UNIVERSIDADE FEDERAL DO PARANÁ
CIÊNCIA DA COMPUTAÇÃO

ANDRÉ FELIPE DE ALMEIDA PONTES GRR20196474

TRABALHO 1
PROGRAMAÇÃO PARALELA

CURITIBA
2023

Resumo

Este trabalho tem como objetivo a implementação e análise do algoritmo do caixeiro viajante de forma paralela, fazendo uso da biblioteca OpenMP.

PROBLEMA SEQUENCIAL

O Algoritmo do caixeiro viajante é um algoritmo que visa encontrar a menor distância entre n cidades, visitando apenas uma vez cada cidade.

A função tsp é onde o caixeiro viajante é resolvido em sua maior parte, essa função implementa um algoritmo recursivo para encontrar a menor rota passando por cada cidade apenas uma vez. Se foi encontrado um caminho maior ou igual ao atual faz o backtracking. Caso a profundidade na árvore seja igual ao número de cidades, o comprimento atual recebe a distância da origem até a cidade onde estamos, se essa distância for menor que a menor distância atual, encontramos uma rota melhor. Caso contrário iteramos entre cada um dos vizinhos checamos com a função present se a cidade vizinha já está presente no caminho, se não estiver colocamos ela no caminho e entramos recursivamente na função. Obs: poderia ser feita uma otimização na função present, pois ela tem complexidade linear, transformando em O(1) se fosse implementado um HashMap.

```c
void tsp (int depth, int current_length, int *path) {
    int i;
    if (current_length >= min_distance) return;
    if (depth == nb_towns) {
        current_length += dist_to_origin[path[nb_towns - 1]];
        if (current_length < min_distance)
            min_distance = current_length;
    } else {
        int town, me, dist;
        me = path[depth - 1];
        for (i = 0; i < nb_towns; i++) {
            town = d_matrix[me][i].to_town;
            if (!present (town, depth, path)) {
                path[depth] = town;
                dist = d_matrix[me][i].dist;
                tsp (depth + 1, current_length + dist, path);
            }
        }
    }
}
```

ESTRATÉGIA DE PARALELIZAÇÃO

A estratégia de paralelização adotada consistiu na seleção de um vértice inicial, a partir do qual cada vizinho é percorrido, dando origem a uma task. Em um grafo completo, esse procedimento resulta em um total de n-1 tasks, onde *n* representa o número de vértices no grafo. Cada tarefa é executada em paralelo, visando otimizar o desempenho computacional.

CONCORRÊNCIA

Dado que cada task está envolvida no cálculo da menor distância, é possível encontrar potenciais problemas de concorrência no cenário em que duas tarefas identificam uma distância mínima simultaneamente. Como medida de mitigação, foi implementado um lock para garantir que apenas uma task possa efetivamente registrar a menor distância identificada.

METODOLOGIA DE TESTES

Os testes foram executados de forma automática, por um script feito em bash, todos os testes foram executados durante a noite com o mínimo de processos atrapalhando a execução.

AMBIENTE DE TESTES

Sistema Operacional: 20.04 LTS (Focal Fossa)
Kernel: 5.10.16.3
Compilador: gcc 9.3.0
Flags de Compilação: -O3 -fopenmp -lm
Processador: AMD Ryzen 5600x 3.6Ghz

LEI DE AMDAHL

Utilizando uma entrada de 18 cidades a porcentagem média de execução sequencial, obtida através de 20 execuções foi 0,0009%

Tempo Sequencial(%)	2 CPUs	4 CPUs	8 CPUs	Infinitos CPUs
0,0009	1,998	3,989	7,949	1111,1111

TESTES

TABELA DE TEMPO DE EXECUÇÃO

	1 CPU		2 CPUs		4 CPUs		8 CPUs	
N	Tempo Médio(s)	Desvio Padrão	Tempo Médio(s)	Desvio Padrão	Tempo Médio(s)	Desvio Padrão	Tempo Médio(s)	Desvio Padrão
15	14,1	0,05	9,47	1,37	9,26	0,83	9,63	0,6
16	81,58	8,10	52,82	4,74	32,24	4,54	32,10	2,10
17	86,06	2,56	51,54	7,14	38,63	3,15	40,69	3,49
18	171,12	25,76	107,43	9,42	112,40	27,15	120,53	17,04

TABELA DE SPEEDUP

N	1 CPU	2 CPUs	4 CPUs	8 CPUs
15	1,0	1,48	1,52	1,46
16	1,0	1,54	2,53	2,54
17	1,0	1,66	2,22	2,11
18	1,0	1,59	1,52	1,41

TABELA DE EFICIÊNCIA

N	1 CPU	2 CPUs	4 CPUs	8 CPUs
15	1,0	0,74	0,38	0,18
16	1,0	0,77	0,63	0,31
17	1,0	0,83	0,55	0,26
18	1,0	0,81	0,38	0,17

ANÁLISE

Os resultados não foram muito bons, pois ao aumentar o número de de CPUs temos uma grande variação tanto dos speedups quanto da eficiência, como cada execução do caixeiro viajante é escolhido um caminho aleatório a aleatoridade faz com que seja difícil até mesmo prever duas execuções de mesmo N e mesma quantidade de CPUs, se uma execução encontra um caminho muito próximo do correto a árvore é podada rapidamente caso contrário a árvore cresce muito e o tempo de execução consecutivamente.

ESCALABILIDADE

De acordo com a tabela de eficiência, é possível concluir que o algoritmo apresenta escalabilidade fraca, pois a eficiência ao aumentar o tamanho da entrada diminui de forma acentuada e ocorrem muitas variações.

CONCLUSÃO

Os resultados obtidos são consistentes, pois estão em conformidade com as características inerentes ao problema do caixeiro viajante. Dado que o algoritmo muitas vezes segue caminhos distintos a cada execução, essa variação é esperada, podemos ver pelos desvios padrões.No entanto, é importante ressaltar que a complexidade fatorial do algoritmo representa um desafio, levando a um crescimento exponencial do tempo de execução à medida que o problema aumenta de escala.