XXIII Seminário de Iniciação Científica
XX Jornada de Pesquisa
XVI Jornada de Extensão
V Mostra de Iniciação Científica Júnior
V Seminário de Inovação e Tecnologia

Modalidade do trabalho: Relatório técnico-científico
Evento: XX Jornada de Pesquisa

PARALELIZAÇÃO DE ALGORITMO DE INSPEÇÃO DE ROTAS UTILIZANDO PERMUTAÇÃO LEXICOGRÁFICA[1]

Jessica De Almeida Berlezi[2], Janiel Ceretta Foletto[3], Edson Luiz Padoin[4], Rogério S. M. Martins[5].

[1] Trabalho realizado no curso de Bacharelado em Ciência da Computação da Unijuí na Disciplina de Processamento Paralelo no contexto do projeto Sistema de Decisão Adaptativo Baseado em Redes Neurais para Uma Arquitetura Aplicada a Agricultura de Precisão.
[2] Bolsista PIBIC/Unijuí, Aluna do curso Ciência da Computação da Unijuí, jessica.berlezi@gmail.com
[3] Aluno do curso de Ciência da Computação da Unijuí, janielfoletto@gmail.com
[4] Professor Orientador, padoin@unijui.edu.br
[5] Professor Pesquisador, rogerio.martins@unijui.edu.br

Introdução

A teoria dos grafos é uma área da matemática que estuda as relações entre os objetos de um determinado conjunto. Um grafo é uma estrutura abstrata que representa um conjunto de elementos denominados vértices e suas relações de interdependências ou arestas (GOLDBARGD, 2012).

O problema da inspeção de rotas (PIR), ou ainda, Problema do carteiro chinês (PCC) consiste em encontrar o caminho mais curto de tal forma que cada aresta seja atravessada pelo menos uma vez. Sendo análogo ao problema enfrentado por um carteiro que deve entregar a correspondência ao longo de cada vértice de um gráfico e retornar ao seu ponto de partida (EDMONDS, 1973).

O objetivo deste trabalho é desenvolver um algoritmo que receba pontos de uma matriz, ou mapa, e calcule vetores na forma de um circuito euleriano (com todas as arestas visitadas) paralelo. Este trabalho ainda propõe uma metodologia diferenciada, onde não será definido um ponto de origem inicial. O melhor ponto de origem para um determinado grafo será apresentado pelo algoritmo.

Com esta nova implementação a complexidade do problema é reduzida de exponencial para fatorial. Em outras palavras, o esforço computacional para a sua resolução cresce em ordem fatorial com o tamanho do problema (dado pelo número de arestas a serem atendidas).

Problemas assim demandam de grandes recursos computacionais independente do poder de processamento dos atuais processadores (YAMIN, 2006). Nesse contexto, para reduzir o tempo de execução, a técnica de paralelização tem sido empregada.

Metodologia

Modalidade do trabalho: Relatório técnico-científico
Evento: XX Jornada de Pesquisa

A realização desta pesquisa está dividida em três etapas. A primeira etapa consiste na implementação do algoritmo sequencial do PIR, onde é proposto um modelo baseado em permutações. A segunda etapa trata da paralelização da implementação do PIR, conceito empregado em soluções para o processamento numéricamente intensivo. Por fim, a terceira etapa, busca analisar se a paralelização tornou o processamento do PIR mais rápido.

Esta estrutura foi construída por vetores/matrizes de valores inteiros desenvolvidos em linguagem de programação C. O sistema operacional utilizado foi Ubuntu 14.04.2 LTS COM Kernel VERSÃO 3.13.0-53-generic. O equipamento possui processador Intel x64 Pentium T4200 Dual Core com tecnologia HT, o qual possibilita a execução de 2 threads. A máquina ainda conta com 4GB de memória.

Resultados e Discussões

Nesta seção será apresentado um estudo sobre grafos e seu uso no PIR, seguido da diagramação lexicográfica utilizada. Posteriormente, o algoritmo paralelizado e seus resultados serão apresentados.

Um grafo é uma estrutura de abstração bastante útil na representação e solução de diversos tipos de problemas. Matematicamente, um grafo formaliza relações de interdependência existente entre elementos de um conjunto. Os elementos do conjunto são denominados nós ou vértices. As relações entre os elementos do conjunto são denominadas arestas ou arcos (GOOLBARG, 2012).

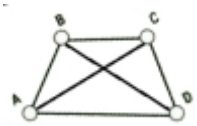
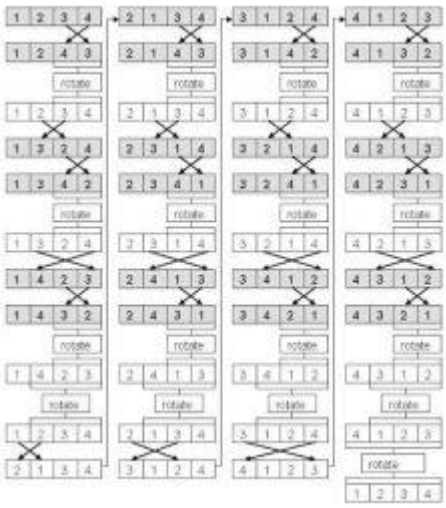


Figura 1 - Representação de um grafo com 4 arestas e 6 vértices (GOOLBARG, 2012).

Na figura 1, temos um grafo com 4 arestas e 6 vértices. Um caminho em um grafo orientado, é qualquer sequência de arcos, onde o vértice final de um arco é o vértice inicial do próximo (BANEGAS CHAVEZ, 1985). Em nossa implementação todos os valores em destaque na figura 2, representam as 4! = 24 rotas possíveis para a figura 1, pois todas as arestas serão calculadas como origem.

O algoritmo de permutação é apresentado na fiigura 2. O algoritmo calcula recursivamente as permutações, ou trocas, de subconjuntos cada vez maiores do conjunto (por exemplo, um conjunto de dois, três elementos). Há um custo incorrido quando a permutação é produzida em ordem lexicográfica. Depois de cada passo recursivo, o subconjunto é rodado de volta para a ordem original na etapa precedente (BEERCAVE). A cada combinação o grafo resultante é vetorizado, adicionando uma aresta por vez, e verificando ao final o melhor e o pior trajeto.



Permutação em ordem lexicográfica, ou ordenada, recebendo como parâmetro 4 arestas (BEERCAVE).

Para a paralelização do algoritmo PIR sequencial finalizado, duas tecnologias foram utilizadas, sendo elas pThreads e openMP. Onde cada origem foi definida para iniciar independentemente do restante do grafo.

Os fenômenos naturais são inerentemente paralelos. Escrever programas sequenciais, via de regra, implica impor uma ordem à ações que são potencialmente independentes e que poderiam executar concorrentemente (YAMIN, 2006). Como na programação sequencial é inevitável arbitral uma particular ordem na qual as ações serão colocadas. Os pontos de origem de cada vetor são exemplos onde a ordem da execução praticamente não importa para o entendimento do problema real (YAMIN, 2006).

Os resultados estão assim organizados. Primeiramente são apresentados os resultados da execução. Nela pode ser visto a comparação entre o algoritmo sequencial, pThread e openMP. Na sequência são apresentados os resultados quanto ao custo computacional da criação de threads utilizando os métodos de paralelização.

Na Tabela 1 são apresentados os resultados dos tempos de execução dos algoritmos sequencial, pThread e openMP. Ambos os métodos paralelizados obtiveram melhor resultado que o método sequencial, Nos testes utilizando de 4 até 9 arestas, o openMP obteve melhor desempenho, já com o aumento no número de arestas o pThread foi quem o teve melhor resultado.

Tabela 1 - Resultado dos testes. Fonte: Próprio autor.

Número de Arestas	Sequencial (ms)	pThread (ms)	OpenMp (ms)
4	0,20	0,24	0,20
5	0,20	0,30	0,23
6	0,20	0,58	0,31
7	0,25	0,58	0,31
8	2,15	2,88	1,50
9	16,68	11,02	10,92
10	176,47	89,94	90,89
11	1820,71	959,86	1032,12
12	23154,96	11187,37	11243,30
13	283982,16	147652,18	158699,45
14	4157070,57	2175631,93	2248972,29

Para os testes da tabela acima, tanto o algoritmo com a tecnologia openMP quanto pThread foram criadas 2 (duas) threads. Na figuras 3 e 4, são apresentadas o resultado da criação de mais threads, mostrando-se ineficiente para este problema.

Na figura 3 é exposto o cálculo de Speed-up alcançado com a utilização de diferentes quantidades de threads. Onde o Speed-up representa quantas vezes o programa paralelo ficou mais rápido que a versão sequencial. É calculado pela razão entre o melhor tempo sequencial e o tempo da versão paralela (DE ROSE, 2016).

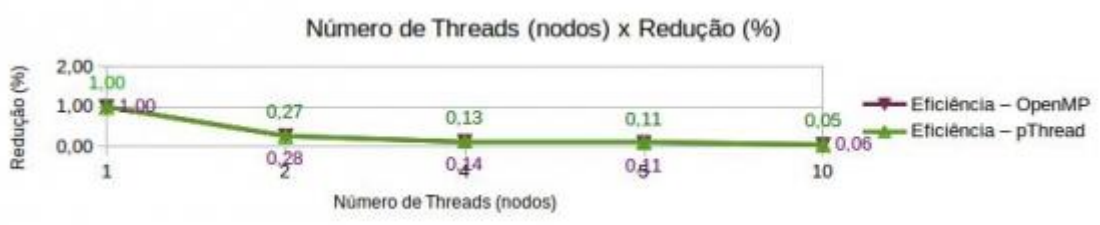


Figura 3: Speed-up na adoção da paralelização com pThread e openMP. Fonte: Próprio autor.

Na figura 4 é mostrado os resultados quanto a eficiência. A eficiência indica como foi a taxa de utilização média das unidades ativas utilizadas e se os recursos foram bem aproveitados. É calculado pela razão entre o Speed-up e o número de nodos utilizados (DE ROSE, 2016).

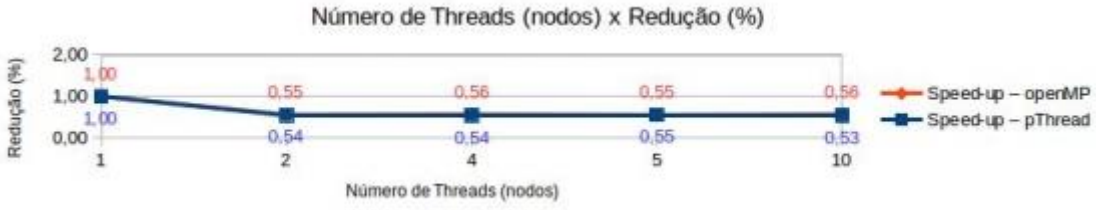


Figura 4 - Eficiência na adoção da paralelização com pThread e openMP. Fonte: Próprio autor.

Conclusões e Trabalhos Futuros

Com a paralelização do algoritmo de Inspeção de Rotas conseguiu-se ganho de desempenho de até 2 vezes sobre o sequencial, seja utilizando pThread ou openMP. A redução no tempo de execução foi semelhante nas duas implementações paralelas. Na comparação entre as duas APIs, pThread apresentou o melhor desempenho utilizando mais de 2 threads. Sendo que nos testes com até 9 arestas openMP obteve melhor resultado, já com a utilização de mais de 9 arestas foi pThread que obteve melhor desempenho.

Como trabalho futuro pretende-se paralelizar o algoritmo para a realização de testes utilizando placas de video da NVIDIA e tecnologia CUDA. Outro trabalho futuro poderia ser alterar o tipo de permutação escolhido, verificando o custo computacional da permutação lexicográfica, que é alto, onde possivelmente melhoraria o desempenho.

Palavras-chave: Problema de inspeção de rotas, Problema do carteiro chinês, Paralelização de algoritmos, Programação em linguagem C, Problema do caixeiro viajante.

Agradecimentos

Agradecemos ao Professor e tutor Msc. Edson Padoin pelos conhecimentos transmitidos durante a disciplina e realização deste trabalho.

Referências bibliográficas

BANEGAS CHAVEZ, Jorge Raul et al. Uma metodologia para o problema do carteiro chinês em redes mistas. 1985.

BEERCAVE. An ordered lexicographic permutation algorithm based on one published in practical algorithms in c++. Available at http://www.bearcave.com/random_hacks/permute.html. Accessed 05 Jun. de 2015.

DE ROSE, César AF. Fundamentos de Processamento de Alto Desempenho. Ijuí: ERAD- Escola Regional de Alto Desempenho, 2006.

EDMONDS, Jack; JOHNSON, Ellis L. Matching, Euler tours and the Chinese postman. Mathematical programming, v. 5, n. 1, p. 88-124, 1973.

GOLDBARG, Marco. Grafos: Conceitos, algoritmos e aplicações. Elsevier Brasil, 2012.

YAMIN, Adenauer Corrêa. Reflexões sobre aspectos que promovem a adoção do paralelismo. Ijuí: ERAD- Escola Regional de Alto Desempenho, 2006.