UMA ABORDAGEM PARA SIMULAÇÃO QUÂNTICA VIA SOFTWARE COM BAIXO CONSUMO DE MEMÓRIA

MARON, Adriano; REISER, Renata; PILLA, Maurício

Universidade Federal de Pelotas – PPGC/Ciência da Computação
{akmaron, reiser, pilla}@inf.ufpel.edu.br

1. INTRODUÇÃO

A Computação Quântica (*CQ*) (NIELSEN; CHUANG, 2003) fundamenta o desenvolvimento de algoritmos quânticos, os quais são capazes de solucionar problemas complexos exigindo menos recursos de tempo e memória. Entretanto, o *hardware* quântico está nos estágios iniciais de desenvolvimento, não suportando computações complexas. A simulação quântica em computadores clássicos é uma alternativa para estudo e validação de algoritmos quânticos. Para isso, alguns conceitos associados à *CQ* devem ser implementados.

O *qubit* é a unidade básica de informação quântica, vetorialmente representado por $(a,b)^t$, sendo *a* e *b* amplitudes associadas aos estados básicos |0> e |1> do *qubit*. Sistemas compostos por muitos *qubits* tem seu espaço de estados gerado a partir do produto tensorial dos espaços de estados de cada *qubit*. Para um sistema quântico de 2 *qubits*, tem-se o vetor $(a,b,c,d)^t$, com cada amplitude associada aos estados |00>, |01>, |10> e |11>, respectivamente.

A evolução do estado é feita por transformações quânticas, definidas por matrizes quadradas ortonormalizadas de ordem 2^Q, sendo *Q* a quantidade de *qubits* do sistema. Assim, o próximo estado do sistema é obtido por um produto matriz-vetor de complexidade exponencialmente maior que seu tamanho.

A simulação de um algoritmo quântico implica em muitos produtos matriz-vetor, refletindo no tempo de simulação e no consumo de processador e memória.

O ambiente *VPE-qGM* (*Visual Programming Environment for the qGM Model*) (MARON et. al, 2011) está em desenvolvimento para suporte a modelagem e simulação distribuída de algoritmos da *CQ*, a partir de abstrações do modelo *qGM* (REISER; BURLAMAQUI, 2010). Nesse ambiente, para modelagem e simulação de uma transformação quântica *U*, considera-se uma estrutura de sincronização de processos elementares (*PEs*), onde cada *PE* gera, independentemente, os valores de uma linha da matriz de *U* e multiplica-os pelo vetor de estado, gerando uma amplitude do próximo estado global do sistema.

Neste trabalho é apresentada uma otimização para a biblioteca de execução dos *PEs*, denominada *qGM-Analyzer*, utilizando funções matemáticas e métodos recursivos para geração dinâmica dos valores que definem as transformações quânticas. Essa abordagem apresenta redução significativa no uso de memória.

2. METODOLOGIA

Na nova concepção da biblioteca *qGM-Analyzer*, as matrizes associadas às transformações quânticas básicas são substituídas por um conjunto de funções elementares, definidos de forma análoga a Figura 1.

Figura 1. Função de definição da transformação quântica *Hadamard*.

$$H(linha, pos) = \begin{cases} \frac{1}{\sqrt{2}}, & para\ linha = 0 \\ \frac{-1^{pos}}{\sqrt{2}}, & para\ linha = 1 \end{cases}$$

Um *VPP* consiste em um conjunto de tuplas (*valor*, *posicao*) que armazenam produtos parciais de funções elementares. A geração dinâmica dos *VPPs* ocorre a partir dos dados que parametrizam os *PEs*, sendo inseridos na Tabela 1 para geração de uma lista de *VPPs* (L_{vpp}).

Tabela 1: Tabela estrutural do *PE*

-	$x_1 = 0$ $x_2 = 0$	$x_1 = 0$ $x_2 = 1$	$x_1 = 1$ $x_2 = 0$	$x_1 = 1$ $x_2 = 1$
U_0 (b_0, x_1) x U_1 (b_1, x_2)	(α_0, p)	(β_0, p)	(γ_0, p)	(δ_0, p)
U_2 (b_2, x_1) x U_3 (b_3, x_2)	(α_1, p)	(β_1, p)	(γ_1, p)	(δ_1, p)
⋮	⋮	⋮	⋮	⋮
U_{n-2} (b_{n-2}, x_1) x U_{n-1} (b_{n-1} ,x_2)	$(\alpha_{\lfloor (n/2)-1 \rfloor}, p)$	$(\beta_{\lfloor (n/2)-1 \rfloor}, p)$	$(\gamma_{\lfloor (n/2)-1 \rfloor}, p)$	$(\delta_{\lfloor (n/2)-1 \rfloor}, p)$

Seguem algumas elucidações acerca da simbologia adotada:

- *U_0 até U_{n-1}*: Transformações quânticas contidas no atributo *Acao* do *PE*;
- *n*: Quantidade de transformações quânticas referenciadas pelo *PE*;
- *b_0* até *b_{n-1}*: Indexação de cada *bit* proveniente do número binário correspondente ao valor instanciado no atributo *Posicao* do *PE*. *b_0* é tido como o *bit* mais significante da representação;
- *x_1 e x_2:* Parâmetros para definição do valor retornado pela função;
- *p: Posição parcial associada ao produto, definido por $p = 2x_1 + x_2$.*

Caso algum produto *U_{k-1} (b_{k-1}, x_1) x U_k (b_k ,x_2)*, para *$k < n$*, seja 0, esse valor não é inserido na estrutura. Essa condição trata situações que levariam a sucessivas multiplicações por 0, como no caso da geração de valores associados a um vetor componente esparso.

É necessário, ainda, a geração de uma lista (sizesList) com valores base para indexação de posições de leitura para cada VPP. Essa lista é utilizada no cálculo da posição de memória que armazena a amplitude a ser multiplicada pelo valor gerado. Sua generalização é dada por *sizesList* = [2^{n-2}, 2^{n-4}, 2^{n-6}, … 2^{n-n}].

Para exemplificação, busca-se gerar os elementos da linha 35 da matriz $H^{\otimes 4} \otimes Id^{\otimes 2}$. A parametrização do correspondente *PE* é: *(i)* Acao = H, H, H, H, Id, Id; *(ii)* Posicao = 35; *(iii)* Parametros = Null. Substituindo todas as variáveis na Tabela 1, tem-se que L_{vpp} é

$$L_{vpp} = \begin{bmatrix} [(\frac{1}{2}, 0), (\frac{1}{2}, 1), (-\frac{1}{2}, 2), (-\frac{1}{2}, 3)], \\ [(\frac{1}{2}, 0), (\frac{1}{2}, 1), (\frac{1}{2}, 2), (\frac{1}{2}, 3)], \\ [(1, 3)] \end{bmatrix}$$

De forma análoga, tem-se que *sizesList* = [16, 4, 1]. Segue-se a aplicação de uma função recursiva sobre essas estruturas, gerando cada um dos valores associados ao vetor componente representado por este *PE*.

A partir de *L_{vpp}* e *sizesList*, tem-se a possibilidade de gerar os valores finais não nulos associados a cada elemento do vetor componente. E, para evitar a expansão do espaço de estados, utiliza-se uma função recursiva que calcula os produtos entre elementos de cada *VPP*. Produtos e posições-base parciais são gerados quando de uma chamada recursiva, sendo passados para os próximos

níveis da recursão. No caso base, caracterizado pela chamada recursiva ao último *VPP*, são gerados os valores e posições finais.

A posição indica qual amplitude deve ser multiplicada pelo valor gerado no caso base. O resultado dessa multiplicação é somado com os resultados calculados nas iterações e chamadas recursivas anteriores. Após término das recursões, o valor final é retornado à primeira chamada, sendo atualizado na posição de memória indicada pelo atributo *Posição* do *PE*. Esse processo caracteriza a obtenção da amplitude associada a um dos estados do sistema quântico após a execução das transformações. Uma abordagem mais detalhada desta metodologia pode ser vista em MARON et. al, 2011.

3. RESULTADOS E DISCUSSÕES

A validação das implementações realizadas na biblioteca de execução considera dois algoritmos quânticos: somador quântico de 5 *bits* (*Sum*) (TAKAHASHI; TANI; KUNIHIRO, 2009) e Algoritmo de Grover (*AG*) (GROVER, 1996), em instâncias para listas de 128 *(AG1),* 256 (*AG2*) e 512 (*AG3*) elementos.

A metodologia adotada nos testes consiste na análise de desempenho a partir dos seguintes parâmetros: *(i)* tempo médio de simulação; *(ii)* desvio padrão; *(iii)* consumo de memória. Os dados foram gerados a partir da execução de 15 simulações para cada estudo de caso. Para efeito de comparação, considera-se o simulador quântico universal *Zeno* (BARBOSA, 2007).

A máquina utilizada para os testes possui as seguintes configurações de *hardware*: Processador Intel Core i5 2,30 GHz, 4 GB Ram e sistema operacional Ubuntu 11.04 64 *bits*. As principais informações obtidas da simulação são apresentadas na Tabela 2.

Tabela 2: Comparação entre os ambientes *VPE-qGM* e *Zeno.*

	VPE-qGM			Zeno		
	Tempo (s)	Desvio Pad. (s)	Memória (MB)	Tempo (s)	Desvio Pad. (s)	Memoria (MB)
AG1	2,94	0,01	84,5	0,91	0,01	379,6
AG2	12,11	0,07	122,7	6,4	0,06	419,5
AG3	67,29	0,03	203,8	49,85	1,05	600,7
Sum	8,86	0,06	61,3	17,32	0,10	711,7

Em uma primeira análise, percebe-se o melhor desempenho do simulador *Zeno* quando considerado o tempo de simulação para as instâncias de *AG*. Essa diferença pode ser justificada pelo melhor desempenho da linguagem *Java* (*Zeno*) frente à linguagem *Python* (*VPE-qGM*).

O ambiente *VPE-qGM* mostra-se superior quando analisado o consumo de memória exigido para as simulações. Esse resultado provém do não armazenamento das estruturas de definição das transformações quânticas. Para as instâncias de *AG*, apesar da menor dimensão das transformações quânticas, o uso de memória mostrou-se maior do que a instância de *Sum*. Esse comportamento é justificado pela construção gráfica utilizada, a qual, para a modelagem adotada, requer uma quantidade elevada de componentes gráficos.

O ganho proveniente da nova implementação da biblioteca *qGM-Analyzer* é evidenciado quando analisado o uso de memória para o algoritmo *Sum*. Nesse cenário, para simuladores que fazem uso da operação de produto tensor, espera-

se um elevado uso da memória, como observado no simulador *Zeno*. Por utilizar uma abordagem diferenciada, o *VPE-qGM* não apresentou esse comportamento, exigindo apenas 61,3 MB de memória RAM.

Os estados finais dos sistemas quânticos obtidos para ambos os ambientes mostraram-se iguais, validando os resultados obtidos pelo *VPE-qGM*.

4. CONCLUSÕES

Os resultados obtidos neste trabalho compreendem um avanço importante para suporte a algoritmos quânticos com muitos *qubits* no *VPE-qGM*. Além de contribuir com uma nova interpretação para definição das transformações quânticas, obteve-se uma redução significativa na complexidade de espaço do algoritmo implementado na biblioteca de execução do ambiente.

A continuidade do trabalho consiste na extensão da biblioteca de execução do ambiente *VirD-GM* considerando as otimizações apresentadas neste trabalho, visando à otimização da simulação quântica distribuída. Busca-se, ainda, a otimização da representação da estrutura de memória associada ao espaço de estados dos sistemas quânticos, evitando o crescimento exponencial e otimizando o acesso aos dados armazenados.

AGRADECIMENTOS: À FAPERGS, pelo financiamento do projeto ExploreD-GM, e a CAPES, pela bolsa de mestrado concedida.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BARBOSA, A. **Um Simulador de Circuitos Quânticos.** 2007. Dissertação de Mestrado em Ciência da Computação. Universidade de Campina Grande.

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

MARON, A.K.; PINHEIRO, A.B.; REISER, R.H.S; PILLA, M. Consolidando uma Infraestrutura para Simulação Quântica Distribuída. **Anais da ERAD 2011**, Porto Alegre, v.1, p. 213-216, 2011.

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

REISER, R.; AMARAL, R. The Quantum States Space in the qGM Model. **Anais do III WECIQ**, Petrópolis, p. 91-101, 2010.

TAKAHASHI, Y; TANI, S.; KUNIHIRO, N. Quantum Addition Circuits and Unbounded Fan-Out. **Quantum Information & Computation,** New Jersey, v.10, n.9, p.872-890, 2010.