Computação Quântica: Simulação e Linguagem de Programação Quântica

Luiz Guilherme Dall Acqua, Mirkos Ortiz Martins

[1]Centro Universitário Franciscano (UNIFRA)
Rua Dos Andradas – 97000-970 – Santa Maria – RS – Brazil

Abstract. *In the 1980s, came the first definitions of quantum computing as a new computing paradigm, driven by the possibility of an exponential leap in processing power and overcoming the current computing paradigm. This study aims to report the results obtained in the development of an integrated environment computations quantum simulation, which was implemented with Haskell programming language using the web platform and uses QML programming language to perform simulation of quantum algorithms.*

Resumo. *Na década de 1980, surgiram as primeiras definições sobre a computação quântica como um novo paradigma computacional, impulsionado pela possibilidade de um salto exponencial na capacidade de processamento e superação do paradigma atual de computação. Este trabalho tem como objetivo relatar os resultados obtidos no desenvolvimento de um ambiente integrado de simulação de computações quânticas, que foi implementado com linguagem de programação Haskell utilizando a plataforma web e que utiliza linguagem de programação QML para realizar simulação de algoritmos quânticos.*

1. Introdução

A computação quântica tem atraído a atenção de pesquisadores de diversas áreas impulsionados pela possibilidade de um salto exponencial na capacidade de processamento e pela superação do paradigma atual de computação. Entretanto, existem sérias dificuldades relacionadas à implementação deste paradigma computacional devido a decoerência e as limitações impostas pelas escalabilidade [Nielsen and Chuang 2010].

A decoêrencia em sistemas quânticos é compreendida pela dificuldade em manter este tipo de sistema isolado do resto do ambiente, com o intuito de evitar que a informação quântica se corrompa. Além disso, as dificuldades encontradas na escalabilidade dos sistemas quânticos, está relacionada com a demanda de qubits utilizados no processamento da informação quântica. Portanto, em razão ao que foi descrito, torna-se imperativo o uso de simuladores na computação quântica para explorar, desenvolver novos algoritmos e linguagens de programação que englobem este modelo de computação.

Os primeiros algoritmos a proporem a utilização das propriedades da mecânica quântica para efetuar computações, ficaram conhecidos como a transformada quântica de Fourier, que soluciona problemas matemáticos tranformando em algum outro problema para o qual a solução já seja conhecida que posteriormente foi utilizado por Peter Shor em seu algoritmo de fatoração [Shor 1994] e o algoritmo de buscas de Grover [Grover 1996], que tem seu foco na resolução do problema de buscas em um base de

dados desordenado. A execução destes algoritmos em simuladores quânticos, têm antecipado o conhecimento acerca de seu comportamento quando executados sobre hardware quântico real [Cabral 2004], desta forma, estimulando o aprendizado e, consequentemente, o desenvolvimento de novos algoritmos e que conforme as afirmações de [Mauerer 2005, Cabral 2004], evidenciam as potencialidades de uma máquina quântica. Estes fatores, assim contribuíram para que pesquisas em linguagens de programação para computadores quânticos se desenvolvesse consideravelmente nos última década afirmam [Vizzotto and da Rocha Costa 2006].

Ao longo dos anos, as pesquisas em linguagens de programação quântica contribuíram positivamente para que houvesse uma abstração dos formalismos matemáticos e detalhes de hardware para implementação de novos algoritmos desenvolvidos para este modelo de computação. Em virtude disso, por ainda basear-se em muitos fundamentos matemáticos da álgebra linear, a computação quântica demonstra possuir fortes conexões pelo paradigma de programação funcional por incluir noções predefinidas de paralelismo, onde estes formalismos podem ser modelados em uma linguagem funcional com alto nível de abstração do sistema real [Karczmarczuk 2003].

Tendo em vista a implementação, desenvolvimento e execução de algoritmos codificados com a linguagem QML do inglês *Quantum Meta Language*, o presente trabalho tem como objeto de estudo, desenvolver um ambiente integrado para simulação de computações quânticas projetado para a plataforma web.

Com o intuito de atingir o objetivo geral e sua complementação de acordo com as etapas consecutivas, os objetivos específicos deste estudo são: criar uma uma metodologia de desenvolvimento aplicada à programação funcional; simular computações quânticas através de algoritmos escritos em linguagem de programação quântica QML; criar um ambiente integrado para a realização de simulações utilizando o paradigma de programação funcional com a linguagem de programação Haskell para plataforma web, e apresentar o ambiente integrado de simulação como uma ferramenta inovadora no que tange o estudo e desenvolvimento das linguagens de programação quântica.

Seguindo a temática desenvolvida, este trabalho se faz importante pela inovação ao utilizar a plataforma web como ambiente voltado para simulação de sistemas complexos como o desenvolvimento de novos algoritmos e linguagens de programação quântica, o que cria um novo patamar no que tange o desenvolvimento de pesquisas na área. Desta forma, esta pesquisa surge como uma atividade diferenciada que possibilita a ampla visão sobre um ambiente integrado de simulação e colabora para o aumento do conhecimento, proporcionando novas experiencias válidas para o futuro.

2. Computação Quântica

A computação quântica pode ser compreendida como a investigação das tarefas que podem ser realizadas pelo processamento da informação, propondo o uso de efeitos da mecânica quântica. Portanto, da mesma forma que se procede na computação clássica, com o uso de bits para representar a informação, para um sistema quântico poder ser aproveitado para computação é necessária a representação desta informação através de bits quânticos, chamados qubit [Nielsen and Chuang 2010].

A palavra qubit provém do termo *quantum binaray digit* que é empregado para distinguir bits clássicos de bits quânticos. Em sistemas computacionais clássicos, um bit

corresponde a menor unidade de informação, que é representada por dígitos do sistema binário. Desta forma, um qubit pode representar qualquer sistema físico de dois níveis, que são encontrados sob diversas formas.

Um qubit é demonstrado como um vetor de estado, unitário e bidimensional, isto é, como um vetor de duas dimensões em um espaço de Hilbert $\mathcal{H}$, [1] onde qualquer escolha de base é válida desde que os vetores que a constituam sejam ortonormais[2] [Carvalho et al. 2011].

Em outras palavras, um bit quântico é qualquer sistema quântico de dois níveis, ou seja, dois possíveis estados para um qubit, que são $|0\rangle$ e $|1\rangle$. O estado de um qubit pode ser representado por uma combinação linear de estados, que pode ser visualizada na equação 1, esta é chamada de superposição coerente [ALVES 2003], ou ainda pela representação matricial para os vetores que definem o espaço vetorial de um qubit, esta segunda forma é apresentada na equação 2.

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

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} e\, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2}$$

Os qubits podem ser implementados fisicamente e foram exaustivamente validados por experimentos de sistemas físicos, exemplos destes experimentos são o alinhamento de um *spin* nuclear em um campo magnético uniforme e os dois estados de um elétron orbitando ao redor de um átomo [Nielsen and Chuang 2010].

Os sistemas quânticos possuem detalhes e formalismos matemáticos que descrevem suas características e que, muitas vezes, poderiam ser abstraídos, criando um conjunto de propriedades encontradas somente em sistemas físicos quânticos, tornando seu entendimento menos complexo.

Alguns exemplos das propriedades citadas acima são: o emaranhamento, que descreve que os sistemas quânticos quando combinados possuem a característica de não poderem mais ser descritos como dois espaços Hilbert, mas sim, como um único espaço maior [de Lima Marquezino 2006]; a superposição, que é compreendida pela propriedade que os qubits possuem de registrar ao mesmo tempo diferentes valores [ALVES 2003]; e o paralelismo, que é uma consequência direta da superposição dos estados de um sistema quântico. Conforme o sistema evolui através de operações unitárias, todos os componentes superpostos do estado em questão são modificados ao mesmo tempo, como se fossem argumentos individuais para a função computada pela operação [de Lima Marquezino 2006, ALVES 2003].

O desenvolvimento das linguagens de programação quântica é motivado pelas abstrações que modelam características como o emaranhamento, superposição e para-

[1] Um espaço de Hilbert é um espaço vetorial $\mathcal{H}$ dotado de produto interno, ou seja, com noções de distância e ângulos, de dimensão finita ou infinita.

[2] Na álgebra linear, o conceito de ortonormalidade define que um conjunto de vetores em um espaço com produto interno é ortonormal quando todos vetores do conjunto são vetores unitários, sendo assim, com norma igual a 1, ou que o produto interno de dois vetores distintos seja zero sendo assim, cada par de vetor é ortogonal

lelismo quântico, além de permitirem a criação de um *framework*[3] para a especificação formal de operações quânticas e sua execução, favorecendo sua visualização e análise antes mesmo de sua implementação em um hardware físico [Selinger 2004].

Em seu trabalho sobre linguagens de programação quântica, [Selinger 2004] descreve o modelo conceitual de circuito quântico como sendo um dos principais para a execução de algoritmos escritos em linguagem de programação quântica. Este modelo é análogo ao modelo de circuitos clássicos, que são compostos por um arranjo de portas lógicas, pois os circuitos quânticos utilizam portas quânticas que, além de serem sempre reversíveis, também correspondem às transformações unitárias em um espaço vetorial complexo. Na sequência de execução das operações, ou seja, durante a execução do algoritmo quântico, as operações de medida são deixadas sempre como o último passo da computação, preservando suas propriedades quânticas. Para que se possa compreender melhor, a Figura 1 ilustra este modelo.

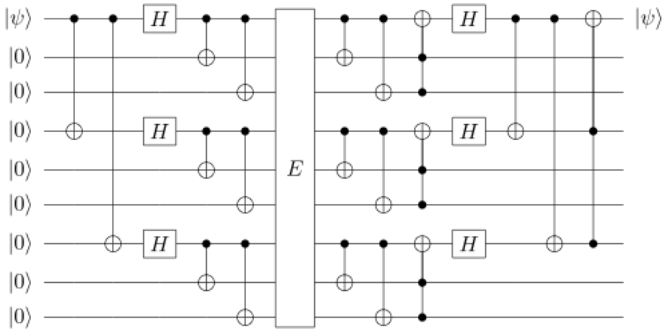

Figura 1. Exemplo do modelo de circuíto quântico. Fonte: adaptado de [Nielsen and Chuang 2010]

2.1. Algoritmos Quânticos

Um algoritmo é um procedimento contendo regras para a execução de uma tarefa específica [Nielsen and Chuang 2010]. Quando um computador executa operações aritméticas como, por exemplo, a operação de soma entre dois números, o computador executa um algoritmo, portanto os algoritmos são conceitos-chave na computação.

Os algoritmos quânticos são aqueles que fazem uso de algumas das características da mecânica quântica para computar uma tarefa. O estudo destes algoritmos teve início apenas na década de 1980 com resultados comparativos entre máquinas de Turing determinísticas, probabilísticas e não-determinística, afirma [Oliveira 2007]. Entretanto, [Nielsen and Chuang 2010] afirmam que atualmente existem três classes de algoritmos quânticos que fornecem vantagem real sobre os algoritmos clássicos conhecidos.

A primeira classe de algoritmos quânticos citada, é baseada na versão quântica da tranformada de Fourier, introduzida no ano de 1994 por Peter Shor [Shor 1994], que criou um algoritmo polinomial para fatorar números grandes. Além do algoritmo de Shor, também são incluídos nesta classe algorítimos para resolver logaritmos discretos

[3]Compreende-se por *framework* como uma abstração de alto nível que une funções que são de uso frequente com a finalidade de prover uma funcionalidade genérica.

[Nielsen and Chuang 2010]. Tendo em vista que a segurança computacional atual é embasada na dificuldade de fatoração, esta classe algorítmica vem despertando o interesse de vários pesquisadores.

A segunda classe de algoritmos é representada pelos algoritmos de busca, introduzidos em 1996 por Grover [Grover 1996] e é utilizada para realizar a busca de elementos dentro de um conjunto desordenado, apresentando assim um ganho quadrático mais eficiente sobre o algoritmo clássico. A terceira e última classe citada, corresponde aos algoritmos de simulação, por meio dos quais um computador quântico é utilizado para simular um sistema quântico.

A seguir, a Figura 2 representa o estado da arte sobre algoritmos quânticos conhecidos até o momento, incluido alguns exemplos de aplicações. Naturalmente, no centro do diagrama, está a transformada de Fourier quântica e o algoritmo quântico de busca de Grover [Nielsen and Chuang 2010]. Particularmente interessante é o algoritmo quântico de contagem, pois trata-se de uma combinação inteligente da transformada de Fourier quântica e do algoritmo quântico de busca de Grover. Este algoritmo pode ser usado para estimar o número de soluções de um problema de busca de forma mais rápida do que é possível, quando comparado a um computador clássico.

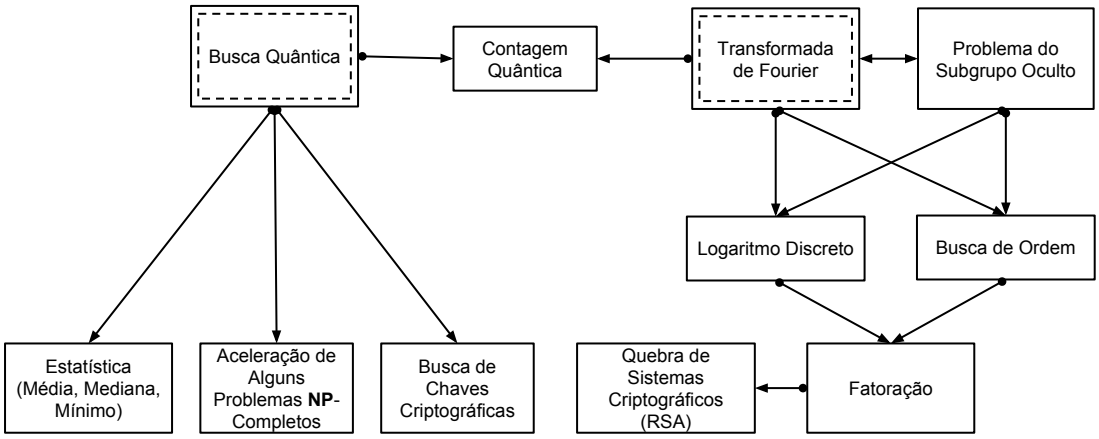

Figura 2. Principais algoritmos quânticos e suas relações, incluindo algumas aplicações importantes. Fonte: adaptado de [Nielsen and Chuang 2010]

2.2. Linguagens de Programação Quântica

As linguagens de programação buscam criar paradigmas ou abstrações de alto nível, para que se possa pensar em soluções para problemas sem haver a preocupação por detalhes de baixo nível. Assim, as linguagens de programação quântica, são uma tentativa de abstrair os formalismos matemáticos da mecânica quântica, proporcionando meios adequados e necessários para descrever algoritmos quânticos complexos.

As linguangens de programação quântica, são semelhantes às linguagens de programação clássica [Mauerer 2005], diferindo-se apenas por apresentarem outras estruturas sintáticas e semânticas necessárias por consequência de características provenientes da mecânica quântica aplicada a computação, entretanto, neste contexto [Sofge 2008],

afirma que elas podem ser classificadas por paradigmas de programação: imperativo, funcional e outros que majoritariamente incluem formalismos matemáticos não destinados à execução em um computador.

Desta forma, apresenta-se com maior enfoque o paradigma de programação funcional que, de acordo com [Karczmarczuk 2003, Sabry 2003], possui fortes conexões com a computação quântica e, pela existência de investigação, por parte da comunidade científica, sobre a implementação de linguagens quântica em dialeto[4] Haskell.

2.2.1. O paradigma de programação funcional

A programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas. [Sebesta 2003] define que estas funções são mapeamentos nomeados e não nomeados, que usam apenas expressões condicionais e recursão para controlar suas avaliações. Em outras palavras, [Baranauskas 1993] ressalta que estas funções podem também ser definidas por um mapeamento de membros de um domínio em um contra domínio. Neste paradigma, um programa é composto por um conjunto de funções e todas as operações são executadas por funções que tomam como parâmetros outras funções. Entretanto, [Sebesta 2003] afirma que em linguagens de programação que se classificam como puramente funcionais, não existem variáveis ou comandos de atribuição. A ordem de avaliação de suas expressões de mapeamento é controlada por recursão e expressões condicionais.

Em razão disso, o paradigma de programação funcional demonstra possuir fortes conexões com a computação quântica porque ela inclui uma noção predefinida de paralelismo, mesmo que essa noção seja qualitativamente diferente da noção encontrada na programação funcional e está baseada em fundamentos matemáticos, como: espaços vetoriais, álgebra de matrizes, entre outros, que podem ser modelados elegantemente em uma linguagem funcional.

2.2.2. A linguagem de programação Haskell

Haskell, de acordo com [Peyton and Hughes 1997, Thompson 1999], é uma linguagem funcional de propósito geral e puramente funcional, isto é, não utiliza variáveis e nem atribuições de sentença, de semântica não estrita [5]e de tipagem forte, que incorpora muitas inovações recentes no que tangencia o projeto e concepção de linguagens de programação, representando assim, a solidificação de anos de pesquisa em linguagens funcionais.

A linguagem Haskell, também possui implementado junto ao seu *kernel*, funções de alta ordem, tipos algébricos definidos pelo usuário, suporte estático aos polimorfismos paramétrico e *ad hoc*, casamento de padrões, compreensão de listas, sistema de módulos, monadas, e um rico conjunto de tipos primitivos, incluindo *arrays*, inteiros de precisão

[4]Os dialetos ordinariamente encontrados nas linguagens que implementam o paradigma programação funcional, podem ser compreendidos como extensões que incorporam recursos extras, tais como estruturas de dados especiais ou modificação da sintaxe, sem alterar a natureza intrínseca da linguagem.

[5]Em uma avaliação não estrita, os argumentos para uma determinada função não são avaliados a menos que sejam utilizados na avaliação do corpo da função.

fixa e arbitrária e números de ponto flutuante [de Carvalho Junior 2000].

2.2.3. *Quantum Meta Language*

A linguagem *Quantum Meta Language* ou QML, introduzida por [Altenkirch and Grattage 2005], aborda o paradigma da programação funcional de primeira ordem, ou seja, nesta forma de programação, as funções são tratadas como um tipo de dado qualquer que permite-as servir como parâmetros de entrada, bem como parâmetros de retorno para outras funções, além de apresentar recursos em sua estrutura tanto para dados clássicos, quanto para dados quânticos.

É uma linguagem pioneira, quando observada pelo fato de ter introduzido primitivas para controle condicional quântico, chamado if^0. Conforme [Vizzotto and da Rocha Costa 2006], esta primitiva condicional consegue analisar o qubit sem aplicar uma medida sobre ele, conservando seu estado quântico. É possível observar na Figura 3 a simplicidade e a clareza na codificação do algoritmo de Deutsch e no algoritmo de Teletransporte Quântico. Além disso, a linguagem QML utiliza como base a arquitetura conceitual da máquina de Turing quântica proposta por [Deutsch 1985, Knill 1996].

```
-- Algoritmo de Deutsch

-- Operador de Hadmard
Had (b,qb) |- ifo b then hF * qfalse + (-hF) * qtrue
              |       else hF * qfalse + hF * qtrue :: qb;

Deutsch (a,qb) (b,qb)  |-
    let  (x,y) = ifo   (hF * qtrue , hF * qfalse)
                 then  (qtrue,   ifo  a
                                then  ((qfalse,-qtrue),(qtrue,b))
                                else  ((-qfalse,qtrue),(qfalse,b))
                       )
                 else  (qfalse,  ifo  b
                                then  ((-qfalse,qtrue),(a,qtrue))
                                else  ((qfalse,-qtrue),(a,qfalse))
           in Had (x) :: qb;

DeutschC |- Deutsch ((qtrue),(qfalse)) :: qb;
DeutschT |- Deutsch ((qtrue),(qtrue)) :: qb;
DeutschF |- Deutsch (qfalse,qfalse) :: qb;
DeutschHT |- Deutsch ((Had(qtrue)),(Had(qtrue))):: qb;
DeutschHF |- Deutsch ((Had(qfalse)),(Had(qfalse))):: qb
```

```
-- O algoritmo de Teleporte Quântico de:
-- "A lambda calculus for quantum computation
-- with classical control" (Selinger & Valiron,2005)
-- Operador de Hadmard
Had (b,qb) |- ifo b then hF * qfalse + (-hF) * qtrue
                    else hF * qfalse + hF * qtrue :: qb;
-- Operador QNot
Qnot (b,qb) |- ifo b then qfalse else qtrue :: qb;
-- O Operador CNot
CNot (s,qb) (t,qb) |- ifo s then (qtrue,Qnot (t))
                           else (qfalse,t) :: qb*qb;
-- O Operador de medição usando "if"
Meas (x,qb) |- if x then qtrue else qfalse :: qb;
-- A Constante par EPR
Epr |- hF * (qtrue,qtrue) + hF * (qfalse,qfalse) :: qb*qb;

-- O operador de medição
Bmeas (x,qb) (y,qb) |- let (xa,ya) = CNot (x,y)
                       in  (Meas (Had (xa)),Meas (ya)) :: qb*qb;
Bnmeas (x,qb) (y,qb) |- let (xa,ya) = CNot (x,y)
                        in  (Had (xa),ya) :: qb*qb;
-- A correção das operações
Uol (x,qb) |- ifo x then  qfalse else qtrue  :: qb;
Ulo (x,qb) |- ifo x then -qtrue  else qfalse :: qb;
Ull (x,qb) |- ifo x then -qfalse else qtrue  :: qb;
U (q,qb) (xy,qb*qb) |- let (x,y) = xy
                       in if x then (if y then Ull (q) else Ulo (q))
                               else (if y then Uol (q) else q) :: qb;
Tele (a,qb) |- let (b,c) = Epr ()
               in  let f = Bnmeas (a,b)
                   in U (c,f) :: qb;

TeleT  |- Tele (qtrue) :: qb;
TeleF  |- Tele (qfalse) :: qb;
TeleHT |- Tele (Had (qtrue)) ::qb;
TeleHF |- Tele (Had (qfalse)) ::qb
```

Figura 3. Exemplo de codificação com a linguagem QML. Ao lado equerdo, o algoritmo Deutsch e ao lado direito o algoritmo do Teletransporte Quântico

Na máquina de Turing quântica as operações de leitura, escrita, bem como os movimentos da cabeça são definidos através de interações quânticas. Pela explanação de [Knill 1996] compreende-se que a fita e a cabeça existem em estados quânticos. Em particular, enquanto uma célula da fita da máquina de Turing podia carregar ou 0 ou 1, agora, uma célula da fita da máquina de Turing quântica pode carregar um qubit, o qual pode estar com o sistema binário em superposição.

A Figura 4 detalha a máquina de Turing quântica e sua evolução, onde a fita à direita representa a condição inicial que, com a evolução da máquina, a cabeça se move simultaneamente em três direções diferentes e o estado da máquina se torna uma superposição dos três estados ilustrados à direita da fita inicial.

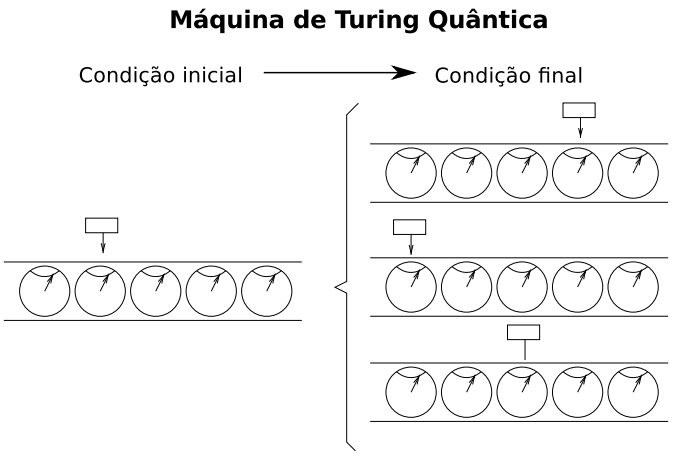

Figura 4. Representação gráfica da máquina de Turing quântica e sua evolução. Fonte: adaptado de [Vizzotto and da Rocha Costa 2006]

Através dos recursos provenientes da linguagem QML, isto é, das bibliotecas que fazem parte do contexto do compilador, além de realizar simulações de computações quânticas, é possível gerar especificações de circuitos em consequência da forma com que linguagem foi implementada.

2.3. Simulação

Uma das aplicações práticas mais importantes da computação é a simulação de sistemas físicos [Nielsen and Chuang 2010]. As técnicas de simulação são largamente utilizadas com o intuito de analisar o comportamento de sistemas, sem considerar sua natureza. Como em diversas ocasiões existem fatores que impossibilitam a construção de um sistema físico, seja pela dificuldade de manuseio ou exigência de avançados recursos tecnológicos que não estejam disponíveis, como é o caso de um computador quântico, faz-se o uso da simulação para prever o comportamento do sistema em questão.

De forma pragmática, pode-se definir o processo de modelagem e simulação como sendo uma experimentação computacional de alto nível, onde usa-se modelos de um sistema real ou idealizado para a investigação de problemas reais e de natureza complexa, com o objetivo de averiguar diferentes alternativas operacionais a fim de encontrar e propor melhores formas de operação que visem à otimização do sistema como um todo.

Partindo desta ideia sobre modelagem e simulação, um simulador de computadores quânticos pode ajudar a investigar e analisar resultados provenientes de experimentações computacionais, seja como ferramenta didática ou como auxílio ao pesquisador durante o processo de desenvolvimento de um novo algoritmo. Pode também, servir como ferramenta para verificar o desempenho de um algoritmo não somente em situações ideais, mas também na presença de erros e decoerência [Mauerer 2005].

Computadores quânticos podem simular sistemas quânticos para os quais não se conhecem simulações clássicas como anteriormente previsto por Richard Feynman[Feynman 1982]. Em virtude da simulação oferecer vantagens para áreas onde não é possível implementar o sistema real, a simulação torna-se uma importante ferramenta, conforme afirma [Barbosa 2007]. Sendo assim, a simulação possui um papel significativo para todas as áreas da ciência, seja este no estudo ou no desenvolvimento de

determinada área.

O uso da simulação acarreta um grande número de vantagens, o melhor exemplo dos benefícios obtidos com a simulação é na velocidade de obtenção das simplificações. Os simuladores são aplicados em diversos contextos, desde ferramentas voltadas para apoio ao ensino até ferramentas com finalidade bélica. Para o ensino de matérias relacionadas à computação, existem ferramentas para simulação de autômatos finitos e máquinas de Turing [Cabral 2004]. Um exemplo de ferramenta para simulação de circuitos quânticos foi proposto por [Cabral 2004] e é denominada Zeno.

Além disso, a simulação quântica permite a repetição do experimento com valores diferentes e teste de situações de limites sem que nenhum equipamento seja danificado ou precise de manutenção. Não menos importante, é a utilidade que a simulação tem no contexto do aprendizado teórico sobre o sistema que está sendo simulado. Os simuladores, por permitirem repetição dos testes e uma gama de distintos valores, permitem a compreensão a baixo nível do funcionamento do sistema.

3. Metodologia

Por não existir metodologia apropriada para o desenvolvimento de software implementado sobre o paradigma de programação funcional, esta pesquisa propôs desenvolver uma metodologia que abrangesse este paradigma, onde realizou-se a extração de artefatos funcionais para a documentação e implementação do Ambiente Integrado de Simulação, assim, esta metodologia decorreu em três etapas distintas: (A) realizou-se a revisão bibliográfica referente a teoria da computação quântica, onde, estudou-se as necessidades para que o projeto do ambiente integrado de simulação pudesse ser implementado; (B) Definiu-se as prioridades do projeto e iniciou-se o desenvolvimento do software; (C) Elaborou-se rotinas de teste para avaliar se as funcionalidades implementadas estariam de acordo com as necessidades descritas na primeira etapa; (D) Implementou-se algoritmos codificados com a linguagem QML com o intuito de verificar a plataforma desenvolvida.

A revisão bibliográfica deste trabalho refere-se a teoria da computação quântica, onde decorreu-se através de pesquisas em periódicos, livros, artigos e páginas da internet para elucidar os conceitos teóricos fundamentais para a compreenção do funcionamento dos algoritmos quânticos

Considerando o fato de não existir metodologia apropriada para o processo de desenvolvimento de software implementado sobre o paradigma de programação funcional, criou-se uma metodologia própria com base na metodologia de desenvolvimento iterativo e incremental seguindo os seguintes passos: (A) Definiu-se os requisitos do software; (B) Com os requisitos definidos, implementou-se o módulo principal do software; (C) com a base do software em funcionamento partiu para a adaptação do código fonte do compilador da linguagem QML; (D) Com as adaptações necessárias realizadas no código fonte do compilador, buscou-se novas referências para a implantação de rotinas de teste usando o compilador; (F) com o conhecimento adquirido, modularizou-se o código fonte.

Além disso, como o objetivo deste trabalho é a utilização da linguagem Haskell, optou-se pela utilização do *framework Yesod* em razão de estruturar de forma organizada os arquivos, gerenciar dependências do Haskell atráves do *sandbox* do *cabal* e, principalmente, trabalhar com o código modularizado, que proporcionou a acoplagem de novos módulos para o software.

Em virtude do ambiente integrado de simulação, cada interação de desenvolvimento tornou-se cada vez mais complexa, desta forma surgiu a necessidade da elaboração de rotinas de testes automáticas para avaliar se o software atendia os requisitos definidos.

4. Implementação

Neste Seção, serão abordados os aspectos técnicos do desenvolvimento do ambiente integrado de simulação. Na Subseção 4.1 é descrito o processo de implementação do software, onde são detalhados os principais módulos da ferramenta. Seguindo, na Subseção 4.2 descreve-se como é o funcionamento do software. Na sequência, fundamentando a Subseção anterior, descreve-se a arquitetura do software. Para finalizar, são descritos os aspectos da interface gráfica do usuário na Subseção 4.4.

As ferramentas descritas a seguir foram utilizadas no desenvolvimento do ambiente integrado de simulação. Utilizou-se o Vim como IDE (Ambiente Integrado de Desenvolvimento) para trabalhar com o código fonte do software, integrado a ele utilizou-se o Git como software responsável pelo versionamento do código fonte. Para visualização da interface do software, utilizou-se o navegador de internet *Chromium*.

4.1. Desenvolvimento

A proposta deste trabalho baseia-se no uso da linguagem de programação funcional Haskell, em razão dos recursos oferecidos por esta linguagem para o desenvolvimento de uma ferramenta para a plataforma *web* e no uso da Linguagem QML, para a codificação e simulação de algoritmos quânticos, conforme descrito anteriormente na Subseção 2.2.2.

Esta ferramenta tem por requisito fornecer para o usuário um ambiente com as condições necessárias para o desenvolvimento de algoritmos quânticos que sejam codificados com a linguagem QML, além de ter seu código fonte segmentado em módulos, a fim de que se possa estender em novas funcionalidades e ofereça suporte ao escalonamento dinâmico, quando houver a necessidade de processamento em virtude de sua operação. Em consequência disso, para atender estes requisitos, optou-se pelo uso de *framework*.

O desenvolvimento de software, com o auxilio de algum *framework*, torna-se uma garantia para que o desenvolvedor esteja com a estrutura do software correta e de acordo com suas regras de negócio, possuindo assim, fácil manutenibilidade. Sendo assim, os três principais *frameworks* existentes para a plataforma *web* codificados com a linguagem Haskell, que estão disponíveis para uso são: *Snap*, *Happstack* e o *Yesod*.

Estes três *frameworks* foram analisados para averiguar qual teria melhor aptidão para o desenvolvimento do software proposto. Através dos requisitos iniciais, para o desenvolvimento do software, o *framework* que melhor atendeu às necessidades foi o *Yesod*. Este *framework* possui rotinas e métodos para a construção de testes automatizados, é fortemente estruturado, possui configuração fácil e é integrável com qualquer servidor *web* que acople *sockets* do sistema operacional.

O ambiente integrado de simulação por ter sua plataforma *web*, exigiu que se integrasse o software com um servidor de arquivos estáticos sobre o protocolo HTTP. Em virtude de existir uma ampla variedade de servidores web, dentre os mais destacados atualmente no mercado estão: *Apache*, *Light HTTP Server*, *Nginx* entre tantos outros. Desta forma, definiu-se alguns critérios para a escolha do servidor que melhor atendesse aos objetivos deste projeto.

Entre os principais critérios utilizados na escolha do servidor *web* que atendesse os requisitos do ambiente integrado de simulação foram: o consumo dos recursos computacionais em virtude da demanda de processamento em determinadas circunstâncias de simulação, a agilidade para escalonar serviços dinamicamente dependentes do cenário proposto para a simulação e a configurabilidade fácil.

Em razão destes fatos, optou-se pelo uso do *Nginx*, por ser facilmente configurável, por ter o menor consumo de recursos computacionais em relação a seus pares e por ter a facilidade de integrar-se com qualquer tipo de *socket* ou serviço compatível com padrão *POSIX* de sistemas operacionais do tipo *UNIX*, além de atender plenamente os critérios citados anteriormente.

Para que a simulação de algoritmos quânticos codificados com a linguagem QML pudesse ser realizada dentro da plataforma que foi desenvolvida e ainda pudesse aproveitar todos os recursos que esta linguagem fornece, foi necessária uma modificação no código fonte do compilador da linguagem QML, prposto por[Altenkirch and Grattage 2005], para acoplá-lo junto ao *framework Yesod*.

Naturalmente os programas escritos com a linguagem QML são interpretados através do console interativo do Haskell, onde o compilador da linguagem é carregado em memória. Entretanto, para executar um programa escrito em QML é necessário, primordialmente tê-lo em arquivo, para que se possa usá-lo como parâmetro de entrada para as funções do compilador da linguagem.

Em virtude da forma que são executados os programas escritos com a linguagem QML e tendo em vista a utilização de todos os recursos provenientes desta linguagem de programação, como por exemplo a compilação de código fonte em circuitos, modificou-se o código fonte do compilador, onde os parâmetros de entrada das funções 'RunS' e 'RunI' que são responsáveis pela entrada de código fonte, foram modificados para aceitar, em vez de um arquivo, os dados provenientes da interface. Além disto, alterou-se também o cabeçalho dos arquivos, que foram adicionados os parâmetros.

No que tangenciou o desenvolvimento da interface gráfica com o usuário, levou-se em consideração que o sistema é composto em uma única tela por quatro painéis distintos, contendo um painel para a entrada do código fonte de QML, um painel para saída em console dos resultados, um para demonstrar o circuíto gerado pelo algoritmo escrito e outro painel, para demonstrar, através de gráficos, o consumo de recursos computacionais em relação ao tempo de execução.

Os painéis da interface gráfica, além da utilização dos recursos provenientes do HTML5, foram usados recursos dos *plugins Ace* e *Highcharts*, em que ambos são escritos em *JavaScript* para *framework JQuery*. O *plugin Ace* habilita no primeiro painel a identação da sintaxe no editor do código fonte e o outro *plugin* fornece os recursos necessários para a construção de gráficos relativos ao tempo de processamento.

4.2. Funcionamento

O funcionamento deste software tem por base receber um código fonte que o usuário tenha implementado, tratar este código a fim de verificar a existência de erros de sintaxe e enviar para o compilador da linguagem. Com o compilador recebendo o código fonte, este executa o código fonte e retorna algum resultado para o usuário.

Anteriormente, no Seção 4.1, descreveu-se que o compilador da linguagem QML está acoplado neste software, logo, quando executam-se as operações de simulação ou de execução do código fonte QML, realizam-se as chamadas funções ”RunI”e ”RunS’”, passando como argumentos o código fonte proveniente da interface gráfica e a função principal. A Figura 5 ilustra como isso ocorre exemplificando com um trecho do código fonte.

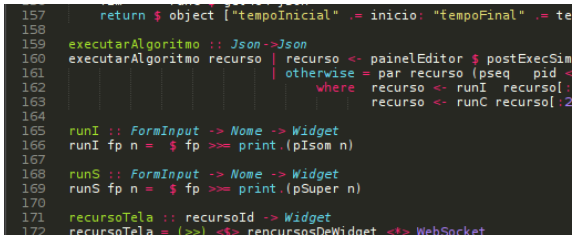

Figura 5. Código fonte contendo as funções do compilador que foram modificas.

Para a geração de gráficos, o software, antes de mandar o código fonte para o compilador, realiza-se uma medida *timestamp* da entrada, para então enviar o código fonte para o compilador. Com o código fonte, o compilador executa a simulação ou execução do código, e devolve o processamento para aplicação. Após este processamento é feita outra leitura do *timestamp* para então extrair a diferença da medida de entrada com a medida de saída. Esta medida é enviada então para a interface do usuário onde são gerados os gráficos relativos ao tempo gasto no processamento do código.

Através do compilador da linguagem QML, ao executar a função ”RunC”e passando como parâmetro da função o código fonte proveniente da interface, obtém de saída a compilação do circuito relativo ao código compilado. A saída corresponde a um texto escrito por uma linguagem de marcação pória do compilador. Posteriormente este texto é lido e convertido para uma imagem que represente o diagrama de circuíto quântico.

4.3. Arquitetura

O desenvolvimento do software é segmentado por nove módulos distintos, sendo estes representados na Figura 6.

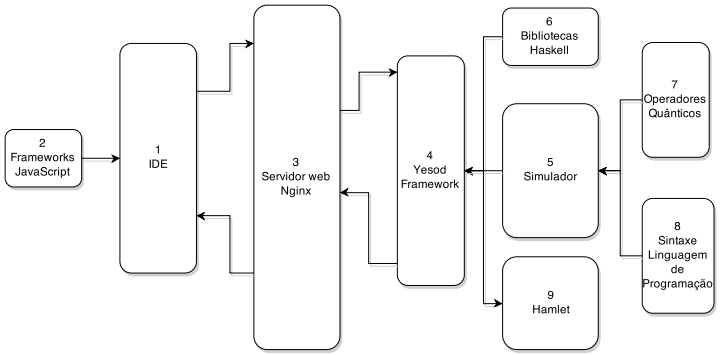

Figura 6. Principais módulos do ambiente integrado de simulação

Para que se possa compreender melhor, o módulo de número 1 representa a interface com o usuário. O módulo de número 2, é a representação dos componentes *JavaS-*

cript que estão inseridos no módulo número 1 para prover funcionalidades na interface com o usuário. O módulo número 3, representa o servidor web, o qual intermedia as requisições tanto de entrada quanto de saída. O módulo número 4 é a representação do *framework*, que processa todas as requisições e define rotas e módulos para execução das ações oriundas da interface. O módulo número 5, é a representação do simulador composto pelos módulos de números 7 e 8, que respectivamente representam os módulos do compilador da linguagem QML que anteriormente foram descritos. O módulo 6 é representado por bibliotecas da linguagem Haskell. E, por último, o módulo número 9, que corresponde a representação do *hamlet*, sistema de *templates* do *framework Yesod*.

4.4. Interface do usuário

A interface deste software inicialmente resume-se em uma única tela, contendo quatro painéis distintos, conforme pode ser visto na Figura 7. Cada um destes quatros painéis possui uma funcionalidade específica e todos trabalham em conjunto, detalhando cada resultado obtido durante as operações executadas pelo software.

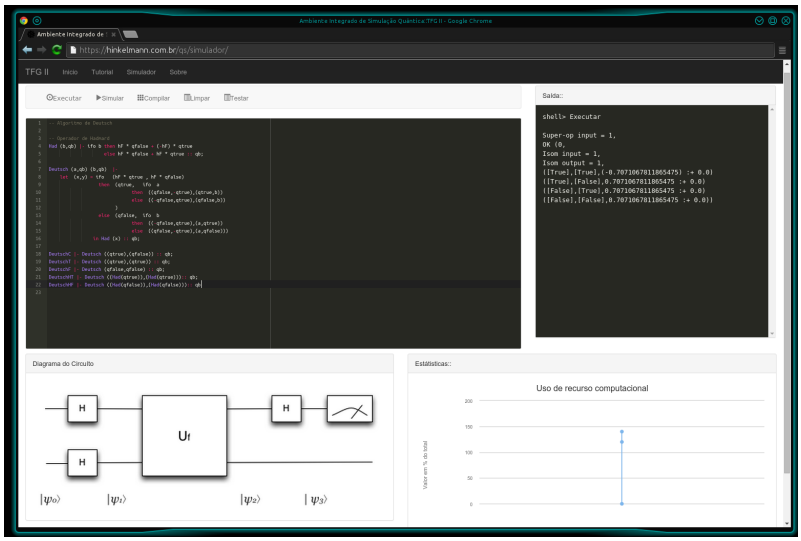

Figura 7. Tela Inicial do Simulador

O primeiro painel é um editor de código fonte QML e está localizado na posição superior esquerda, conforme é mostrado na Figura 7. Na parte superior deste mesmo painel, encontram-se cinco botões que são responsáveis pelo desencadeamento da execução das funcionalidades implementadas no software, sendo estas, a operação de simulação, execução, compilação, limpeza, e testes. O funcionamento de cada operação que é desencadeada através do manuseio destes botões é descrita na seção 4.2.

No segundo painel, encontra-se um console com a saída dos resultados obtidos no processamento executado através do primeiro painel, este painel está localizado na posição superior direita, e pode ser visto na Figura 7.

O terceiro painel, que encontra-se localizado na posição inferior esquerda, encontra-se a imagem que é gerada do circuito quântico que foi percorrido através da simulação, esta é representada na Figura 7. Esta operação faz com que a cada operador quântico visitado no processo de simulação seja inserida uma notação dentro de um

gráfico, no final da execução, este gráfico é percorrido e seus símbolos são demonstrados na forma de um diagrama de circuito quântico.

O último painel, representado na Figura 7, demonstra um gráfico referente à demanda de processamento e o tempo total para o processamento de cada operação executada no primeiro painel. A finalidade destes gráficos é auxiliar o usuário desta ferramenta desde a otimização de algoritmos até a comparação entre outros algoritmos executados.

5. Resultados

Neste Seção são apresentados os resultados da presente pesquisa de acordo com os objetivos propostos e a conclusão sobre o trabalho desenvolvido. É importante ressaltar que as simulações que foram realizadas pela ferramenta foram executadas em computador com 16GB de memória RAM, com sistema operacional FreeBSD 9.5, tendo como processador Intel Xeon E5645 com 2,2GHz e 12 MB de cache.

Conforme o objetivo geral desta pesquisa, o desenvolvimento do ambiente integrado de simulação foi criado conforme proposta, de acordo com o que fora descrito no Seção 4.

Conforme os objetivos específicos descritos neste trabalho, o objetivo de criar uma metodologia de desenvolvimento aplicada à programação funcional foi alcançado, o que possibilitou o desenvolvimento do software proposto.

O objetivo de simular computações quânticas através de algoritmos escritos em linguagem de programação quântica QML, foi possível através da modificação de algumas funções, bem como da arquitetura do compilador. Estas alterações viabilizaram que o compilador fosse acoplado no *framework Yesod*, possibilitando assim que fossem aproveitados os recursos provenientes da linguagem de programação QML, principalmente o recurso da simulação de algoritmos quânticos, que é o objeto deste estudo.

Criou-se o ambiente integrado para a realização de simulações utilizando o paradigma de programação funcional com a linguagem de programação Haskell para plataforma web. Desta forma, averiguou-se o seu funcionamento através da simulação dos algoritmos de Deutsch proposto por [Deutsch 1985] e do algoritmo de teletransporte quântico proposto por [Selinger and Valiron 2005], onde ambos estão descritos na Figura 3 e codificados com a linguagem QML.

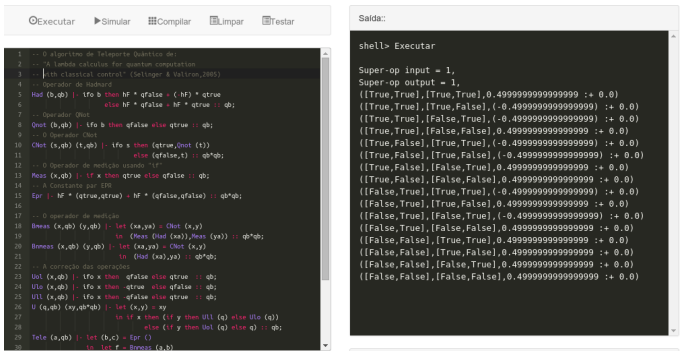

Figura 8. Resultado da execução de um algoritmo quântico de Teletransporte

Pode-se analisar na Figura 8 a execução destes dois algoritmos, que mostra que

é possível utilizar o software para a realização de simulações de computações quânticas, através de algoritmos codificados com a linguagem QML.

6. Conclusão

O ambiente integrado de simulação demonstrou-se como uma ferramenta inovadora no que tange ao estudo e desenvolvimento das linguagens de programação quântica, por ser um software voltado para a plataforma web, poderá facilmente ser escalonado quando houver a necessidade de demanda por processamento em virtude da execução de algoritmos quânticos complexos.

De acordo com a proposta desta pesquisa, os resultados obtidos com o desenvolvimento deste trabalho foram satisfatórios, pois além da modificação do compilador QML, tranformando-o em um módulo para o *framework Yesod*, criou-se um ambiente integrado de simulação para a plataforma web, utilizando a linguagem de programação Haskell.

Devido à complexidade e do tempo necessário para o desenvolvimento desta ferramenta, espera-se que as próximas atualizações do software consigam abranger inteiramente a sintaxe da linguagem de circuitos resultante da compilação do código fonte QML.

Referências

[Altenkirch and Grattage 2005] Altenkirch, T. and Grattage, J. (2005). A functional quantum programming language. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 249–258.

[ALVES 2003] ALVES, F. L. (2003). Computação quântica: Fundamentos físicos e perspectivas. *Univesidade Federal de Lavras. Lavras, MG*.

[Baranauskas 1993] Baranauskas, M. C. C. (1993). Procedimento, função, objeto ou lógica? linguagens de programação vistas pelos seus paradigmas. *Computadores e Conhecimento: Repensando a Educação. Campinas, SP, Gráfica Central da Unicamp*.

[Barbosa 2007] Barbosa, A. (2007). Um simulador simbólico de circuitos quânticos. Dissertação de mestrado, Universidade Federal de Campina Grande.

[Cabral 2004] Cabral, G. E. M. (2004). Uma ferramenta para projeto e simulação de circuitos quânticos. Dissertação de mestrado, Universidade Federal de Campina Grande.

[Carvalho et al. 2011] Carvalho, L., Lavor, C., and Motta, V. (2011). Caracterização matemática e visualização da esfera de bloch: Ferramentas para computação quântica doi: 10.5540/tema. 2007.08. 03.0351. *TEMA-Tendências em Matemática Aplicada e Computacional*, 8(3):351–360.

[de Carvalho Junior 2000] de Carvalho Junior, F. H. (2000). Haskell#, uma extensão paralela para haskell. Dissertação de mestrado, Universidade Federal de Pernambuco.

[de Lima Marquezino 2006] de Lima Marquezino, F. (2006). *A transformada de fourier quântica aproximada e sua simulação*. PhD thesis, Master's thesis, Laboratório Nacional de Computação Científica.

[Deutsch 1985] Deutsch, D. (1985). Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117.

[Feynman 1982] Feynman, R. P. (1982). Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488.

[Grover 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM.

[Karczmarczuk 2003] Karczmarczuk, J. (2003). Structure and interpretation of quantum mechanics: a functional framework. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, Haskell '03, pages 50–61, New York, NY, USA. ACM.

[Knill 1996] Knill, E. (1996). Conventions for quantum pseudocode. Technical report, Citeseer.

[Mauerer 2005] Mauerer, W. (2005). Semantics and simulation of communication in quantum computing. *Master's thesis, University Erlangen-Nuremberg*.

[Nielsen and Chuang 2010] Nielsen, M. and Chuang, I. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

[Oliveira 2007] Oliveira, N. A. (2007). A utilização do algoritmo quântico de busca em problemas da teoria da informação. Dissertação de mestrado, Universidade Federal de Campina Grande.

[Peyton and Hughes 1997] Peyton, S. and Hughes, J. (1997). *Report on the Programming Language Haskell: A Non-strict, Purely Functional Language: Version 1.4*.

[Sabry 2003] Sabry, A. (2003). Modeling quantum computing in haskell. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, pages 39–49. ACM.

[Sebesta 2003] Sebesta, R. W. (2003). *Conceitos de linguagens de programação*. Bookman.

[Selinger 2004] Selinger, P. (2004). A brief survey of quantum programming languages. In Kameyama, Y. and Stuckey, P., editors, *Functional and Logic Programming*, volume 2998 of *Lecture Notes in Computer Science*, pages 1–6. Springer Berlin Heidelberg.

[Selinger and Valiron 2005] Selinger, P. and Valiron, B. (2005). A lambda calculus for quantum computation with classical control. In *Typed Lambda Calculi and Applications*, pages 354–368. Springer.

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE.

[Sofge 2008] Sofge, D. A. (2008). A survey of quantum programming languages: History, methods, and tools. In *Quantum, Nano and Micro Technologies, 2008 Second International Conference on*, pages 66–71.

[Thompson 1999] Thompson, S. (1999). *Haskell: the craft of functional programming*, volume 2. Addison-Wesley.

[Vizzotto and da Rocha Costa 2006] Vizzotto, J. K. and da Rocha Costa, A. C. (2006). Linguagens de programaçao quântica: Um apanhado geral. In *Workshop-Escola de Computaçao e Informaçao Quântica*.