EXTENSÃO DO AMBIENTE VIRD-GM PARA SUPORTE A PROCESSOS MISTOS PARCIAIS PARA PORTAS CONTROLADAS

MURILO FIGUEIREDO SCHMALFUSS[1]; MATHEUS FIGUEIREDO SCHMALFUSS[1]; RENATA HAX SANDER REISER[1]; MAURÍCIO LIMA PILLA[1]

[1]Universidade Federal de Pelotas –
{mfschmalfuss, matheusfschmalfuss, reiser, pilla}@inf.ufpel.edu.br

1. INTRODUÇÃO

A Computação Quântica (*CQ*) é fundamentada nos postulados da Mecânica Quântica (NIELSEN; CHUANG, 2011; PORTUGAL et al., 2004), a qual provê a concepção de algoritmos mais eficientes que seus análogos clássicos (GROVER, 1996; SHOR,1997). Nestes postulados a unidade básica de informação é o *qubit* genericamente descrito na notação de Dirac pela expressão $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde os coeficientes α e β são números complexos correspondente às amplitudes dos respectivos estados, respeitando a condição de normalização $|\alpha|^2 + |\beta|^2 = 1$.

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

A mudança de estado em um sistema quântico é feita por transformações quânticas unitárias, chamadas de portas quânticas. Abaixo são apresentadas as principais portas quânticas utilizadas na *CQ*. A porta Identidade (*Id*) mantém o *qubit* no estado atual, a porta *Hadamard* (*H*) coloca os *qubits* em um estado de superposição, a porta *Pauli X* (*X*) inverte o estado do *qubit* a qual é aplicada e a porta controlada *CNOT*, aplicada a dois *qubits* (um controle e um alvo), inverte o estado do *qubit* alvo se a condição de controle for satisfeita.

$$Id = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

O difícil acesso a *hardware* quântico dificulta a concepção de algoritmos quânticos, justificando a utilização e a otimização de simuladores quânticos para a modelagem e simulação destes algoritmos. O objetivo deste trabalho é estudar novas técnicas de paralelização e otimização das estruturas associadas a transformações quânticas controladas, voltados para a simulação distribuída/paralela via *software* de algoritmos quânticos.

2. METODOLOGIA

Através da combinação das *Distribuições de QPPs* e *Distribuição de PCPs*, descritos em MARON et al. (2013), em um *layout* misto/integrado, é possível gerenciar o consumo de memória e a sobrecarga de rede. Este *layout* é nomeado *Mixed Partial Process* (*MPP*) (AVILA et al., 2014).

O ambiente *VirD-GM* se encontra em desenvolvimento com o objetivo de realizar um gerenciamento transparente de execução distribuída. O ambiente é composto de três módulos principais:

(i) *VirD-Loader*: é responsável pela interpretação de arquivos descritores contendo o algoritmo a ser simulado e seu vetor de estados inicial;
(ii) *VirD-Launcher*: realiza o escalonamento e controle de fluxo de execução das tarefas;
(iii) *VirD-Exec*: controla a comunicação entre os clientes (*VirD-Clients*).

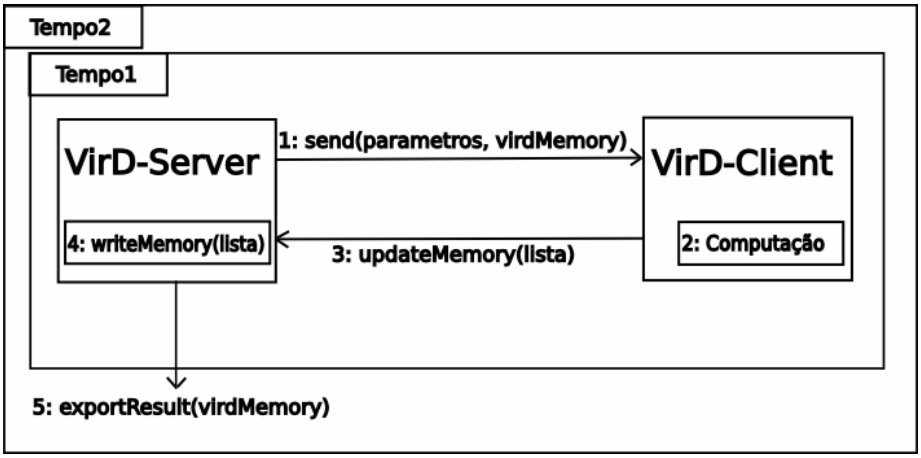


Figura 1: Estrutura do Ambiente Vird-GM.

Na Figura 1 temos a estrutura de execução do ambiente *VirD-GM*. A simulação distribuída de um algoritmo quântico a partir de *MPPs* consiste no envio de cada *MPP* para um *VirD-Client* pela aplicação do método *send*. Ao passar os parâmetros do *MPP* e a porção necessária da memória atual, o *VirD-Client* realiza a computação e gera uma lista com os valores parciais calculados, a qual é enviada para o *VirD-Server* através do método *updateMemory*. E assim, o *VirD-Server* atualiza a sua memória com os dados contidos nesta lista.

As transformações quânticas a serem simuladas são gerados em um arquivo descritor. Muitos parâmetros para simulação são estáticos e podem ser gerados na construção do descritor, como por exemplo, o particionamento das memórias de leitura e escrita para *MPPs.*

A geração do arquivo descritor foi modificada para incluir uma lista dos controles, informando a posição e o valor dos controles da transformação. A partir desta lista é gerada duas máscaras (cadeia de binários), uma com as posições do controle definidas como 1 e a outra com os valores dos controles nas posições.

O ambiente *VirD-GM* foi estendido para receber estes novos parâmetros na forma de *Strings* que precisam ser interpretadas e transformadas em números inteiros, que internamente são representados por números binários, que facilita a comparação das máscaras.

Para manter a compatibilidade com o ambiente *VirD-GM*, desenvolvido em *Java*, as chamadas ao *CUDA Kernel* foram implementadas utilizando a biblioteca *JCUDA* (YAN et al., 2009). O *CUDA Kernel*, responsável pelos cálculos dos resultados das transformações, recebe estes parâmetros já tratados e faz uma comparação com o identificador do *thread* para verificar se os valores a serem calculados satisfazem a condição de controle. Caso não haja posições válidas a memória de entrada é replicada na memória de saída.

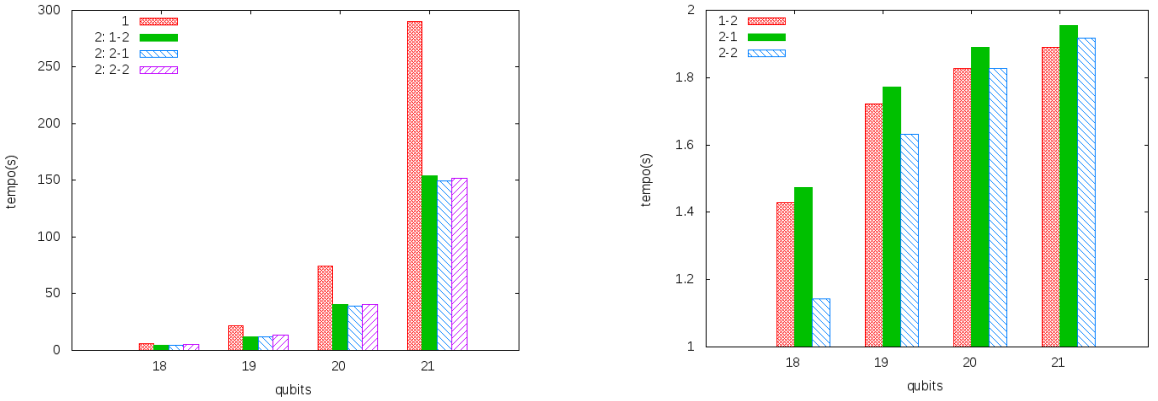
Em otimizações realizadas anteriormente no *CUDA Kernel*, por questões de desempenho, definiu-se que cada *thread* calcula 4 posições. Dessa forma, quando a comparação com a máscara indica que existem posições válidas para o cálculo, após todos os valores terem sido gerados, uma nova comparação é feita para verificar quais posições calculadas pelo thread devem ser escritas na memória de saída. As demais posições são copiadas da memória de entrada.

3. RESULTADOS E DISCUSSÃO

Para validação e teste da implementação foram utilizados como estudo de caso transformações controladas com alvos em *Hadamards* de até 21 *qubits*. A escolha da transformação *Hadamard* é justificada por ser uma matriz densa que gera uma maior carga de execução no ambiente de simulação. A metodologia para a simulação considera a execução de 40 simulações de cada instância.

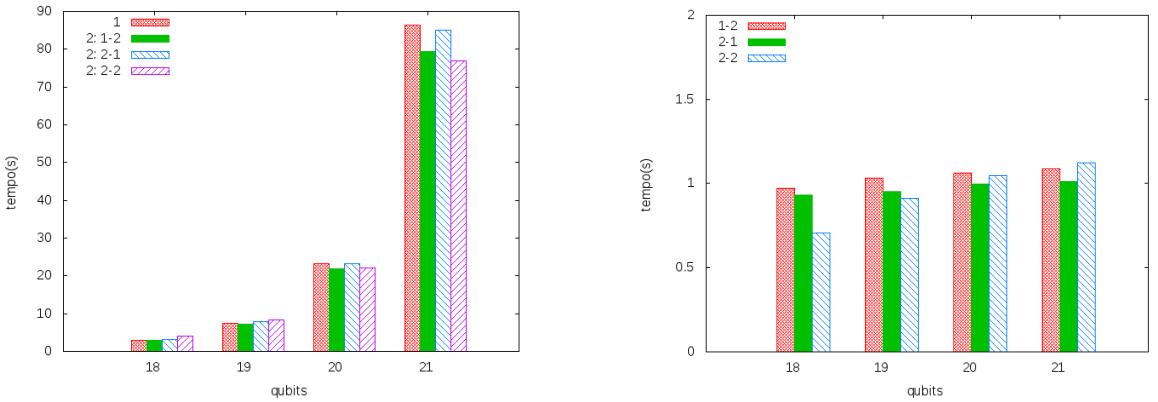
Os testes foram realizados em duas máquinas Intel Core i7, 8 GiB de RAM e GPU NVIDIA GT640. Uma máquina adicional, conectada ao *cluster* por uma rede *Fast Ethernet* foi utilizada para executar o servidor do *VirD-GM*.

As configurações de *MPPs* usadas na distribuição são descritas na forma x:*n*-*m* onde *x* é o número de clientes, *n* é o número de partes em que a memória de leitura foi particionada e *m* é o número de partes em que a memória de escrita foi particionada.



(a) Tempos de simulação. (b) *Speedup* Relativo.

Figura 2. Resultados para a simulação distribuída entre 2 clientes com controle no último *qubit*.



(a) Tempos de simulação. (b) *Speedup* Relativo.

Figura 3. Resultados para a simulação distribuída entre 2 clientes com controle no primeiro *qubit*.

Como podemos observar na Figura 2(a), a distribuição das simulações com controle no último *qubit* em 2 clientes obteve resultados significativos comparados à 1 cliente. Chegando próximo do *speedup* relativo ideal para a distribuição entre 2 clientes, como demonstrado na Figura 2(b). O *speedup* relativo apresenta um comportamento crescente, justificado pelo aumento da complexidade das simulações, tornando o tempo de inicialização irrelevante para o tempo total.

Na distribuição das simulações com o controle no primeiro *qubit*, como observado na Figura 3(a) os resultados da distribuição entre 2 clientes ficaram próximos dos resultados para 1 cliente. Nos casos de menor complexidade, o o tempo de simulação para 2 clientes foi maior que para 1 cliente, como observado no *speedup* relativo na Figura 3(b). Esse resultado é justificado pela forma que a matriz é dividida quando o controle encontra-se no primeiro *qubit*, gerando um desbalanceamento de carga entre os clientes.

4. CONCLUSÕES

O trabalho colaborou para consolidar a dinâmica de execução do ambiente *VirD-GM* e a integração com outras frentes de pesquisa (AVILA et al., 2015). A principal contribuição proposta pela extensão do ambiente *VirD-GM* é prover uma solução para o suporte a transformações controladas através da simulação distribuída/paralela através de *clusters* de *GPUs* utilizando *MPPs*. Os resultados alcançados identificam-se com a atual e crescente busca de otimizações para consumo de memória e desempenho de algoritmos quânticos.

5. REFERÊNCIAS BIBLIOGRÁFICAS

AVILA, A.; SCHMALFUSS, M.; REISER, R.; KREINOVICH, V. Fuzzy Xor Classes from Quantum Computing. In: **Artificial Intelligence and Soft Computing - 4th International Conference**, ICAISC 2015, Proceedings, Part II. Zakopane, Poland: NY Springer Series, Advances in Intelligent and Soft Computing, 2015. p.1–12.

AVILA, A.; MARON, A.; REISER, R.; PILLA, M.; YAMIN, A. GPU-Aware Distributed Quantum Simulation. In: **ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING**, 29., Gyeongju, 2014. p.860-865.

GROVER, L. K. **A Fast Quantum Mechanical Algorithm for Database Search**. New York, NY, USA: ACM, 1996. 212–219p.

NIELSEN, M. A.; CHUANG, I. L. **Quantum Computation and Quantum Information**. 10.ed. New York, NY, USA: Cambridge University Press, 2011.

MARON, A.; REISER, R.; PILLA, M. High-Performance Quantum Computing Simulation for the Quantum Geometric Machine Model. In: **INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 13**. 2013. Proceedings... ACM, 2013. p.474–481.

PORTUGAL, R.; LAVOR, C.; MACULAN, N. **Uma Introdução à Computação Quântica**. SP: Notas em Matemática Aplicada - SBMAC, 2004.

SHOR, P. W. Polynomial-time algoritms for prime factorization and discrete logarithms on a quantum computer. **SIAM Journal on Computing**, Philadelphia, PA, USA, v.26, n.5, p.1484–1509, Oct. 1997.

YAN, Y.; GROSSMAN, M.; SARKAR, V. JCUDA: A Programmer-Friendly Interface for Accelerating Java Programs with CUDA. In: **INTERNATIONAL EURO-PAR CONFERENCE ON PARALLEL PROCESSING, 15**. 2009, Berlin, Heidelberg. Proceedings... Springer-Verlag, 2009. p.887–899.