qGM$_C$-ANALYZER: UMA BIBLIOTECA EM C++ PARA SUPORTE À SIMULAÇÃO QUÂNTICA NO VPE-qGM

SCHMALFUSS, Murilo[1]; MARON, Adriano[2]; PILLA, Maurício[3]; REISER, Renata[3]

[1]Universidade Federal de Pelotas, Bacharelado em Ciência da Computação; [2]Universidade Federal de Pelotas, Mestrado em Ciência da Computação; [3]Universidade Federal de Pelotas.
Centro de Desenvolvimento e Tecnologia
{mfschmalfuss, akmaron, pilla, reiser}@inf.ufpel.edu.br

1 INTRODUÇÃO

A Computação Quântica (*CQ*) é um paradigma computacional, fundamentado nos postulados definidos pela Mecânica Quântica (*MQ*), que prevê a concepção de algoritmos quânticos, os quais, em vários cenários, são exponencialmente mais rápidos que seus análogos clássicos (GROVER 1996, SHOR 1997). Entretanto, tais algoritmos obtém esse desempenho quando executados sobre um *hardware* quântico, o qual está em fase de desenvolvimento e, em seu atual estágio, sem suporte a sistemas mais complexos. Nesse contexto, a simulação de algoritmos em computadores clássicos viabiliza o desenvolvimento e teste de algoritmos quânticos, antecipando o conhecimento acerca de seu comportamento quando da execução sobre um *hardware* quântico. Assim, no cenário atual (BARBOSA 2007, VIAMONTES 2007), a simulação de sistemas quânticos através de computadores clássicos ainda se mostra um desafio de pesquisa em aberto, justificando o estudo de soluções voltadas para a simplificação no processo de modelagem e interpretação de algoritmos e fenômenos quânticos.

O ambiente *VPE-qGM* (*Visual Programming Environment for the Quantum Geometric Machine Model*) (MARON et al. 2010) está em desenvolvimento com o objetivo de auxiliar na modelagem e simulação, sequencial e distribuída, de algoritmos quânticos, apresentando as construções e a evolução dos estados dos sistemas quânticos a partir de um conjunto de *interfaces* gráficas desenvolvidas em *Python*. Considerando o alto custo computacional inerente à simulação de algoritmos quânticos em computadores clássicos, propõe-se a extensão das capacidades de simulação do *VPE-qGM.* Neste sentido, a principal contribuição deste trabalho consiste no desenvolvimento da biblioteca de execução *qGM$_C$-Analyzer*, uma extensão da *qGM-Analyzer* na linguagem *C/C++*, e sua integração ao ambiente *VPE-qGM.* Os esforços relacionados a este trabalho se justificam pelo melhor desempenho da linguagem *C/C++* frente à *Python*, para aceleração das computações relacionadas a simulação de algoritmos quânticos no *VPE-qGM*.

Na *CQ*, o *qubit* é a unidade básica de informação, definido por um vetor de estado, unitário e bidimensional, genericamente descrito, na notação de *Dirac* (NIELSEN 2003), pela expressão $|\upsilon\rangle = \alpha|0\rangle + \beta|1\rangle$. Os coeficientes α e β são números complexos correspondentes às amplitudes dos respectivos estados, respeitando a condição de normalização $|\alpha|^2 + |\beta|^2 = 1$, garantindo a unitariedade do vetor de estado do sistema, representado por $(\alpha, \beta)^t$. As amplitudes permitem que o sistema represente, simultaneamente, estados distintos, configurando um estado de superposição quântica. O espaço de estados de um sistema quântico de múltiplos *qubits* é compreendido pelo produto tensorial do espaço de estados de seus sistemas componentes. Considerando um sistema quântico de dois *qubits*,

$|\upsilon\rangle = \alpha|0\rangle + \beta|1\rangle$ *e* $|\varphi\rangle = \gamma|0\rangle + \delta|1\rangle$, o espaço de estados é composto pelo produto tensor $|\upsilon\rangle \otimes |\varphi\rangle$, ou seja $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$.

O ambiente *VPE-qGM*, fundamentado no modelo de processos *qGM* (*Quantum Geometric Machine Model*) (REISER e AMARAL 2010), é constituído de construtores para modelagem e simulação gráfica de aplicações quânticas. De acordo com o modelo *qGM*, a noção de portas quânticas pode ser substituída pelo conceito de sincronização de processos elementares (*PEs*). Neste contexto, uma transformação quântica, aplicada à *N qubits*, pode ser modelada pela sincronização de 2^N *PEs*, cujas parametrizações satisfazem as condições equivalentes à definição dos vetores componentes da matriz associada. Assim, durante a simulação, ocorre a execução dos *PEs*, os quais têm suas correspondentes computações efetuadas pela biblioteca *qGM-Analyzer*, manipulando os dados presentes nas posições de memória e simulando o comportamento de um sistema quântico.A biblioteca de execução dos *PEs*, denominada *qGM-Analyzer*, implementa otimizações que controlam o aumento exponencial dos vetores componentes das matrizes de definição do operador (*MDOs*) de múltiplos *qubits*, conforme introduzido em (MARON et al., 2011). Os resultados relacionados comprovam a redução no consumo de memória durante a simulação, suportando algoritmos com 11 *qubits*. Entretanto, o tempo total de simulação obtido permanece elevado. Tal desempenho está fortemente relacionado com a linguagem de programação adotada pelo ambiente *VPE-qGM*. Por ser uma linguagem interpretada, *Python* apresenta desempenho inferior as linguagens tradicionalmente utilizadas para solução de problemas computacionalmente intensivos. Nesse sentido, justifica-se a extensão da biblioteca *qGM-Analyze*r, considerando a linguagem *C*++.

2 METODOLOGIA

A implementação da biblioteca *qGM$_C$-Analyzer*, além de incluir as otimizações já introduzidas em (MARON et al. 2011), também altera as estruturas de dados fazendo uso de recursos nativos oferecidos pela linguagem, visando a otimização da execução. Para compatibilidade da biblioteca com o ambiente *VPE-qGM*, considera-se a biblioteca *Boost 1.49.0* (BOOST 2012), integrando as interfaces entre *Python* e *C*++.

A biblioteca *Boost* oferece diversos recursos e ferramentas que estendem a linguagem *C*++. Desses recursos, explora-se, especificamente, as funcionalidades providas pelo módulo *Boost-Python*. Usando funções definidas na biblioteca *Boost* é possível especificar quais métodos tornar-se-ão visíveis para o interpretador Python. A ferramenta *Boost-Jam* auxilia no processo de compilação e *binding* da biblioteca, provê a leitura do código-fonte e gera a biblioteca compartilhada utilizando o compilador *GCC* 4.6.3 com as seguintes *flags* de compilação: *O3* (identifica o nível de otimização realizada pelo compilador) e *funroll-loops* (desenrolamento de *loops*). Esta biblioteca compartilhada pode ser então importada e utilizada dentro do ambiente.

Seguem-se duas das principais otimizações desenvolvidas na nova abordagem da *qGM$_C$-Analyzer:* (*i*) a abordagem em *Python* utilizava listas como principal estrutura de dado, em *C*++ foram utilizados vetores; (*ii*) passagem dos parâmetros por referência para as funções auxiliares, eliminando assim o *overhead* da cópia. Como *Python* possui tipagem dinâmica, a resolução dos tipos das variáveis era feita em tempo de execução, na nova abordagem com todas as

variáveis utilizadas sendo declaradas no código-fonte, essa resolução é feita em tempo de compilação, melhorando o desempenho e permitindo ao compilador otimizações no código gerado.

3 RESULTADOS E DISCUSSÃO

Os estudos de casos para validação e análise de desempenho da implementação da *qGM$_C$-Analyzer* em *C++* contemplam sistemas entre 12 e 17 *qubits*. A metodologia dos testes utilizada considera, para cada estudo de caso, a realização de 10 simulações, e a máquina utilizada possui as seguintes características principais: processador *Intel Core i7-3770 @ 3.5 GHz*, *8GB RAM* e sistema operacional *Ubuntu 12.04 64 bits*. Foram monitorados o tempo de execução de cada amostra e o correspondente consumo de memória. A principal comparação de desempenho se dá com a biblioteca *qGM-Analyzer* implementada em *Python*. Os tempos de simulação, a quantidade de *PEs* executados em cada estudo de caso e a correspondente quantidade de valores gerados estão descritos na Tabela 1.

Tabela 1. Tempos de Simulação

Operação	Qubits	Nº de PEs	Qtd. Valores	Python		C++	
				Tempo (s)	Mem (Mb)	Tempo(s)	Mem (Mb)
Hadamard 1	12	4096	4096	24,904	13	4,436	15
Hadamard 2	13	8192	8192	87,040	13	16,307	15
Hadamard 3	14	16384	16384	324,506	14	62,807	16
Hadamard 4	15	32768	32768	1238,691	14	244,753	20
Hadamard 5	16	65536	65536	7125,83	16	1062,291	24
Hadamard 6	17	131072	131072	24138,29	18	3944,302	29
Pauly X	17	131072	1	61,450	18	15,349	29
Controladas 1	7	128	1	0,026	13	0,006	15
Controladas 2	5	32	2	0,006	13	0,001	15
Controladas 3	14	16384	1	7,253	14	2,063	16
Controladas 4	16	65536	1024	210,462	16	37,284	22

Como esperado, o aumento exponencial na quantidade de operações para os estudos de caso baseados em operações *Hadamard* gera elevado tempo de simulação. Para as transformações *Pauly X*, o tempo de execução foi menor do que os estudos de caso com operações *Hadamard*, apesar da maior quantidade de *qubits* aplicada. Tal comportamento se refere a redução dos valores gerados por *PEs*. Devido à arbitrariedade das configurações aplicadas, as operações compostas por transformações controladas apresentam dois comportamentos: (*i*) o primeiro, resultando em um baixo tempo de execução, sem aplicação de transformações *Hadamard*; (*ii*) o segundo, resultando em tempo de simulação maior devido a presença de várias transformações *Hadamard*.

4 CONCLUSÃO

As otimizações realizadas na *qGM$_C$-Analyzer* apresentaram bom desempenho em relação a biblioteca anterior, reduzindo significativamente o tempo

de simulação da biblioteca de execução dos *PEs*. A análise da eficiência da *qGM$_C$-Analyzer* para para determinadas configurações de transformações quânticas acima 17 *qubits* passa a ser desafio, uma vez que o tempo de simulação cresce exponencialmente com o aumento na quantidade de *qubits*.

A continuidade do trabalho consiste na extensão da biblioteca para execução paralela, integrando com módulos e bibliotecas para paralelização disponíveis para *C++* como *OpenMP* (AYGUADE e CHAPMAN 2003), e também para paralelização massiva utilizando placas gráficas através da arquitetura *CUDA* (NVIDIA 2009). A *API OpenMP* é direcionada para o uso com *C/C++*, possibilitando sua utilização futura na biblioteca. Estudos já estão sendo realizados para a paralelização da biblioteca.

5 REFERÊNCIAS

AYGUADE, E; CHAPMAN, B. Introduction: Special Issue: OpenMP. **Scientific Programming**, v. 11, n. 2, p. 79-80, 2003.

BARBOSA, A. **Um Simulador Simbólico de Circuitos Quânticos**. 2007. Tese de Mestrado, UFCG.

BOOST. **Boost 1.49.0 Library Documentation**. www.boost.org/doc/libs/1_49_0/, 2012.

GROVER, L. A Fast Quantum Mechanical Algorithm for Database Search. **Proc. of the Twenty-Eight Annual ACM Symposium on Theory of Computing**. p. 212-219, 1996.

MARON, A; PINHEIRO, A; REISER, R, YAMIN, A; PILLA, M. Ambiente VPE-qGM: Em Direção a uma Nova Abordagem para Simulações Quânticas. **Rev. do CCEI**. v.14, p. 29-46, 2010.

MARON, A; ÁVILA, A; REISER, R; PILLA, M. Introduzindo uma Nova Abordagem para Simulação Quântica com Baixa Complexidade Espacial. **Anais do DINCON 2011**. SBMAC. p. 1-6, 2011.

NIELSEN, M; CHUANG, I. **Computação Quântica e Informação Quântica**. Bookman, 2003.

NVIDIA, **CUDA Programming Guide**. NVIDIA Corp., 2009.

REISER, R; AMARAL, R. The Quantum States Space in the qGM Model. **Anais III Workshop Escola de Computação e Informação Quântica**. Editora do LNCC. p. 92-101, 2010.

SHOR, P. Polynomial-Time Algorithms for Prime Factorization and Discret Logarithms on a Quantum Computer. **SIAM Journal on Computing**. 1997.

VIAMONTES, G. **Efficient Quantum Circuit Simulation**. 2007. Tese de Doutorado, The University of Michigan.