Uma Abordagem para o *Knight's Tour Problem* utilizando *Depth First Search*

Bruno Kenzo Yui, Gabriel Andreazi Bertho, Karen Ketlyn Ferreira Barcelos, Rodrigo Takizawa Yamauchi, Helio Crestana Guardia

Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – São Carlos – SP – Brazil

karenkfbarcelos@gmail.com,
brunoyui@estudante.ufscar.br,gabrielbertho@estudante.ufscar.br
rodrigo.yamauchi@estudante.ufscar.br, helio.guardia@ufscar.br

Abstract. *The Knight's Tour is a well-known problem in graph theory and combinatorial optimization, where a knight moves across a chessboard visiting each square exactly once. This paper presents an approach based on Depth First Search (DFS) to explore possible knight's tours, analyzing its computational complexity and efficiency. We also discuss heuristic optimizations, such as Warnsdorff's Rule, to improve search performance. The results demonstrate the feasibility of using DFS for solving this NP-hard problem.*

Resumo. *Knight's Tour é um problema clássico da teoria dos grafos e da otimização combinatória, no qual um cavalo deve percorrer um tabuleiro de xadrez e visitar cada casa exatamente uma vez. Este artigo apresenta uma abordagem para o problema baseada em Depth First Search (DFS) e outra usando método de Monte Carlo para explorar possíveis passeios do cavalo, e analisa suas complexidades computacionais e eficiências. Além disso, discutimos otimizações heurísticas, como a Regra de Warnsdorff, para melhorar o desempenho da busca. Os resultados demonstram a viabilidade do uso de DFS para resolver esse problema NP-difícil.*

1. Introdução

Knight 's Tour (Problema do Passeio do Cavalo) é um problema de otimização que exige que um cavalo (peça de xadrez) percorra um tabuleiro de xadrez N×N, visitando cada casa exatamente uma vez.

Analisando o problema como um grafo para a busca dos caminhos, cada posição no tabuleiro pode ser representada como um nó no grafo, enquanto cada movimento possível do cavalo pode ser representado como uma aresta. Assim, para resolver o problema paralelamente, é necessário abordar um caso especial de Busca em Profundidade: DFS - *Depth First Search*. Seu objetivo é explorar o máximo possível em profundidade, conectando o maior número de nós no grafo e ramificando-se quando necessário. Além disso, foi utilizada uma heurística conhecida como a Regra de Warnsdorff, que faz uma análise em cada movimento do cavalo, escolhendo a casa do tabuleiro que terá o menor número de movimentos futuros.

Este artigo propõe uma abordagem baseada em DFS e em um método de Monte Carlo, paralelizando-a com OpenMP e CUDA para analisar o ganho de desempenho obtido em ambientes multi core e GPUs.

2. Questões conceituais essenciais do problema

A abordagem deste projeto utilizou inicialmente DFS sequencial, posteriormente paralelizada através de **OpenMP**: dividindo o espaço de busca em tarefas independentes exploradas por múltiplas *threads* na CPU; e **CUDA**: aproveitando o paralelismo massivo das GPUs para explorar simultaneamente várias trajetórias.

Diversos estudos abordam técnicas sequenciais, heurísticas e paralelas para resolver o *Knight's Tour Problem*. Warnsdorff (1823) propôs uma heurística que reduz a complexidade de uma busca sequencial, que pode ser utilizada para aumento de eficiência do problema a ser paralelizado. De acordo com a heurística, o cavalo deve se mover para a casa que contém o menor número de futuros movimentos, o que faz com que a solução possa ser encontrada mais rapidamente, visto que cada thread terá menos casas a serem procuradas a cada movimento. Essa heurística é utilizada em outros casos gerais de grafos, visto que um movimento na teoria dos grafos é feito para o vértice adjacente com menor grau (ou valência).

Para o estudo do caso abordado, outra técnica utilizada foi um método semelhante ao Método de Monte Carlo, que se baseia em amostragens aleatórias massivas para obter resultados numéricos. No projeto, o paralelismo massivo da GPU é explorado ao lançar milhares de *threads* simultâneas, cada uma executando um caminho aleatório do cavalo no tabuleiro. Dessa forma, combinando essas threads com a heurística de Warnsdorff, o problema pode ser resolvido paralelamente para grandes tabuleiros, com tempo de execução eficiente.

3. Desenvolvimento do trabalho

Inicialmente, utilizou-se uma solução sequencial que utiliza a técnica clássica de busca em profundidade (DFS) [8th Marathon of Parallel Programming WSCAD-SSC/SBAC-PAD-2013]. É importante notar que o objetivo dos códigos implementados neste projeto é encontrar o primeiro caminho completo do cavalo, dessa forma, o código é encerrado ao se obter esse caminho. Posteriormente, realizou-se a paralelização em dois contextos distintos, aplicando a Regra de Warnsdorff e o Método de Monte Carlo em ambos os casos:

- OpenMP: O espaço de busca foi dividido em tarefas independentes exploradas por múltiplas *threads* simultaneamente. Cada *thread* realiza uma busca independente criando seu próprio tabuleiro local, percorrendo diferentes caminhos no grafo, com a diretiva *pragma omp parallel*. Utilizou-se a diretiva *pragma omp critical* para garantir que um tour válido do cavalo foi completo.
- CUDA: Utilizou-se o paralelismo massivo característico das GPUs para executar milhares de *threads* simultaneamente, cada uma executando um caminho aleatório do cavalo no tabuleiro. O uso da Regra de Warnsdorff reduz a quantidade de explorações desnecessárias e melhora a chance de encontrar um caminho válido.

Algoritmo geral:

1. Inicializar o tabuleiro.
2. Para um dado movimento válido, se não visitado:
3. Marcar posição como visitada.
4. Chamar recursivamente a função DFS.
5. Realizar passos anteriores até o cavalo encontrar a casa inicial com todas as demais casas marcadas como visitadas.
6. Caso o movimento não tenha mais casas válidas para prosseguir e as condições em (5) não sejam satisfeitas, repetir (2) para outro movimento.
7. Encontrando validade em (5), o código retorna e termina.

Analisando o algoritmo acima, ocorre a paralelização de (2) até (6) em N *threads* para as possibilidades de movimento do cavalo, que utilizam da função DFS para recursivamente encontrar uma solução, com o auxílio da Regra/Heurística de Warnsdorff para tal. É importante ressaltar também o uso do Método de Monte Carlo, para gerar possíveis milhares de tentativas e soluções randomicamente.

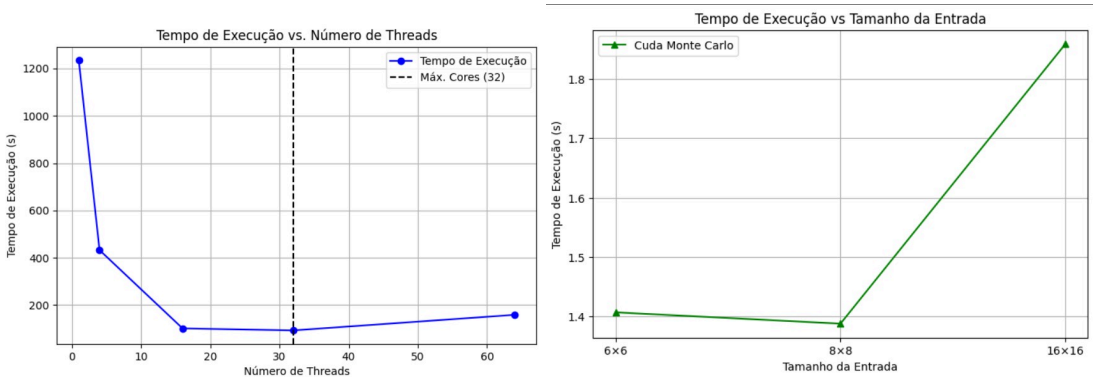
A complexidade computacional do *Knight's Tour* é exponencial ($O(k^N)$), onde k é o número das dimensões do tabuleiro e N é o número de casas), tanto para a implementação sequencial quanto para a paralela. Com a paralelização, essa complexidade é mitigada significativamente em termos de tempo, embora a complexidade espacial ainda seja mantida.

4. Resultados e Discussões

Os experimentos foram realizados utilizando um computador com 4 processadores Intel Core i7-1355U (10 núcleos) e GPU NVIDIA Tesla T4 com CUDA versão 12.4. Os testes OpenMP com um tabuleiro 10x10 foram realizados em computador Intel(R) Xeon(R) CPU E7- 4870 (80 núcleos).

No tabuleiro 8×8, a versão sequencial levou aproximadamente 21,227 segundos para encontrar uma solução, devido à natureza exponencial da DFS. Em OpenMP (CPU), avaliando diferentes quantidades de *threads* (1, 2, 4, 8, 10, 12, 16), observou-se melhora expressiva até 10 *threads*, com tempo mínimo alcançado de 0,496 segundos. O desempenho piorou levemente acima desse limite devido ao *overhead* do *hyperthreading* e à concorrência por recursos. O código em OpenMP também foi testado em uma máquina com 32 cores com um tabuleiro 10x10, alcançando o tempo de 101,75 segundos para 16 *threads* e 93,27 segundos (mínimo) para 32 *threads*.

Em CUDA (GPU), alcançou tempo de 1,388 segundos para 8×8 e crescimento moderado em tabuleiros maiores (1,859 s para 16×16), indicando boa escalabilidade.



Figuras 1a e 1b. (1a) Tempos de execução do código em CPU para tabuleiro 10x10 de acordo com o número de threads e (1b) tempo de execução do código em GPU em tabuleiro 6x6, 8x8 e 16x16, respectivamente.

Os resultados indicam que a abordagem paralela em GPU é extremamente escalável sem muitas perdas de desempenho em casos de grandes dimensões, o que se deve ao fato de milhares de *threads* serem lançadas simultaneamente para encontrar um caminho completo viável. Enquanto isso, a abordagem em CPU se mostra eficiente para tabuleiros menores e com uma quantidade de threads que seja menor que o número de núcleos do processador, evitando problemas de *overhead*. Outro ponto a ser destacado é a aleatoriedade dos tempos gerados: por conta do Monte Carlo, threads podem encontrar uma solução completa rapidamente ou demorar muito mais que o esperado para uma abordagem paralela. Dessa forma, inconsistências podem ocorrer nos resultados tanto para CPU quanto para GPU.

5. Conclusões

O estudo demonstrou que a paralelização via CPU (OpenMP) e CUDA (GPU) pode efetivamente resolver o *The Knight's Tour Problem* com ganhos substanciais de desempenho abordando heurísticas específicas do problema e métodos probabilísticos (Monte Carlo), enquanto a implementação via GPU se mostra mais eficiente para tabuleiros maiores. Sugere-se explorar futuramente outras estratégias heurísticas e abordagens híbridas que possam potencializar ainda mais o desempenho em arquiteturas de alto desempenho, como o uso de Beam Search.

7. Referências

Warnsdorff, H.C. (1823). Des Rösselsprungs einfachste und allgemeinste Lösung. Schmalkalden. Schwenk, A. (1991). Which rectangular chessboards have a knight's tour? *Mathematics Magazine*, 64(5), 325-332.

Demaine, E.D., Demaine, M.L., Kominers, S.D., Lynch, J., & Lynch, N. (2022). Taming the Knight's Tour: Minimizing Turns and Crossings. *Theoretical Computer Science*, 921, 88–97.

Conrad, K., Hindman, N., & Schaar, J. (2005). Optimal Algorithms for Constructing Knight's Tours on Arbitrary n × m Boards. *Discrete Applied Mathematics*, 145(2), 236–243.