoritmus a vysvětlit jeho princip. V sekci 2 jsou zavedeny kvantové bity, z nich pak sestaveny kvantové registry, představeny elementární hradla a ukázány základní obvody. Následující sekce 3 popisuje stavební prvek mnoha kvantových algoritmů, kvantovou Fourierovu transformaci. Shorův algoritmus je vybudován od základů v sekci 4. Nejprve je ukázán obvod na určení fáze, který se využije pro vytvoření obvodu na výpočet řádu grupy. Konečně se ukáže, že faktorizace se dá převést na nalezení řádu konkrétní multiplikativní grupy. Tím je Shorův algroritmus hotov a v sekci 5 se zabývám otázkami jeho implementace v jazyce QCL.

Jako součást práce jsem implementoval Shorův algoritmus v jazyce QCL. Program je přiložen v dodatku A. V dodatku B pro ilustraci procházím po krocích výpočet Shorova algoritmu pro konkrétní data a ukazuji některé výstupy programu.

Literatura

[1] J. Formánek: *Úvod do kvantové teorie I*, Academia, Praha, 2004.

[2] M. A. Nielsen, I. L. Chuang: *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[3] P. Shor: *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal of Computing 26, 1997, pp. 1484-1509.

[4] B. Ömer: *Quantum Programming in QCL*, master thesis computing science, TU Vienna, 2000.

[5] B. Ömer: *Structured Quantum Programming*, dissertation, TU Vienna, 2003.

Dodatek A

```
/* Implementation of The Shor's algorithm in QCL */

qufunct flip(qureg q)
{
  int i;
  for i = 0 to #q/2 - 1 {
    Swap(q[i], q[#q-i-1]);
  }
}

/* dft */

operator dft(qureg q)
{
  const n = #q;
  int i;
  int j;

  for i = 1 to n {
    H(q[n-i]);
    for j = i+1 to n {
      V(pi/2^(j-i), q[n-i] & q[n-j]);
    }
  }
  flip(q);
}

/* Functions */

int vector euclide(int a, int b)
{
  int vector r[3];
  int vector s[3];

  if b == 0 {
    return vector(a, 1, 0);
  } else {
    r = euclide(b, a mod b);
    s[0] = r[0];
    s[1] = r[2];
    s[2] = r[1] - (a/b)*r[2];
    return s;
  }
}

int invmod(int a, int n)
{
  int vector r[3];
  r = euclide(n, a);
  return (r[2] + n) mod n;
}
```

```
int iexpn(int a, int exp, int n)
{
  int s = a;
  int r = 1;
  int i = 0;
  int e = 1;

  while e <= exp {
    if bit(exp, i) {
      r = (r * s) mod n;
    }

    s = (s * s) mod n;
    i = i + 1;
    e = 2 * e;
  }
  return r;
}

int ilog(int x)
{
  int lg=0;
  int i=1;

  while i < x {
    lg = lg + 1;
    i = i*2;
  }
  return lg;
}

/* Operators */

cond qufunct fulladdbit(boolean b, quconst q, qureg sum, qureg carry)
{
  if b {
    CNot(carry, sum);
    Not(sum);
  }

  CNot(carry, sum & q);
  CNot(sum, q);
}

cond qufunct halfaddbit(boolean b, quconst q, qureg sum)
{
  if b {
    Not(sum);
  }

  CNot(sum, q);
}

cond qufunct sum(int b, quconst q, quvoid sum)
{
  int i;
  for i = 0 to #q - 2 {
    fulladdbit(bit(b, i), q[i], sum[i], sum[i+1]);
  }

  halfaddbit(bit(b, #q-1), q[#q-1], sum[#q-1]);
}
```

```
cond qufunct lt(int b, qureg q, quvoid flag, quvoid junk)
{
  int i;
  Not(flag);
  for i = 0 to #q-1 {
    if bit(b, i) {
      Not(q[i]);
      CNot(junk[i], flag);
      CNot(flag, q[i] & junk[i]);
    } else {
      Not(junk[i]);
      CNot(junk[i], flag);
      CNot(flag, q[i] & junk[i]);
    }
  }
}

cond qufunct summodn(int b, int n, quconst q, quvoid sum, quvoid flag)
{
  qureg junk[#q];
  qureg qq = q;
  qureg f[1];

  lt(n-b, qq, f, junk);
  CNot(flag, f);
  !lt(n-b, qq, f, junk);

  if flag {
    sum(2^#q+b-n, q, sum);
  } else {
    sum(b, q, sum);
  }

}

cond qufunct cswap(qureg a, qureg b)
{
  int i;
  for i = 0 to #a-1 {
    CNot(a[i], b[i]);
    CNot(b[i], a[i]);
    CNot(a[i], b[i]);
  }
}

cond qufunct inplacesum(int b, int n, qureg sum)
{
  qureg junk[#sum];
  qureg flag[1];

  summodn(b, n, sum, junk, flag);
  cswap(sum, junk);
  Not(flag);
  !summodn(n-b, n, sum, junk, flag);
}
```

```
cond qufunct mul(int b, int n, quconst q, qureg r)
{
  int i;

  for i = 0 to #q-1 {
    if q[i] {
      inplacesum(b*2^i mod n, n, r);
    }
  }
}

cond qufunct ipmuln(int b, int n, qureg q)
{
  qureg junk[#q];

  if gcd(b, n) > 1 {
    exit "ipmuln: b and n have to be relatively prime";
  }

  mul(b, n, q, junk);
  cswap(junk, q);
  !mul(invmod(b, n), n, q, junk);
}

/* expn: <q|<0| -> <q|<b^q mod n| */
qufunct expn(int b, int n, quconst q, quvoid r)
{
  int i;

  Not(r[0]);
  for i = 0 to #q-1 {
    if q[i] {
      ipmuln(iexpn(b, 2^i, n), n, r);
    }
  }
}

/* Computes order of x modulo N */
procedure order(int x, int N, qureg t, qureg u)
{
  int i;

  H(t);
  expn(x, N, t, u);

  !dft(t);
}
```

```
/* The continued fraction expansion */

int denominator(int a, int b, int max)
{
  int x=a;
  int y=b;
  int z;
  int q0=0;
  int q1=0;
  int q2=1;

  {
    q0 = q1;
    q1 = q2;

    z = x mod y;
    if z == 0 { break; }

    x = y;
    y = z;

    q2 = (x/y)*q1 + q0;
  } until q2 >= max;

  return q1;
}

/* computes the factor of n */
int getfactor(int m, int x, int n, int w)
{
  int y;
  int d;

  d = denominator(m, 2^w, n);

  if d mod 2 == 1 {
    if 2*d < n {
      print "The denominator is odd, multiplying by 2...";
      d = 2*d;
    } else {
      print "The denominator is odd:", d;
      return 1;
    }
  }

  print "The denominator is", d;

  if iexpn(x, d, n) != 1 {
    print "The denominator isn't order of", x;
    return 1;
  }

  y = iexpn(x, d/2, n);

  if y == 1 or y == n-1 {
    print "x^(d/2) is +- 1";
    return 1;
  }

  return max(gcd(y-1, n), gcd(y+1, n));
}
```

```
/* Shor's Algorithm */

procedure shor(int n)
{
  int w = ilog(n);
  int x;
  int y;
  int m;
  int factor;
  qureg t[2*w];
  qureg u[w];

  print "Factoring number", n;
  factor = 0;
  {
    reset;

    {
      x = floor(random()*(n-2))+2;
    } until gcd(x, n) == 1;
    print "Random x is", x;

    order(x, n, t, u);
    measure t, m;

    if m == 0 {
      print "Zero measured";
    } else {
      print "Measured:", m;
      factor = getfactor(m, x, n, 2*w);
    }

  } until factor > 1;

  print "Factor of", n, "is", factor;
}
```

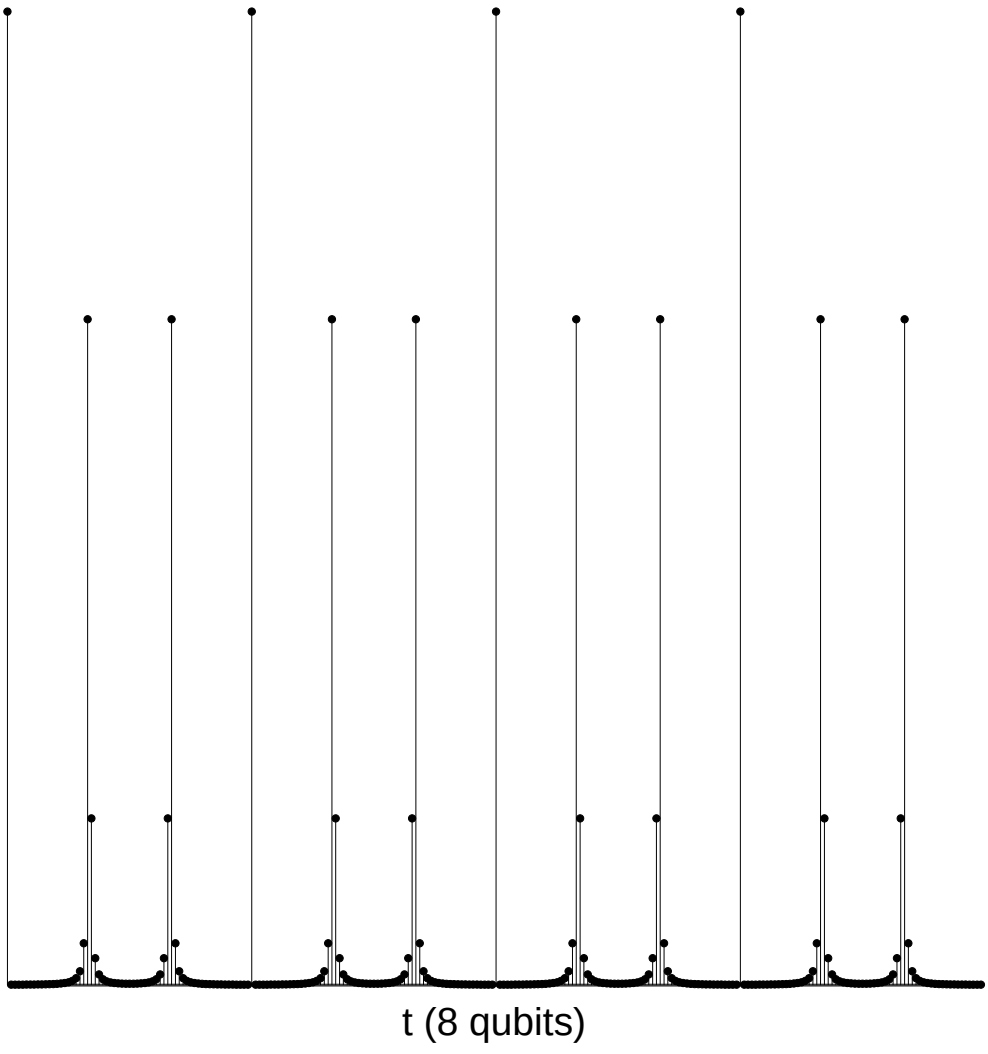
Dodatek B

Průběh Shorova algoritmu si ukážeme na příkladu. Nechť chceme najít dělitele čísla $N = 39$. Jako náhodné číslo pro hledání řádu modulo N zvolíme $x = 20$. Na zapsání čísla N potřebujeme $w = 6$ bitů a z tohoto počtu odvodíme velikost kvantových registrů. Jak jsme ukázali v sekci Implementace, hlavní část algoritmu je modulární umocňování, které vyžaduje dva registry. Operátorový registr o velikosti w na vlastní stavy operátoru $U : |y\rangle \rightarrow |yx \pmod N\rangle$, jejichž fáze určujeme, a cílový registr, který má tolik q-bitů, s jakou přesností chceme určit fázi. Pro naši potřebu zvolíme velikost $t = 8$.

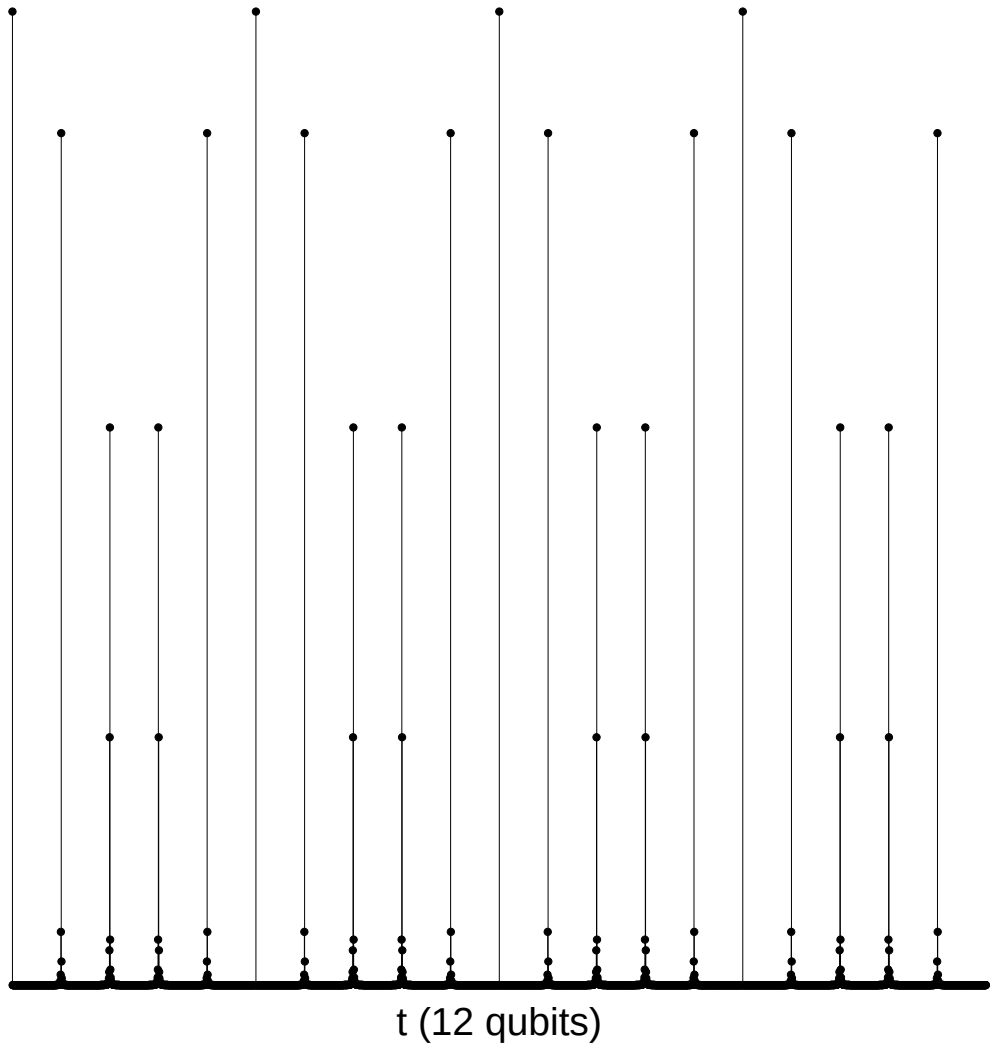
Začneme tím, že cílový registr připravíme do superpozice všech stavů a operátorový registr do stavu $|1\rangle$, což je superpozice vlastních stavů U. Poté provedeme modulární mocnění a zpětnou Fourierovu transformaci. Nyní máme v cílovém registru požadovanou fázi, ve skutečnosti superpozici všech fází vlastních čísel příslušných vlastním stavů U. Interpreter QCL nám umožňuje vynést do grafy amplitudy pravděpodobností jednotlivých stavů, využijeme toho k zobrazení amplitud pravděpodobností stavů cílového registr na obrázku 7. Amplitudy pravděpodobnosti tvoří píky, které jsou způsobeny tím, že řád x modulo N nedělí celkový počet stavů 2^t. Čím jsou píky širší, tím je větší šance, že nezměříme řád dostatečně přesně. Zvětšením cílového registru můžeme zvětšit počet změřených bitů řádu a tím zvětšit pravděpodobnost získání správného řádu, jak popisuje rovnice (17).

Změříme cílový registr a nechť dostaneme hodnotu $m = 107$, jejíž pravděpodobnost byla 5.7%. Rozvinutím do řetězového zlomku dostaneme, že zlomek $m/2^t = 107/256$ je nejblíže zlomku $5/12$ a v tomto případě opravdu dostáváme řád $r = 12$. Protože je řád sudý a $x^{r/2} = 25 \neq \pm 1 \pmod N$, alespoň jedno z čísel $x^{r/2} + 1$ a $x^{r/2} - 1$ má s N netriviálního společného dělitele. Největší společný dělitel čísel $x^{r/2} + 1$ a N je 13, což je hledaný dělitel.

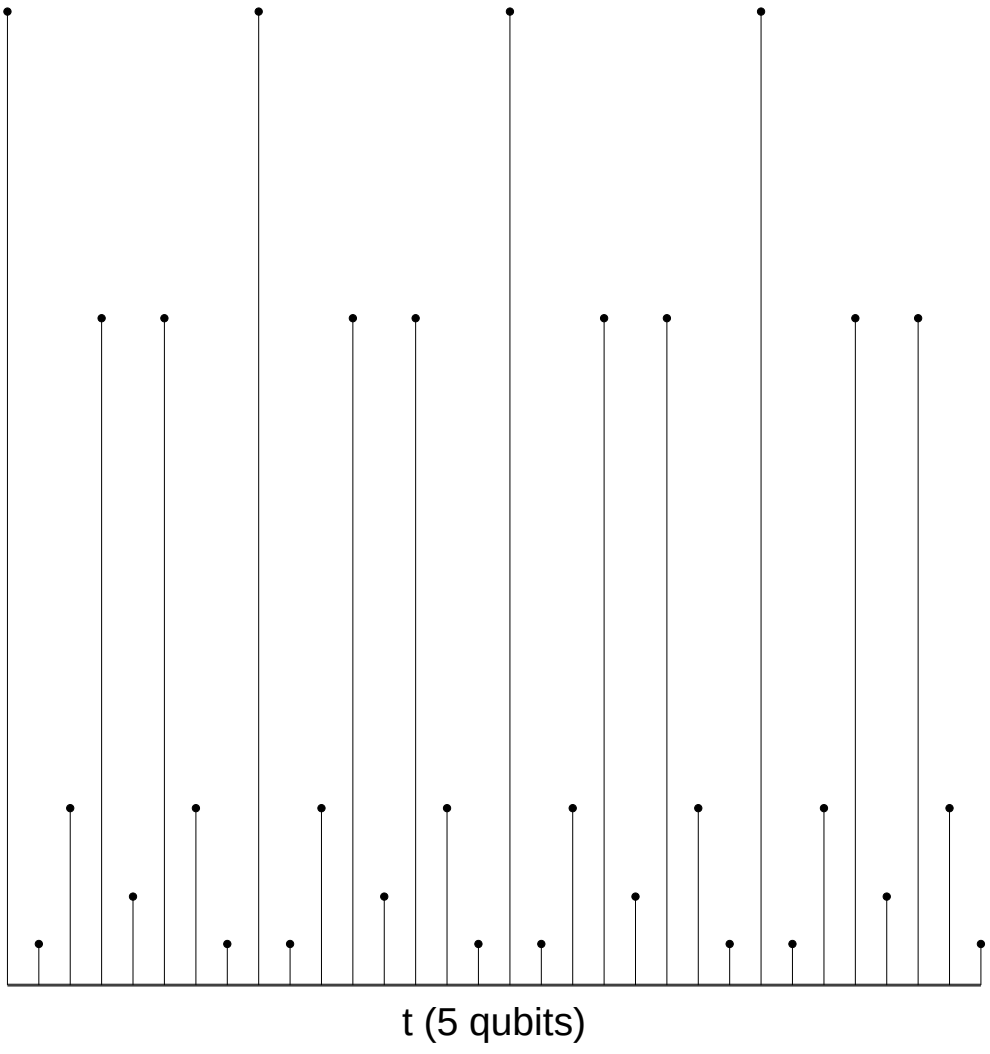
Na závěr ještě ukážeme dva grafy amplitud pravděpodobnosti cílového registru po provedení obvodu určení fáze. Na obrázku 8 má cílový registr 12 q-bitů, na obrázku 9 má pouze 5 q-bitů.



Obrázek 7: Amplitudy pravděpodobnosti cílového registru po provedení obvodu určení fáze pro $N = 39$, $x = 20$ a $t = 8$



Obrázek 8: Amplitudy pravděpodobnosti cílového registru po provedení obvodu určení fáze pro $N = 55$, $x = 51$ a $t = 12$



Obrázek 9: Amplitudy pravděpodobnosti cílového registru po provedení obvodu určení fáze pro $N = 26$, $x = 15$ a $t = 5$