IFNMG — CAMPUS MONTES CLAROS
CIÊNCIA DA COMPUTAÇÃO

Gabriel Davi Mota Baia
Giordani André Versiani

Arquitetura II

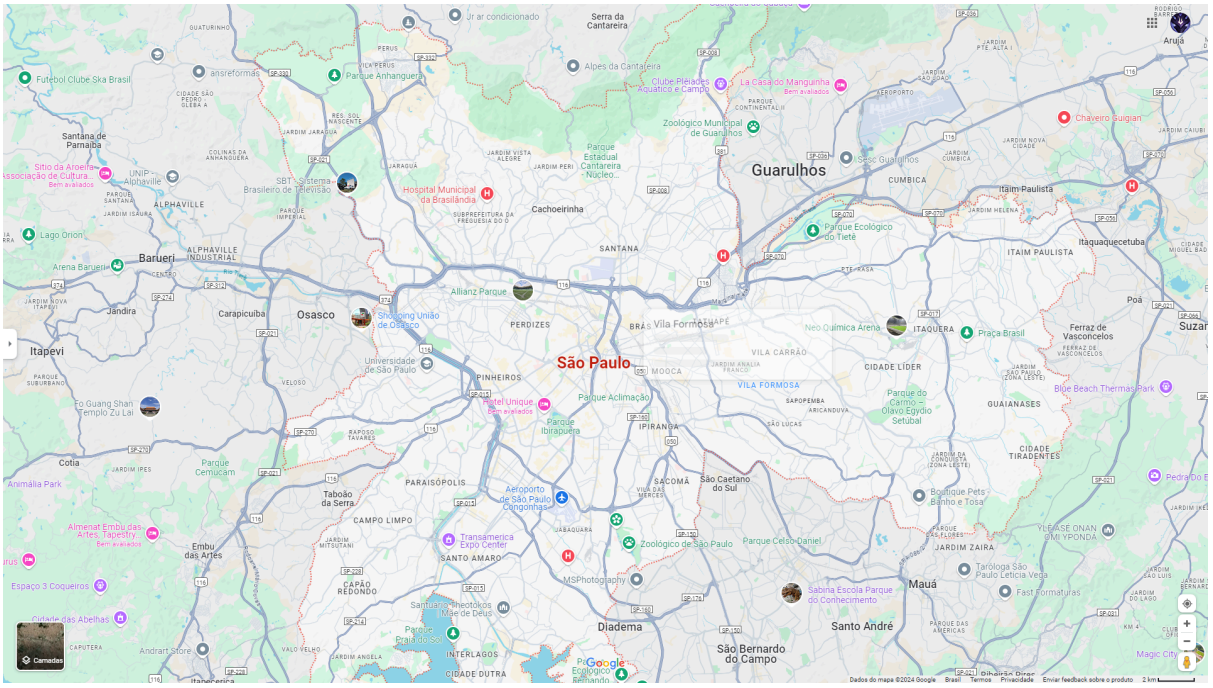
Floyd Warshall

Montes Claros - Minas Gerais

1. Introdução

Este relatório apresenta a implementação de um sistema para calcular distâncias em um mapa urbano, focando na localização de hospitais, pontos de lazer entre outros, além de analisar a distância entre esses locais. O sistema utiliza técnicas de cálculo de distâncias geográficas e implementa o algoritmo de Floyd-Warshall para determinar as menores distâncias entre pontos no mapa.

A aplicação é dividida em módulos que lidam com diferentes aspectos do mapa, como inicialização de distâncias, cálculo de rotas mais curtas, e visualização dos dados em uma matriz que representa o mapa da cidade de São Paulo.



Este recorte do mapa possui de **X: 67,000 x Y: 37,000** Metros

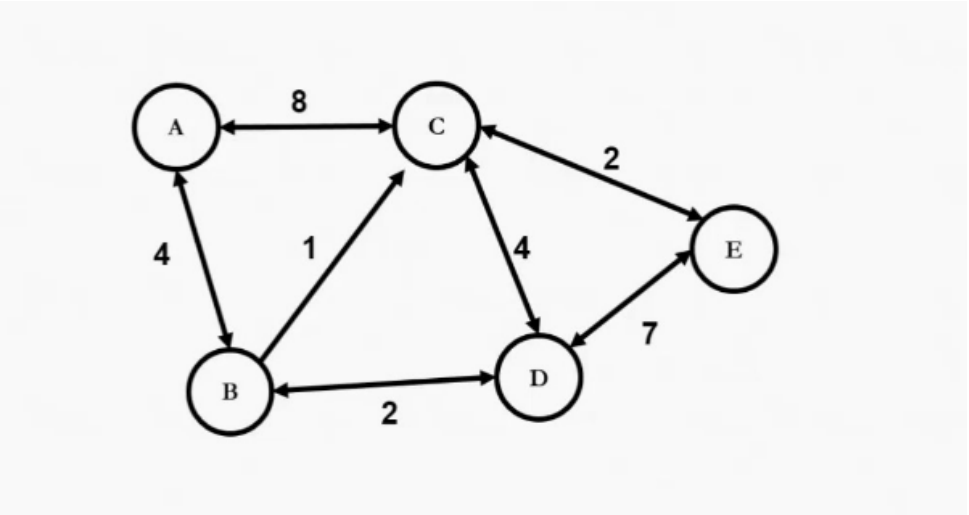
2. Método Floyd Warshall

O método de Floyd-Warshall é um algoritmo utilizado para encontrar as menores distâncias entre todos os pares de vértices por meio de um grafo ponderado. Esse grafo pode conter arestas com pesos positivos ou zero, desde que não haja ciclos de peso negativo (um ciclo onde a soma dos pesos das arestas é negativa, o que resultaria em uma solução indefinida).

- Funcionamento do Algoritmo

O algoritmo de Floyd-Warshall usa a programação dinâmica para calcular as menores distâncias. Ele parte de uma matriz de distâncias, que inicialmente contém as distâncias diretas entre os vértices (ou infinito, se não há uma aresta direta entre eles). Ao longo do algoritmo, essa matriz é atualizada iterativamente, considerando

cada vértice intermediário possível.



- **Inicialização**

O algoritmo começa criando uma matriz de distâncias, que chamaremos de D, onde:

- Cada célula D[i][j] representa a distância do vértice i para o vértice j.
- Se existe uma aresta direta entre i e j, então D[i][j] será o peso dessa aresta.
- Se **não** existe aresta direta, D[i][j] é inicializado com ∞ (infinito).
- A diagonal da matriz (D[i][i]) é sempre 0, pois a distância de um vértice para si mesmo é zero.

Exemplo de matriz inicial para um grafo com 4 vértices:

0	**3**	∞	**7**
8	**0**	**2**	∞
5	∞	**0**	**1**
2	∞	∞	**0**

- **Iteração**

O algoritmo itera sobre cada vértice k, considerando-o como um vértice intermediário. A cada iteração, ele verifica se o caminho entre dois vértices i e j passando por k (i → k → j) é mais curto do que o caminho direto entre i e j.

Para cada par de vértices i e j, a atualização da matriz de distâncias D é feita da seguinte forma:

$$D[i][j] = \min(D[i][j], D[i][k] + D[k][j])$$

Isso significa que a nova distância entre i e j será o menor valor entre a distância já registrada (D[i][j]) e a distância passando por k (D[i][k] + D[k][j]).

- **Repetição**

Este processo é repetido para todos os pares de vértices i e j, para cada vértice intermediário k. Ao final, a matriz D conterá as menores distâncias entre todos os pares de vértices.

Exemplo Completo

Imagine um grafo com 4 vértices e a matriz inicial de distâncias, como mostrado anteriormente:

$$\begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

Após executar o algoritmo de Floyd-Warshall, a matriz de distâncias mínimas entre todos os pares de vértices ficaria assim:

$$\begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

Isso significa que, por exemplo:

- A menor distância entre o vértice 1 e o vértice 3 é **5**.
- A menor distância entre o vértice 2 e o vértice 4 é **3**.

O algoritmo de Floyd-Warshall tem uma complexidade de tempo de **O(n³)**, pois ele executa três laços aninhados que percorrem os vértices do grafo. Cada laço percorre n vértices, resultando no tempo cúbico.

Aplicações:

O algoritmo de Floyd-Warshall tem diversas aplicações, incluindo:

- **Roteamento de Redes**: Encontrar rotas mais curtas em redes de computadores.
- **Mapas Rodoviários**: Determinar as rotas mais eficientes entre diferentes locais.

O algoritmo de Floyd-Warshall é uma solução poderosa para problemas de caminho mínimo quando se precisa das menores distâncias entre **todos** os pares de vértices em um grafo. Embora seja menos eficiente que outros algoritmos como o de Dijkstra para grafos grandes, ele é uma ferramenta valiosa quando aplicado em situações onde todas as combinações de vértices são relevantes.

Distância Euclidiana

A Distância Euclidiana é uma métrica para calcular a distância direta entre dois pontos no espaço. É amplamente usada em geometria e em problemas que envolvem a medição de distâncias em um plano.

Para dois pontos com coordenadas (x_1, y_1) e (x_2, y_2) , a Distância Euclidiana é calculada pela fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esta fórmula deriva do Teorema de Pitágoras e mede a distância direta entre os pontos, ignorando quaisquer obstáculos ou caminhos intermediários.

O código foi implementado em C, utilizando OpenMP para paralelizar as operações que envolvem a atualização das distâncias entre os pontos do grafo. O programa foi estruturado de forma a permitir a escolha de diferentes funcionalidades através de um menu interativo. As funcionalidades incluem o cálculo de distâncias entre pontos específicos, visualização das distâncias mínimas entre todos os pontos e a aplicação do algoritmo de Floyd-Warshall em diferentes escalas de mapas.

3. Aplicação

Para a elaboração do mapa e seu mapeamento, fizemos diversas divisões em grades, em diferentes estilos e cidades. Inicialmente, começamos com o mapa de Montes Claros - MG, em uma escala de teste de 5x5. Após isso, realizamos a análise em uma escala maior, de 25x25, e medimos e analisamos os dados recebidos, comparando-os com os dados reais. Até este ponto, o algoritmo estava gerando resultados satisfatórios. No entanto, como queríamos trabalhar com uma matriz maior, de no mínimo 100x100, precisávamos de uma cidade maior, com mais pontos de entretenimento e locais a serem mapeados, para obter uma matriz mais precisa.

Assim, passamos a utilizar a cidade de São Paulo para um mapeamento mais rigoroso no sistema, utilizando uma matriz maior e mais pontos a serem calculados. Primeiramente, começamos a criar a grade da cidade de São Paulo e a fazer o mapeamento dos pontos em sua grade 100x100. Os pontos principais a serem considerados foram os hospitais, com um total de 25, e os estádios de futebol, com um total de 3, que seriam considerados pontos de lazer.

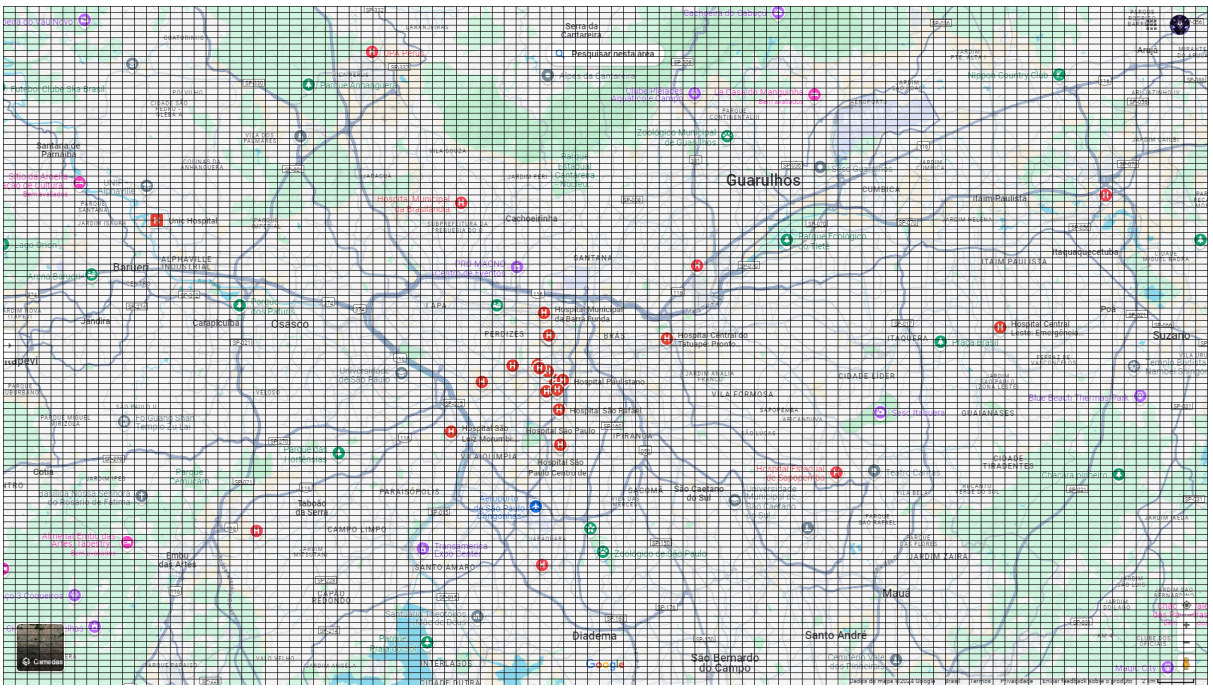
Com isso pronto, os pontos foram mapeados na seguinte grade em um mapa de 67km por 37km:

Segue a imagem abaixo do mapeamento de São Paulo com grade

67x37km

26 hospitais

3 estádios



Link do mapa para melhor visualização: https://prnt.sc/0lMBoJ_P8JAX

Com o mapa e a grade prontos, transformamos posteriormente a grade em uma matriz. Utilizamos um algoritmo para converter essa matriz em um grafo, permitindo o cálculo das distâncias entre os pontos. Em seguida, fizemos a estimativa e os cálculos das distâncias entre os pontos de interesse. Para isso, empregamos o algoritmo de para calcular as menores distâncias entre todos os pontos mapeados, considerando as seguintes informações:

```
6   #define DIST_VE 633.00 // Distância vertical em metros
7   #define DIST_HO 733.00 // Distância horizontal em metros
```

Até este momento, possuímos 28 pontos no nosso mapa, com a adição de mais pontos que serão informados posteriormente.

Hospitais: 26 pontos distribuídos pela cidade.
Estádios de Futebol (Pontos de Lazer): 3 pontos de interesse (Allianz Parque - Palmeiras, Neo Química - Corinthians, Pacaembu - Prefeitura Sp).

Primeiro teste do algoritmo comparando a distância real com a distância encontrada pelo algoritmo nos estádios de futebol:

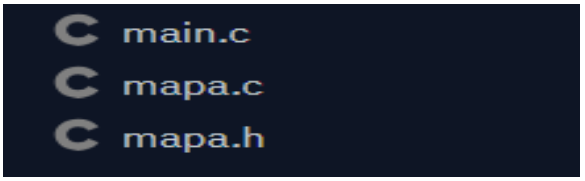
Termo medido	Distância real	Distância Código	Taxa erro
Allianz → Neo Química	20,780 metros	21.093,77 metros	1,51%
Allianz → Pacaembu	2,500 metros	2.635,97 metros	5,44%
Pacaembu → Neo Química	19,090 metros	18.488,24 metros	3,15%

É importante observar que, ao tratar de estádios, que possuem um tamanho grande, a média de erro de 350,50 metros pode variar dependendo de onde traçamos a linha no Google Maps. Ao traçar a linha em pontos diferentes dentro do estádio, a diferença na medida pode chegar a até 250 metros. Isso destaca que, neste primeiro teste do algoritmo, a maior taxa de erro foi observada em medidas menores, onde a escolha do ponto pode ter um impacto mais significativo no resultado final.

Vale salientar que a distância real foi medida utilizando a ferramenta "Medir" do Google Maps como mostrado abaixo

O primeiro teste revelou que o algoritmo de Floyd-Warshall fornece uma estimativa precisa das distâncias no mapa, com diferenças mínimas em relação às distâncias reais. Essas discrepâncias foram analisadas e atribuídas a fatores como a resolução da matriz e a precisão das coordenadas de entrada.

Código



Estrutura e Funcionalidade do Código

O código é dividido em três partes principais: a implementação das funções no arquivo principal, as funções auxiliares, e o cabeçalho que contém definições e declarações. A seguir, descrevemos cada parte do código e suas funcionalidades.

1. Implementação das Funções Principais

- **Menu de Opções:** O código começa com a definição de uma função que exibe um menu interativo para o usuário. Este menu apresenta opções para calcular distâncias entre pontos e encerrar o programa. O menu é uma interface simples que orienta o usuário nas diferentes funcionalidades do sistema.
- **Função Principal (main):** Esta função é o ponto de entrada do programa. Nela, é configurado o número de threads a ser utilizado pelo OpenMP, permitindo a execução paralela das operações. Em seguida, duas estruturas principais são alocadas: um mapa que representa o espaço físico (como um grid) e uma matriz de distâncias para armazenar as distâncias entre hospitais e pontos de lazer. O mapa é inicializado com strings vazias, e a matriz de distâncias é preenchida com valores iniciais que representam a distância máxima. Assim que o programa inicia, ele calcula as distâncias de todos os pontos vazios em relação aos hospitais e pontos de lazer. O programa então entra em um loop onde exibe o menu, lê a escolha do usuário e executa a opção correspondente. O loop continua até que o usuário escolha a opção de encerrar o programa.

2. Funções Auxiliares

- **Alocação de Memória:** Há várias funções para alocar e liberar memória. Isso inclui a alocação de matrizes para armazenar distâncias e o mapa, além de funções para liberar essa memória ao final da execução, evitando vazamentos.
- **Cálculo de Distâncias:** O código possui funções dedicadas ao cálculo da distância entre pontos, tanto em relação a hospitais quanto a pontos de lazer. Ele utiliza a fórmula da distância euclidiana para calcular a distância entre coordenadas (x, y).
- **Inicialização das Distâncias: U**ma função específica é utilizada para inicializar a matriz de distâncias, configurando distâncias máximas entre os pontos e zero para distâncias de um ponto a ele mesmo. Essa função também calcula as distâncias entre todos os hospitais e entre pontos de lazer, além de distâncias entre hospitais e pontos de lazer.
- **Cálculos Paralelos:** O código utiliza OpenMP para calcular distâncias entre os pontos vazios de forma paralela, aumentando a eficiência do processamento. As operações de cálculo são realizadas em um loop paralelo, onde cada thread pode calcular distâncias independentes.

- **Algoritmo de Floyd-Warshall:** O código implementa o algoritmo de Floyd-Warshall, que é um algoritmo clássico para encontrar as menores distâncias entre todos os pares de vértices em um grafo. Neste caso, ele é aplicado à matriz de distâncias previamente calculadas para otimizar e encontrar as menores distâncias entre todos os pontos, incluindo hospitais e pontos de lazer.

3. Cabeçalho (mapa.h)

O cabeçalho do código contém definições de constantes e declarações das estruturas de dados e funções utilizadas. Isso inclui:

- **Definições de Tamanho: T**amanhos constantes são definidos para as dimensões do mapa, distâncias verticais e horizontais, e o número máximo de hospitais e pontos de lazer.
- **Estrutura para Pontos de Lazer:** Uma estrutura que armazena as coordenadas e nomes dos pontos de lazer, facilitando o acesso a essas informações durante os cálculos.
- **Declarações de Funções:** Todas as funções auxiliares são declaradas no cabeçalho para que possam ser utilizadas em diferentes partes do código, promovendo uma melhor organização e modularidade.

Conclusão do Codigo

O código implementa um sistema eficiente para calcular e manipular distâncias em um mapa, utilizando técnicas de programação estruturada e paralelismo. Isso possibilita uma resposta rápida e precisa ao usuário, tornando o sistema útil para diversas aplicações, como planejamento urbano e análise de acessibilidade. O código é modular, com cada parte responsável por uma funcionalidade específica. A utilização do algoritmo de Floyd-Warshall permite calcular as menores distâncias entre pontos de interesse em um mapa urbano, proporcionando uma ferramenta eficaz para análise de rotas e distâncias. A estrutura modular melhora a organização, manutenção e reusabilidade do código.

4.1 Resultados encontrados

As principais métricas avaliadas foram o tempo de execução dos algoritmos e a precisão dos cálculos. O tempo de execução foi medido para diferentes tamanhos de mapas e números de threads, enquanto a precisão foi verificada comparando os resultados obtidos com valores verificados no google maps.

id/ metros			
Allianz → Neo Química |	20.780	| 21.093.77	1.51%
Allianz → Pacaembu |	2.500	| 2.635.97	5.44%
Pacaembu → Neo Química|	19.090	| 18.488.24	3.15%

Ao medir distâncias em estádios ou áreas de grande tamanho, a média de erro pode variar consideravelmente. No caso dos estádios, a diferença na medida pode chegar a até 250 metros dependendo de onde a linha é traçada no Google Maps. Isso destaca que, neste primeiro teste do algoritmo, a maior taxa de erro foi observada em medidas menores, onde a escolha do ponto pode ter um impacto mais significativo no resultado final.

Implementação OpenMP e Paralelismo

A paralelização distribui o trabalho de um programa entre múltiplos núcleos de processamento, permitindo que várias operações sejam executadas simultaneamente. Isso reduz o tempo de execução de tarefas que, de outra forma, seriam realizadas de forma sequencial. No caso do código, a paralelização com OpenMP acelera o cálculo das distâncias ao dividir o processamento entre várias threads, cada uma executando partes independentes do cálculo ao mesmo tempo, melhorando a eficiência em sistemas multicore.

Objetivo da Paralelização

O objetivo da paralelização no código foi otimizar o tempo de execução dos cálculos de distâncias entre um ponto de interesse e uma lista de destinos (hospitais, pontos de lazer, etc) em um ambiente de grid. Dada a potencial grande quantidade de destinos e o custo computacional de calcular distâncias para cada um deles, a paralelização foi utilizada para distribuir essas tarefas entre múltiplos núcleos de processamento, acelerando o tempo total do cálculo.

Escolha das Partes a Paralelizar

Vantagens da Aplicação neste Código

1. **Desempenho Melhorado**:
    - A paralelização permite que múltiplos cálculos sejam realizados simultaneamente, resultando em um tempo de execução significativamente reduzido, especialmente em grandes mapas ou quando há muitos hospitais e pontos de lazer.
2. **Escalabilidade**:
    - O uso de múltiplas threads permite que o código escale bem com o aumento do número de pontos a serem processados. Isso é crucial para aplicações em tempo real ou que lidam com grandes quantidades de dados.
3. **Eficiência de Recursos**:
    - A utilização de recursos computacionais disponíveis, como múltiplos núcleos de CPU, melhora a eficiência do código e proporciona uma melhor utilização do hardware.
4. **Simplicidade na Implementação**:
    - OpenMP oferece uma forma simples de adicionar paralelismo ao código existente sem a necessidade de reestruturar significativamente a lógica do programa. Isso facilita a manutenção e a legibilidade do códig

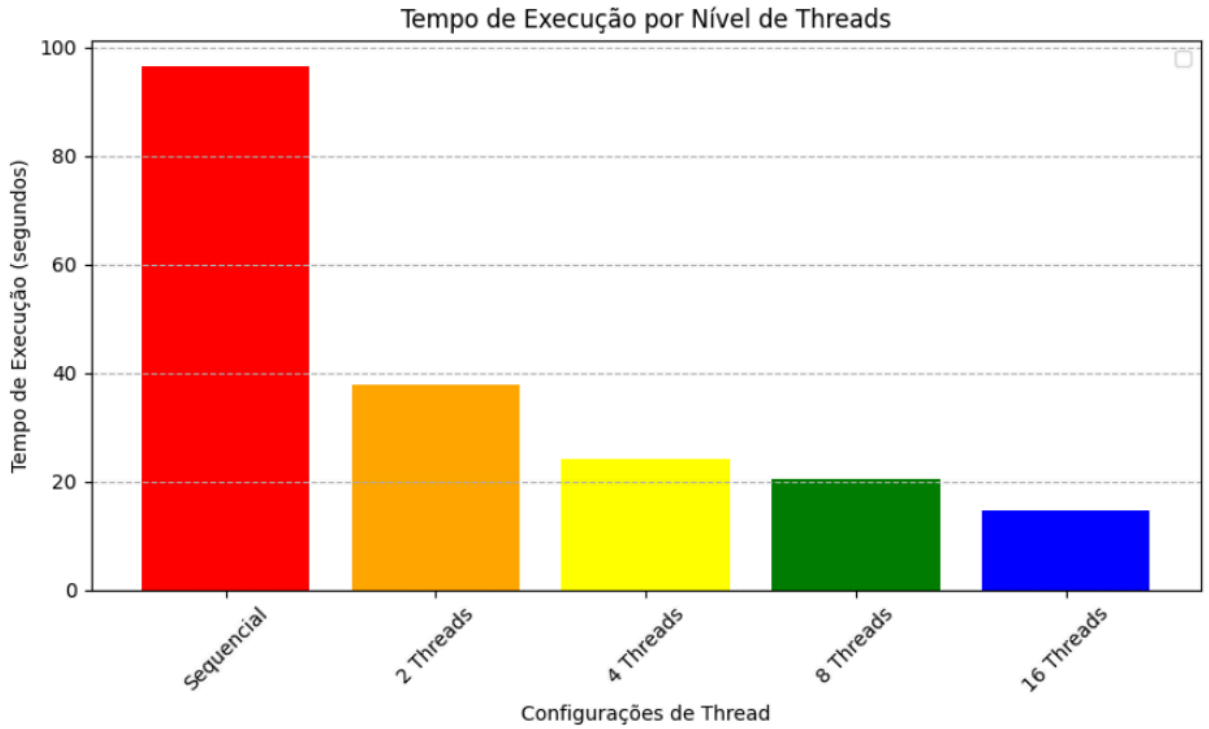
Conclusão

A paralelização do cálculo de distâncias usando OpenMP foi uma escolha natural devido à independência dos cálculos de cada hospital ou ponto de lazer. Ao distribuir essas operações entre múltiplos núcleos, foi possível reduzir significativamente o tempo de execução do programa, especialmente em cenários com um grande número de destinos. Com a devida atenção à sincronização de dados compartilhados, a implementação paralela mantém a correção e a eficiência do programa original.

5.1 Comparativos OpenMP/Paralelismo

Com o desenvolvimento do código e o aumento escalonado das distâncias, **a matriz passou a ter dimensões de 6500 x 6500** muito superior aos 100x100 anteriores, mantendo a precisão dos pontos e distâncias. Realizamos, então, um comparativo, executando dois códigos: um com a paralelização implementada e outro sem essa implementação. Os resultados mostram uma melhora bastante perceptível, com diferenças significativas nos tempos de execução. #define NUM_THREADS N

ID	Tempo exec	Conclusão
Não Paralelo	96,5 segundos	LENTO!
Paralelo 2 Threads	37,87 segundos	redução de cerca de 60,7%
Paralelo 4 Threads	24,16 segundos	redução de cerca de 75,4%
Paralelo 8 Threads	20,34 segundos	redução de cerca de 78,9%
Paralelo 16 Threads	14,77 segundos	redução de cerca de 84,7%



considere uma margem de erro de 3 segundos

- **Teste realizados no processador i5 12500H, 16gb ram dd5**

Conclusão final

A implementação do algoritmo de Floyd-Warshall e Distância Euclidiana, combinada com técnicas de paralelização utilizando OpenMP, resultou em um sistema robusto e eficiente para o cálculo de distâncias em um mapa urbano. O projeto não apenas demonstrou a capacidade do algoritmo em determinar as menores distâncias entre múltiplos pontos de interesse, mas também evidenciou a importância da paralelização para otimizar o desempenho em grandes conjuntos de dados.

Os testes realizados com a cidade de São Paulo, em uma matriz de 100x100 pontos e posteriormente 6500 x 6500, mostraram que o algoritmo fornece estimativas precisas das distâncias, com taxas de erro aceitáveis, especialmente considerando a complexidade do mapeamento urbano. A comparação entre as distâncias reais e as calculadas pelo algoritmo revelou resultados satisfatórios, com discrepâncias mínimas que podem ser atribuídas a fatores como a resolução do mapa e a precisão das coordenadas.

Além disso, a introdução do OpenMP melhorou significativamente o tempo de execução do sistema, permitindo que múltiplos núcleos de processamento fossem utilizados de forma eficaz. Os resultados mostraram uma redução grande no tempo de processamento ao aplicar paralelização, demonstrando claramente os benefícios de otimizar algoritmos para ambientes de computação paralela.

Em suma, este trabalho não só contribui para a análise de distâncias e rotas em contextos urbanos, mas também serve como um exemplo de como técnicas de programação e algoritmos bem escolhidos podem transformar desafios computacionais em soluções eficientes e escaláveis.

Referências:

- https://brilliant.org/wiki/floyd-warshall-algorithm/
- https://www.meuguru.com/blog/distancia-3-mais-usadas/
- https://proceedings.science/sbpo/sbpo-2020/trabalhos/algoritmo-de-floyd-warshall-utilizado-no-mapeamento-das-rotas-de-farmacias?lang=pt-br
- https://conged.deinfo.uepg.br/artigo1.pdf
- https://www.researchgate.net/publication/267763973_Paralelizacao_do_algoritmo_Floyd-Warshall_usando_GPU