qGM$_C$-ANALYZER: UMA BIBLIOTECA PARA SIMULAÇÃO QUÂNTICA EM ARQUITETURAS PARALELAS

MURILO FIGUEIREDO SCHMALFUSS[1]; ADRIANO KURZ MARON[2]; RENATA HAX SANDER REISER[1]; MAURÍCIO LIMA PILLA[1]

[1]Universidade Federal de Pelotas – {mfschmalfuss, reiser, pilla}@inf.ufpel.edu.br
[2]University of Pittsburgh – akk48@pitt.edu

1. INTRODUÇÃO

A Computação Quântica (*CQ*) (NIELSEN; CHUANG, 2000) segue atingindo novos marcos rumo a construção de computadores quânticos. Apesar de todos os esforços, vários desafios técnicos limitam os sistemas atuais à apenas alguns *bits* quânticos (STEEB; HARDY, 2004). Devido à indisponibilidade de *hardware* quântico, o estudo e desenvolvimento de aplicações na *CQ* usualmente é feito estritamente pela especificação matemática das computações ou por meio de ferramentas de simulação. Este último caracteriza a abordagem mais prática, entretanto a complexidade computacional associada a simulação de sistemas quânticos a partir de computadores clássicos limita o tamanho dos sistemas que podem ser simulados.

O projeto no qual este trabalho está inserido busca consolidar a integração de vários esforços de pesquisa, com destaque para:

- Distribuição das computações em *cluster* (ÁVILA et al., 2012);
- Simulação quântica através de *GPUs* (MARON et al., 2013);
- Simulação quântica através de *CPUs multicore*.

Atualmente situado sob o contexto do ambiente de simulação quântica *VPE-qGM* (*Visual Programming Environment for the Quantum Geometric Machine Model*), este trabalho tem o objetivo de estabelecer o suporte à aceleração da biblioteca de execução do ambiente através de processadores *multicore*, beneficiando-se dos recursos providos pela biblioteca *OpenMP* (AYGUADE; CHAPMAN, 2003). Consolida-se assim a biblioteca *qGM$_C$-Analyzer*, com a implementação, desenvolvimento e validação de algoritmo para simulação quântica em arquiteturas *multicore*, cuja modelagem foi introduzida em SCHMALFUSS et al. (2012).

O ambiente *VPE-qGM*, fundamentado no modelo de processos *qGM* (*Quantum Geometric Machine Model*) REISER; 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*).

No ambiente *VPE-qGM*, o *PE* é um elemento estruturado por três atributos: (*i*) *Ação*: Corresponde às transformações quânticas aplicadas a diferentes *qubits* em um mesmo instante de tempo; (*ii*) *Parâmetros*: Contém dados auxiliares associados à definição das transformações quânticas; (*iii*) *Posição*: Posição de escrita em um espaço de memória global e compartilhada, na qual é armazenado o resultado calculado pelo *PE*.

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 (transformação unitária ou de medida) associada.

Assim, durante a simulação, ocorre a execução (sequencial ou síncrona) 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 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, devido a quantidade de operações necessárias para simular uma transformação quântica.

Tendo em vista que o crescimento da complexidade algorítmica se dá de forma exponencial, abordagens paralelas e distribuídas foram estudadas para otimizar os cálculos envolvidos na simulação de uma transformação quântica.

2. METODOLOGIA

A implementação da biblioteca *qGM-Analyzer* em *C++* segue as otimizações introduzidas em MARON et al. (2011), apenas alterando as estruturas de dados utilizados e fazendo uso de recursos nativos oferecidos pela linguagem visando a otimização da execução. Visando a compatibilidade da biblioteca com o ambiente *VPE-qGM*, é considerada a biblioteca *Boost* 1.49.0 (BOOST, 2012) para integração das duas linguagens envolvidas. Para a implementação do paralelismo foi utilizada a biblioteca *OpenMP* (AYGUADE; CHAPMAN, 2003).

Para auxiliar no desenvolvimento da biblioteca *qGM$_C$-Analyzer* foi utilizado o módulo *Boost-Python*, que permite que sejam utilizados no código *C++* tipos do *Python*, como as listas de valores utilizadas no ambiente *VPE-qGM*, e que seja realizado a conversão destes tipos para tipos da linguagem *C++*. Outra funcionalidade utilizada do módulo *Boost-Python* foi a geração da biblioteca compartilhada importada pelo ambiente em *Python*.

A implementação em *C++* manteve o mesmo algoritmo desenvolvido em MARON et al. (2011), porém a forma de armazenamento dos dados e os algoritmos para acesso a esses dados foram modificados para diminuir a complexidade da biblioteca.

Os fatores que contribuíram para um melhor desempenho foram a utilização de vetores como alternativa as listas do *Python e* o fato de a linguagem *C++* ser compilada, permitindo que o compilador gere otimizações no código gerado.

Na implementação paralela, cada *thread* possui uma cópia privada da pilha, e as memórias de escrita e leitura são compartilhadas entre todos os *threads*, pois cada valor calculado é escrito em uma posição diferente da memória.

A divisão dos *threads* é feita de forma a manter juntos os valores das matrizes necessários para o cálculo de uma posição. Para isso as iterações são divididas levando em conta o número de colunas da primeira matriz envolvida na transformação. Nos casos em que a transformação possui apenas uma matriz, os *threads* são divididos de forma diferente, para que o trabalho seja distribuído de forma balanceada.

A definição do número de *threads* utilizadas pela biblioteca é definida por uma variável de ambiente (*OMP_NUM_THREADS*), gerando uma implementação mais flexível. Ou seja, dependendo da transformação quântica, tem-se um controle da granulosidade visando melhor desempenho da biblioteca.

3. RESULTADOS E DISCUSSÃO

Para validação e análise de desempenho da implementação da *qGM-Analyzer* em *C++*, foram desenvolvidos estudos de caso envolvendo sincronizações arbitrárias de transformações quânticas, contemplando sistemas entre 7 e 20 *qubits*.

A metodologia dos testes utilizada contempla, para cada estudo de caso, a realização de 15 simulações. A máquina utilizada na simulação possui as seguintes características principais: processador Intel Core i7-3770 @ 3.4 GHz, 8 GiB RAM e sistema operacional Ubuntu 12.04 64 *bits*.

Foram monitorados o tempo de execução e consumo de memória de cada amostra. A principal comparação de desempenho se dá com a execução da biblioteca em diferentes números de *threads*.

Estas execuções compreenderam duas etapas:

- Geração dos valores não nulos associados ao correspondente vetor componente da matriz de definição da transformação quântica modelada;
- Multiplicação desses valores pelas amplitudes obtidas da estrutura de memória que modela o espaço de estados do sistema quântico.

Nos casos das simulações das *Hadamards*, devido ao elevado número de operações envolvidas, pode-se facilmente observar um desempenho elevado, ou seja, o aumento no número de *qubits* diretamente relacionado com a diminuição significativa do tempo de execução.

Nos casos das transformações controladas *Pauli X* houve um aumento no tempo de execução. Este fato se justifica pelo baixo número de operações envolvidas. Sendo a *Pauli X* uma matriz esparsa, seus valores zerados não são computados pelo algoritmo otimizado, tornando o *overhead* da criação e troca de contexto dos *threads* o principal custo da transformação.

Tabela 1: Tempos de Simulação

Operação	Qubits	1 Thread		2 Threads		4 Threads		8 Threads	
		Tempo (s)	Mem (Mib)	Tempo (s)	Mem (Mib)	Tempo (s)	Mem (Mib)	Tempo (s)	Mem (Mib)
Hadamard	15	18.961	15	9.5865	15	4.9964	16	4.6993	16
Hadamard	16	71.274	18	35.281	18	18.547	18	14.770	18
Hadamard	17	286.41	22	144.13	22	74.951	22	59.780	22
Hadamard	18	1162.6	31	582.92	30	303.35	30	245.16	30
Pauli X	17	0.0793	19	0.0975	23	0.0984	24	0.1121	24
Pauli X	18	0.1629	35	0.1868	36	0.1970	38	0.2248	38
Pauli X	19	0.3410	55	0.3774	54	0.4055	57	0.4345	58
Pauli X	20	0.7075	82	0.7674	82	0.8047	83	0.8841	83
Control.	7	0.0112	11	0.0126	11	0.0129	11	0.0135	12
Control.	7	0.0159	11	0.0174	11	0.0180	12	0.0194	12
Control.	10	0.0081	15	0.0092	15	0.0109	15	0.0145	15

Os tempos de simulação, em cada estudo de caso, abrangendo os casos para 1, 2, 4 e 8 *threads* são descritos na Tabela 1. Como analisado na tabela, os tempos de execução para as transformações *Hadamards* obtiveram os melhores desempenhos, chegando a um *speedup* relativo de 4,75, para o caso de 18 *qubits*. As transformações *Pauli X* e controladas tiveram o mesmo comportamento, com o seu tempo de execução aumentando conforme aumentava o número de *threads*, pois envolvem poucos cálculos, e seu tempo é dominado pela criação e troca de contexto dos *threads*.

O desvio padrão para as médias dos valores inseridos na tabela foram omitidos, seus valores ficaram na faixa de 1% para todas as médias.

O consumo de memória manteve-se constante com a variação do número de *threads*, aumentando juntamente com o número de *qubits* da transformação. Parte deste consumo é relativo ao interpretador *Python* e o carregamento em tempo de execução da biblioteca *Boost*.

4. CONCLUSÕES

As otimizações realizadas e a implementação paralela representaram um ganho significativo no tempo de execução das transformações quânticas, permitindo a simulação de transformações de 18 *qubits* com elevado número de operações, como no caso das *Hadamards*.

A continuidade do trabalho consiste na integração da biblioteca *qGM$_C$-Analyzer* com o ambiente *VPE-qGM* além da expansão para suporte a transformações *multiqubits* e na implementação de aplicações em outras bibliotecas disponíveis, para comparação de desempenho entre diferentes ferramentas para suporte a simulação quântica.

A proposta se identifica diretamente com a integração da biblioteca *qGM$_C$-Analyzer* implementada em *CUDA* para execução de *QPP* (*Quantum Partial Process*) em *GPU*, compreendendo nesta etapa além das operações de controle também operações de medidas quânticas.

Como aplicações, consideram-se as simulações de algoritmos de busca e criptografia com aplicações na computação científica.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BOOST. **Boost 1.49.0 library documentation**. Acessado em 05 de out. 2013. Online. DisponÍvel em: http://www.boost.org/doc/libs/1_49_0/

ÁVILA, A.; MARON, A.; REISER, R.; PILLA, M. **Extending the VirD-GM environment for the distributed execution of quantum processes**. Proceedings of the XIII WSCAD-WIC, 2012.

AYGUADE, E.; CHAPMAN, B. **Introduction: Special issue: OpenMP**. Scientific Programming, 2003.

MARON, A.; ÁVILA, A.; REISER, R.; PILLA, M. **Introduzindo uma nova abordagem para simulação quântica com baixa complexidade espacial**. In: Anais do DINCON 2011, 2011.

MARON, A.; REISER, R.; PILLA, M. **High-performance quantum computing simulation for the quantum geometric machine model**. In: Proceedings of CCGRID 2013, 2013.

NIELSEN, M.; CHUANG, I. **Quantum Computation and Quantum Information**. Cambridge University Press, 2000.

REISER, R.; AMARAL, R. **The quantum states space in the qGM model**. In: Proceedings of the III WECIQ, 2010.

SCHMALFUSS, M.; MARON, A.; REISER, R.; PILLA, M. **qGM$_C$-Analyzer: Biblioteca para suporte à simulação quântica em C++**. In: Proceedings of the XIII WSCAD-WIC, 2012.

STEEB, W.; HARDY, Y. **Problems and solutions in quantum computing and quantum information**. World Scientific, 2004.