Extraindo o Paralelismo da Biblioteca qGM-Analyzer

Murilo F. Schmalfuss, Daniel K. Retzlaff, Adriano K. Maron, Maurício L. Pilla, Renata S. Reiser

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPel)
Caixa Postal 354 – 96.001-970 – Pelotas – RS – Brasil

{mfschmalfuss,dkretzlaff,akmaron,reiser,pilla}@inf.ufpel.edu.br

1. Introdução

Os avanços na Computação Quântica (*CQ*) estão viabilizando a construção do *hardware* quântico. Apesar da constante evolução, esses sistemas ainda possuem baixa escalabilidade, não suportando sistemas quânticos complexos. Dessa forma, é necessário utilizar simuladores, baseados em computadores clássicos, para a análise e validação de algoritmos quânticos. Modelos e simuladores têm sido propostos [Quantiki 2009], porém algumas abordagens mostram-se ineficazes, não oferecendo suporte a sistemas quânticos mais complexos.

A *CQ* é baseada nos postulados da Mecânica Quântica [Knill and Nielsen 2002]. A unidade básica de informação quântica é o *qubit*, sendo definido por um vetor de estado, unitário e bidimensional, descrito genericamente, na notação de Dirac, pela expressão $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, satisfazendo a condição de normalização $|\alpha|^2 + |\beta|^2 = 1$. O espaço de estados de múltiplos *qubits* é composto pelo produto tensor dos espaços de estados dos sistema componentes. Por exemplo um sistema de dois *qubits* $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ e $|\varphi\rangle = \gamma|0\rangle + \delta|1\rangle$, o espaço de estados composto é o produtos tensor $|\psi\rangle \otimes |\varphi\rangle$.

A implementação de uma biblioteca para simulação de algoritmos quânticos deve seguir esses e outros postulados [Nielsen and Chuang 2003], além de determinadas convenções para uma representação consistente.

2. Propostas de Otimização

O objetivo deste trabalho é apresentar propostas de otimizações para a biblioteca *qGM-Analyzer*, considerando a paralelização e uma extensão para a linguagem de programação C++.

O ambiente *VPE-qGM* (*Visual Programming Environment for the qGM Model*) [Maron et al. 2010] está em desenvolvimento para suporte a modelagem e simulação, sequencial e distribuída, de algorimos da *CQ*, considerando as abstrações do modelo *qGM* (*Quantum Geometric Machine*) [Reiser and Amaral 2010].

A biblioteca *qGM-Analyzer* substitui as matrizes associadas as transformações unitárias e o operador de produto tensorial por conjuntos de funções elementares e por uma lista de tuplas (*Lvpp*), a qual armazena produtos parciais dessas funções, como visto na Tabela 1.Uma função recursiva faz uso dessas estruturas para gerar dinamicamente cada elemento associado a um vetor componente.

A possibilidade de otimização da *qGM-Analyzer* reside na paralelização da função recursiva, a qual representa o maior custo de execução da biblioteca.

- -	$x_1 = 0$ $x_2 = 0$	$x_1 = 0$ $x_2 = 1$	$x_1 = 1$ $x_2 = 0$	$x_1 = 1$ $x_2 = 1$
$H(1, x_1) \times H(0, x_2)$	$(\frac{1}{2}, 0)$	$(\frac{1}{2}, 1)$	$(-\frac{1}{2}, 2)$	$(-\frac{1}{2}, 3)$
$H(0, x_1) \times H(0, x_2)$	$(\frac{1}{2}, 0)$	$(\frac{1}{2}, 1)$	$(\frac{1}{2}, 2)$	$(\frac{1}{2}, 3)$
$Id(1, x_1) \times Id(1, x_2)$	$-$	$-$	$-$	$(1, 3)$

Tabela 1. Lvpp para geração dos elementos da linha 35 de $H^{\otimes 4} \times Id^{\otimes 2}$

2.1. Estendendo a biblioteca para linguagem C++

As otimizações recentemente implementadas na biblioteca *qGM-Analyzer* [Maron et al. 2011] resultaram em ganho no consumo de memória quando da simulação de sistemas quânticos com múltiplos *qubits*. Entretanto, o tempo total de simulação obtido permanece elevado, devido a quantidade de operações necessárias para simular uma transformação quântica. Portanto, uma extensão da biblioteca *qGM-Analyzer*, considerando a linguagem C++, é proposta. Tal escolha se justifica pelo conhecido desempenho dessa linguagem, juntamente com a possibilidade de integração com módulos para processamento paralelo, tais como *OpenMP* [Ayguade and Chapman 2003] e *CUDA* [NVIDIA 2009]. Propostas de integração com esses módulos são apresentadas nas seções seguintes.

2.2. Protótipo da qGM-Analyzer Utilizando OpenMP

A busca pela integração da *qGM-Anlyzer* com a biblioteca *OpenMP* se justifica, em primeira instância, pela crescente disponibilidade de máquinas *multi-core*. Ainda, tem-se como motivação secundária a forma de funcionamento da *qGM-Analyzer*, a qual pode ter sua execução paralelizada de forma simplificada através da diretiva **REDUCTION**, presente no *OpenMP*.

Com essa estruturação, cada *thread* executa parte das gerações de valores, as correspondentes multiplicações pelas posições de memória e acumulação de seus resultados locais. Por fim, a diretiva **REDUCTION** realiza o somatório de todos os resultados individuais obtidos pelas *threads*. Finalizando, a *thread master* retorna o valor para atualização na posição de memória correspondente ao PE em execução. Um algoritmo estruturado descrevendo esse funcionamento é visto no Algoritmo 1.

ParallelApplyCall
Input: Lvpp, sizesList,memory,position
$result \leftarrow 0$;
$\#pragma\ omp\ parallel\ for\ \ shared(Lvpp, sizesList, memory)\ \backslash$
$reduction(+ : result)$
for $i \leftarrow 0$ **to** $size(Lvpp[0])$ **do**
  $partial \leftarrow ApplyValues(Lvpp, sizesList, memory, 1, 1, 0, 0)$;
end
$memory[position] \leftarrow result$;

Algoritmo 1: Chamada para execução paralela da função ApplyValues

ApplyValues
Input: Lvpp, sizesList,memory,value,startPos,index,res
for $i \leftarrow 1$ **to** $size(Lvpp[index])$ **do**
  if $index == size(Lvpp) - 1$ **then**
    $pos \leftarrow startPos + Lvpp[index][i][1]$;
    $res+ = (value \times Lvpp[index][i][0] \times memory[pos])$;
  end
  else
    $res = ApplyValues(Lvpp, sizesList, memory, value \times Lvpp[index][i][0], index + 1, startPos + (Lvpp[index][i][1] \times sizesList[index]), res)$;
  end
end
Output: res

Algoritmo 2: Execução de cada thread do algoritmo 1

Percebe-se que a recursão ainda se faz presente na biblioteca para execução da parte sequencial de cada *thread*. O grão da tarefa de cada *thread* pode ser manipulado, inclusive dinamicamente, por um parâmetro da *qGM-Analyzer* que define a quantidade de tuplas armazenadas em cada linha do *Lvpp*. Em uma primeira análise, pode-se definir que a quantidade de *threads* a serem executadas, de acordo com a estrutura do *Lvpp*, é igual a quantidade de tuplas armazenadas na primeira linha da estrutura. Obviamente, tal mapeamento deve considerar, *a priori*, a quantidade de núcleos de processamento disponíveis no *PC/Cluster*, visando uma carga de trabalho balanceada.

2.3. Protótipo da qGM-Analyzer Utilizando CUDA

A perspectiva de integração da biblioteca *qGM-Analyzer* com a plataforma *CUDA* visa a exploração da arquitetura massivamente paralela presente nas *GPUs*. Devido ao crescimento exponencial na quantidade de operações necessárias para simular algoritmos quânticos com muitos *qubits*, é prudente considerar que *CPUs* tradicionais não oferecerão o desempenho exigido para essas tarefas. A quantidade de *stream processors* contidas nas *GPUs* permitem uma melhor escalabilidade a um custo mais baixo.

Entretanto, o mapeamento de tarefas para execução nas *GPUs* não é uma tarefa trivial, especialmente quando o algoritmo sequencial é definido recursivamente – caso da *qGM-Analyzer*. Dessa forma, faz-se necessário substituir a recursão por estruturas iterativas. Estudos estão sendo iniciados nessa área, buscando identificar as aborgadens mais eficientes para tal integração.

Inicialmente, busca-se a execução paralela de vários PEs na *GPU*. Cada um desses PEs também possui um certo grau de paralelismo interno na sua biblioteca de execução que também pode ser explorado, como mostrado na implementação em *OpenMP*. Essa estratégia visa, a longo prazo, possibilitar a distribuição de vários PEs ao longo de um *cluster*. Em cada *cluster*, uma fração do total de PEs é executado em paralelo através de *GPUs*.

3. Conclusões

O desenvolvimento da bilioteca *qGM-Analyzer* em C++ viabiliza uma execução mais rápida da parte crítica do ambiente *VPE-qGM*. Mais significativo, essa implementação caracteriza os estágios iniciais de um importante processo de extensão da biblioteca *qGM-Analyzer*, buscando a possibilidade de execução distribuída em diferentes arquiteturas.

A implementação utilizando a biblioteca *OpenMP* está nos estágios iniciais de desenvolvimento, no qual estão sendo portados os métodos de *parser* dos parâmetros dos processos, da geração da estrutura do *Lvpp* e da função *ApplyValues*.

A implementação em *CUDA* oferece grande poder de processamento, entretanto a simulação de sistemas quânticos com muitos *qubits* fica limitado pela quantidade de memória disponível na *GPU*. O mapeamento da estrutura do *Lvpp* e o acesso constante aos dados armazenados em memória são os principais desafios a serem superados nessa implementação.

Referências

Ayguade, E. and Chapman, B. (2003). Introduction: Special issue: OpenMP. *Scientific Programming*, 11(2):79–80.

Knill, E. H. and Nielsen, M. A. (2002). Theory of quantum computation. http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0010057.

Maron, A., Pinheiro, A., Reiser, R., and Pilla, M. (2010). Modelagem e simulação quântica no ambiente VPE-qGM. In *Workshop Escola de Computação e Informação Quântica*, pages 121–130.

Maron, A., Ávila, A., Reiser, R., and Pilla, M. (2011). Introduzindo uma nova abordagem para simulação quântica com baixa complexidade espacial. In *Anais do DINCON 2011*, pages 1–6. SBMAC.

Nielsen, M. A. and Chuang, I. L. (2003). *Computação Quântica e Informação Quântica*. Bookman.

NVIDIA (2009). *CUDA Programming Guide*. NVIDIA Corp. Version 2.3.

Quantiki (2009). Quantum information wiki and portal. Disponível por WWW em `http://www.quantiki.org` (jun.2010).

Reiser, R. and Amaral, R. (2010). The quantum states space in the qgm model. In *Anais/III Workshop Escola de Computação e Informação Quântica*, pages 92–101, Petrópolis/RJ. Editora do LNCC.