SIMULAÇÃO PARALELA DOS ALGORITMOS DE SHOR E DE GROVER NO SIMULADOR D-GM

ANDERSON BRAGA DE AVILA[1];
RENATA HAX SANDER REISER[1]; MAURICIO LIMA PILLA[1]

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

1. INTRODUÇÃO

A Computação Quântica (CQ) introduz um novo paradigma na ciência da computação, afirmando que os algoritmos quânticos podem desenvolver tarefas mais complexas se comparadas com paradigma de programação atual, como no processamento da informação e criptografia (GROVER, 1996; SHOR 1997). Mas enquanto os computadores quânticos ainda não estão disponíveis, o estudo e desenvolvimento de algoritmos quânticos podem ser feito a partir da descrição matemática ou softwares de simulação. Uma vez que a simulação quântica realizado por computadores clássicos exige muito tempo e elevados recursos computacionais (processos e/ou memória) a pesquisa sobre arquiteturas paralelas pode fornecer possíveis melhorias de desempenho para novos algoritmos quânticos.

Neste contexto, este trabalho contribui com o desenvolvimento do ambiente D-GM, incrementando suas capacidades de simulação usando CPUs e GPUs. Mais especificamente, a principal contribuição se dá pela extensão das otimizações já implementadas no D-GM e pela simulação de dois importantes algoritmos quânticos: Shor e Grover. Também é realizado a comparação dos resultados alcançados com o simulador do grupo de pesquisa *Quantum Architetures and Computation Group (QuArC)* da Microsoft chamado LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing (WECKER, 2014).

2. METODOLOGIA

Em trabalhos anteriores (AVILA, 2015) foram apresentadas otimizações para a simulação de algoritmos quânticos com o objetivo de reduzir a complexidade espacial e temporal relacionadas a aplicações multi-qubits, as quais foram implementados para execuções em GPU e apresentaram bons resultados.

Estas otimizações foram estendidas e adapatadas para dar suporte também há execução em CPUs. Os melhores resultados considerando a abordagem de decomposição quanto ao limite de operadores por passo foi de 1 e 2. Logo, a opção de calcular operador por operador foi escolhida, pois permite a classificação dos operadores em duas classes e consequentemente a aplicação de algumas otimizações:

- **Densos -** operadores definidos por matrizes sem elementos nulos, como o operador Hadamard. Este operadores não permitem a aplicações de otimizações mais severas.
- **Esparsos -** operadores com elementos não nulos apenas na diagonal principal, como o operador Pauly Y, ou na diagonal secundária, como o operador Pauly X. Nestes casos é possível realizar otimizações descartando os cálculos que envolveriam elementos nulos, sem modificar o resultado.

Seguindo a abordagem descrita em (AVILA, 2015) para o cálculo de TQs decompostas evitando a replicação de dados gerados pelo produto tensor com o operador Identidade, operadores densos são calculados é exemplificado na Eq. 1 e operadores esparsos nas Eq. 2 e 3, nestas equações é considerado a aplicação de um operador genérico ao primeiro qubit de um estado com dois qubits.

$$\begin{pmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{pmatrix} \times \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} a_{00} \times m_{00} + a_{10} \times m_{01} \\ a_{01} \times m_{00} + a_{11} \times m_{01} \\ a_{00} \times m_{10} + a_{10} \times m_{11} \\ a_{01} \times m_{10} + a_{11} \times m_{11} \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} m_{00} & 0 \\ 0 & m_{11} \end{pmatrix} \times \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} a_{00} \times m_{00} \\ a_{01} \times m_{00} \\ a_{10} \times m_{11} \\ a_{11} \times m_{11} \end{pmatrix} \quad (2)$$

$$\begin{pmatrix} 0 & m_{01} \\ m_{10} & 0 \end{pmatrix} \times \begin{pmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{pmatrix} = \begin{pmatrix} a_{10} \times m_{01} \\ a_{11} \times m_{01} \\ a_{00} \times m_{10} \\ a_{01} \times m_{10} \end{pmatrix} \quad (3)$$

A execução sequencial de uma TQ operador por operador foi implementada da seguinte forma: para cada um, seu tipo é identificado e então o correspondente *loop* é executado, onde cada iteração realiza o cálculo de uma nova amplitude. A execução paralela em CPU foi implementada usando OpenMP, adicionando a diretiva “parallel for” nos *loops*, assim cada thread calcula $2^n/th$ amplitudes novas, sendo *n* o número de qubits e *th* o número de threads.

3. RESULTADOS E DISCUSSÃO

A principal contribuição deste trabalho pode ser validada pelas simulações em CPU, sequenciais e paralelas com até 4 threads, e GPU dos algoritmos de Shor, de 15 até 25 qubits, e de Grover, de 15 até 21 qubits. Bem como pela comparação destes resultados com o simulador LIQUi|>.

Os testes foram realizados em um desktop com processador Intel Core i7-3770, 8 GB RAM, GPU NVidia Titan com sistema operacional Ubuntu Linux 14.04, 64 bits, com CUDA Toolkit 7.0.

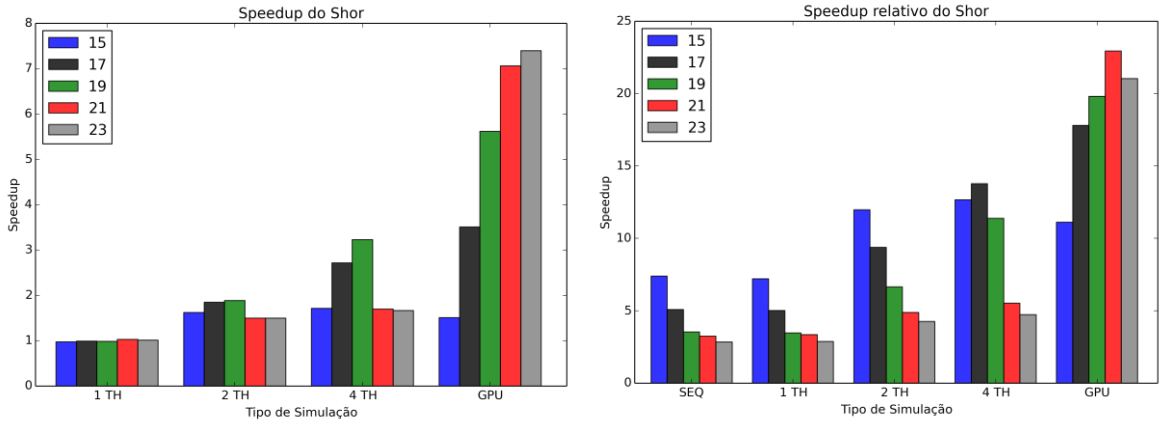
Os tempos médios de execução, para todas as simulações propostas, foram obtidos depois de 10 execuções. Os desvios padrões foram de menos de 1% para a maioria dos casos, com um máximo de 3%.

Para o algoritmo de Shor os tempos de execução no LIQUi|> foram obtidos do “minutes for running” presente no arquivo de *log* gerado após executar o seu Shor integrado, o qual é baseado no mesmo circuito usado pelo simulador D-GM, com a opção de otimização “true”.

A Figura 1(a) mostra os speedups do D-GM em relação a sua simulação sequencial considerando uma faixa de qubots representados em colunas. O

overhead gerado pela simulação paralela com 1 thread não se mostrou significante. Os speedups escalaram melhor com o aumento do número de threads para simulações de 17 à 19 qubits, enquanto de 21 à 23 qubits ficaram limitados pelo aumento exponencial da memória e do número de operações. Simulações com 15 qubits sofreram impacto do overhead inicial da aplicação gerados por criação de estruturas e comunicação, uma vez que este número de qubits é pequeno. Todas as simulações apresentaramgrande escalabilidade para simulações em GPU, exceto pela de 15 qubits. Com 23 qubits, o desempenho foi 7,39x mais rápido que a versão sequencial.

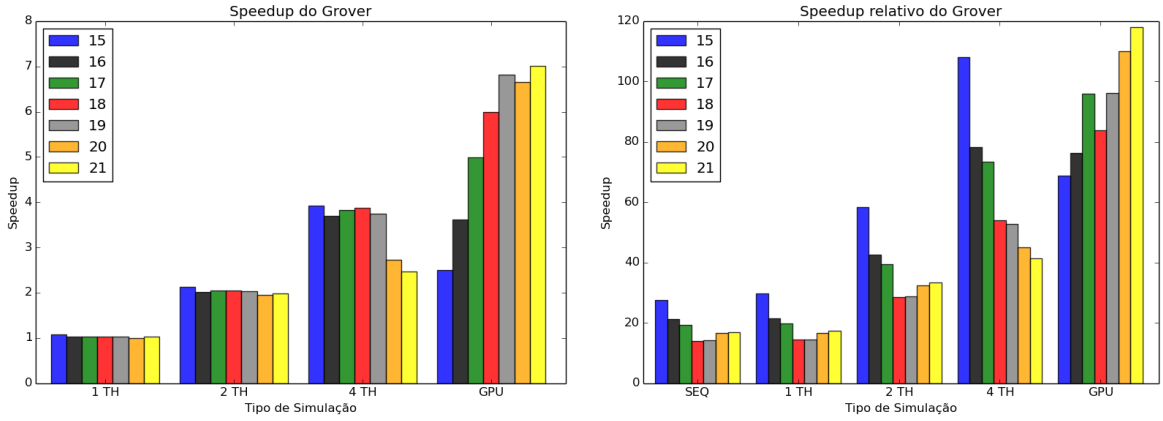
Até mesmo a versão sequencial do D-GM foi mais rápida que o simulador LIQUi|>, como pode ser visto na Figura 1(b) onde é mostrado os speedups do D-GM em relação ao tempos obtidos no LIQUi|>. O melhor caso foi para 21 qubits, onde nosso simulador foi 21,94x mais rápido. Logo, todas as execuções no D-GM obtiveram bons speedups relativos quando comparados com o LIQUi|> para todas as simulações em CPU com diferentes números de threads e em GPU. Novamente, embora a escalabidade para números de qubits pequenos tenha sido melhor nas simulações multi-core, existe uma tendência a melhores speedups para mais qubits nas simulações em GPU.



(a) (b)

Figura 1: Speedups relativos aos tempos de simulação do algoritmo de Shor.

Para as simulações do algoritmo de Grover, o mesmo circuito foi implementado para ambos os simuladores usando como oráculo uma TQ N-Controladas para achar um dado elemento na base de dados.



(a) (b)

Figura 2: Speedups relativos aos tempo de simulação do algoritmo de Grover.

A Figura 2(a) mostra o speedup do D-GM em relação a sua versão sequencial e apresentou melhor escalabildade que as simulações do algoritmo de Shor, e em alguns casos aproximou-se do caso ideal. A principal diferença foi que os speedups escalaram bem até mesmo para casos com poucos qubits. Este comportamento está diretamente relacionado a quantidade de operadores ser muito maior que a do algoritmo de Shor, diluindo assim o overhead. Novamente, quando as aplicações se tornam suficenteme grandes, speedups com mais threads sofrem com *misses* na cache.

A Figura 2(b) mostra que nosso simulador obteve speedups ainda melhores quando comparado aos resultados no LIQUi|>. No melhor caso, nossa simulação usando GPU foi 118x mais rápida com 21 qubits, e para todas as simulações o speedup relativo foi de pelo menos 13,9x.

4. CONCLUSÕES

Neste trabalho, otimizações de simulação de computação quântica foram aplicadas na execução de dois importantes algoritmos quânticos: Shor e Grover. Pelo uso das otimizações descritas foi possível simualdor aplicações com um grande número de qubits em um periodo curto de tempo, aumentando assim as capacidades do nosso simulador.

Ambos simulações dos algoritmos de Shor e de Grover mostraram melhores resultados no simulador D-GM quando comparados ao simulador LIQUi|>. Para o maior número fatorado(10 bits) no algoritmo de Shor, a execução sequencial em CPU foi 2,84x mais rápida, a execução paralela em CPU 4,72x mais rápida com 4 threads, e a execução em GPU 21,03x mais rápida. Para o algoritmo de Grover, o D-GM obeteve seu melhor speedup para a execução em GPU com 21 qubits, sendo 118x mais rápido que o LIQUi|>. Simulações com mais qubits no simualdor LIQUi|> começaram a se tornar inviáveis uma vez que o tempo de compilação do circuito chegou a 32 horas para 21 qubits. Uma vantagem adicional da nossa abordagem é que circuitos quânticos não requerem altos tempos de compilação.

Em trabalhos futuros considera-se a extensão no simulador D-GM para suporte a simulação distribuída, aproveitando ambas unidades de processamento, CPU e GPU.

5. REFERÊNCIAS BIBLIOGRÁFICAS

GROVER, L. A fast quantum mechanical algorithm for database search.In: Proc. of the Twenty-Eighth Annual ACM Symp.on Theory of Computing, p. 212–219, 1996.

SHOR, P. Polynomial-time algoritms for primefactorization and discrete logarithms on a quantumcomputer. SIAM Journal on Computing, 1997.

AVILA, A.; REISER, R.; PILLA, M. Otimização de simulação de computação quântica através da redução e decomposição baseados no operador identidade. Em Anais de WSCAD, páginas 1–12, 2015.

WECKER, D.; SVORE, K. M. LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing, 2014. Disponível em: http://arxiv.org/abs/1402.4467.