Insper, 2024-1
Supercomputação

Projeto final
Jerônimo de Abreu Afrange

`https://github.com/jeronimo-a/projeto-supercomp-2024-1`

Solução 1: método Monte Carlo

Na solução 1, ao invés de fazer uma busca exaustiva, foi utilizado o método de Monte Carlo, para simplificar o processo de implementação. O emprego do método foi feito variando de forma aleatória a ordem dos candidatos fornecidos para a função definida pelo pseudocódigo fornecido, e selecionando, em seguida o resultado mais adequado de todos os obtidos.

Para implementar uma busca exaustiva, seria necessário testar todos os possíveis arranjos do vetor de candidatos, o que teria complexidade computacional altíssima, visto que o número de arranjos possíveis é n fatorial, sendo n a quantidade de vértices da rede.

Até qual tamanho de problema você conseguiu obter um resultado em tempo hábil (aprox. 15 min)?

Considerando que a implementação não foi uma implementação exaustiva, foi possível obter resultados para redes grandes em tempo hábil. Com 10000 iterações foi possível analisar uma rede de 350 vértices em menos de 15 minutos.

No entanto, se a busca fosse exaustiva, seriam necessárias quase 90 bilhões de iterações para uma rede de míseros 14 vértices (14 fatorial). Mesmo se uma iteração demorasse 1µs, o que é muito pouco, a execução completa demoraria mais de um dia.

Você deve apresentar um pseudocódigo de uma heurística para otimizar essa tarefa exaustiva.

Uma possível heurística seria ordenar de forma crescente os candidatos pelos seus graus (número de conexões), em seguida embaralhar os n primeiros candidatos do vetor. Quanto menor o valor de n, menos ordenações diferentes do vetor de candidatos teriam de ser analisadas, mas também se tornaria menor a probabilidade de encontrar a solução ótima (considerando um número grande de iterações).

Pseudocódigo

candidatos = vetor dos candidatos ordenados por identificador
ordenar candidatos de forma crescente a partir do seu grau (número de conexões)

clique máxima = vetor não inicializado
tamanho clique máxima = -1

rodar quanto mais vezes o possível:

embaralhar os n primeiros nós do vetor dos candidatos
clique = clique encontrada a partir da ordem atual dos candidatos

se o tamanho dessa clique encontrada for maior que o tamanho clique máxima definido acima
tamanho clique máxima = tamanho dessa clique atual
clique máxima = clique atual

retornar a clique máxima

É possível implementar alguma poda? Algum critério que evite calcular um nó, dado que você já descobriu uma clique maior?

É possível verificar qual o tamanho máximo da clique que será gerada pelo pseudocódigo fornecido no enunciado, é impossível que ele seja maior que o grau do último nó do vetor dos candidatos. Desta forma, antes de rodar a função do pseudocódigo, faz sentido verificar se o tamanho da maior clique encontrada é maior que o grau do último nó do vetor.

Solução 2: paralelização com a heurística gulosa

A heurítsica gulosa foi implementada calculando somente a clique local do vetor de nós candidatos ordenados pelo grau, sendo o nó de maior grau o último do vetor. O grau do primeiro candidato a ser testado pelo algoritmo fornecido é o tamanho máximo possível da clique a ser obtida, no entanto, isso não quer dizer que a clique obtida será a maior possível.

As mudanças realizadas no progama descrito acima a fim de paralelizar o processamento foram feitas em quatro pontos.

O primeiro ponto foi no laço for de inicialização do vetor de candidatos inicial, na função `findMaximumClique`. O *loop* simplesmente preenche um vetor com números ordenados que vão de 0 até a quantidade total de nodes menos 1, portanto, as mudanças que ocorrem no vetor são completamente

compartimentalizadas, bastando assim, dividir as iterações do *loop* entre a maior quantidade de *threads* possível, de forma indiscriminada.

O segundo ponto de paralelização foi na inicialização da matriz de grau por nó, dentro da função `sortNodesByDegree`. A paralelização, apesar de desta vez o vetor ser bidimensional, foi feita de forma similar à do ponto anterior. Trata-se de um laço duplo aninhado, no qual apenas uma linha da matriz é manipulada por iteração, facilitando o processo de paralelização. Não foi possível colapsar os dois laços em um só porque dentro do laço interno um mesmo elemento da matriz seria incrementado por mais de uma *thread*, causando possíveis colisões.

O terceiro ponto foi na cópia dos índices dos nós da matriz ordenada para um novo vetor. Sendo uma cópia simples, a paralelização foi feita da mesma maneira que no primeiro ponto, simplesmente dividindo as iterações do laço de forma indiscriminada entre o maior número de *threads* possível.

O quarto ponto foi o que fez maior diferença na performance, por paralelizar uma parte mais demorada do processo. O ponto em questão é o *loop* interno da função `findClique`, que encontra o clique local a partir da lista de nós ordenada pelo grau. O *loop* interno percorre todos os nós ainda na lista de candidatos da clique, verificando se este é adjacente a cada membro já pertencente à clique. No caso que o candidato é adjacente a todos, ele é incluso em uma nova lista. A ordem com a qual eles são inclusos nesta nova lista é importante, e deve ser igual a ordem das iterações do *loop*, por isso, neste caso, utilizou-se uma cláusula de *loop* `for` paralelizado ordenado do OpenMP, mantendo a ordem original das iterações. Manter a ordem original é importante para chegar ao mesmo resultado da versão sequencial.

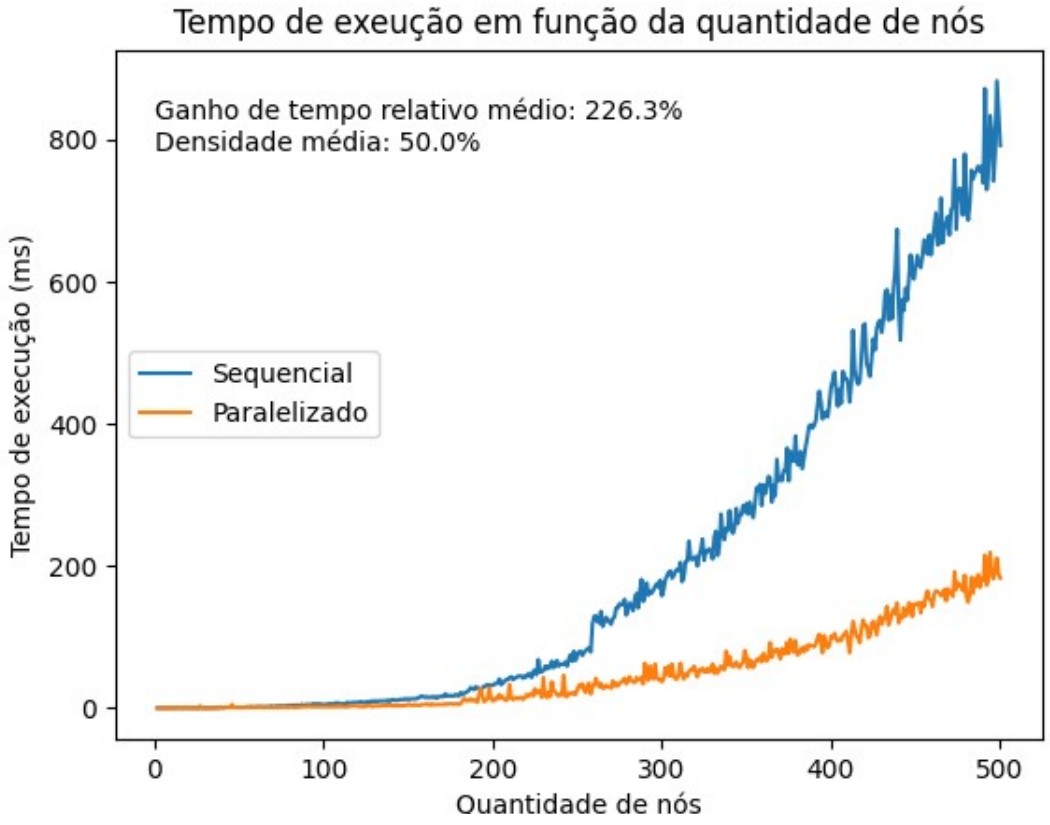
Solução 3: distribuição de processamento

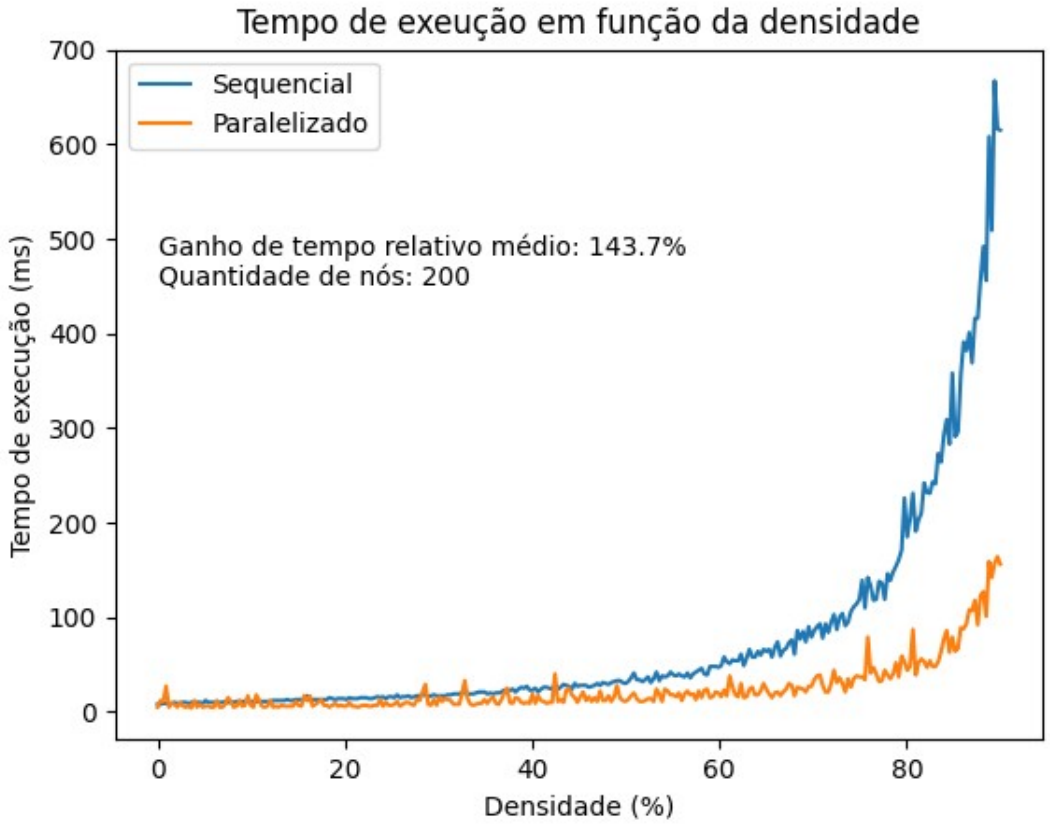
Esta etapa não foi realizada, mas no caso da solução exaustiva, ou de Monte Carlo, bastaria distribuir o processamento das variantes para obter um ganho na performance.

Comparação das performances

A solução feita a partir do método de Monte Carlo, quando executada sobre uma rede de 500 nós, 50% de densidade e com 100 iterações, demorou 66418ms. Com isso, calcula-se que cada iteração, para a tal rede, demora aproximadamente 6,6s. Além disso, sabe-se que o número de iterações necessárias para cobrir todos os arranjos possíveis dos nós é 500 fatorial, que, por si só, já é um número que tem centenas de dígitos. O tempo de execução da solução exaustiva para a rede em questão seria, portanto, 6,6 vezes 500 fatorial segundos, o que provavelmente é ordens de magnitude mais tempo do que a idade do universo em segundos ao quadrado.

Abaixo seguem gráficos do desempenho das implementações sequencial e paralelizada da heurística gulosa. A partir das imagens identifica-se que a performance destes é praticamente infinitamente melhor do que a busca exaustiva.





Aprofundamento Futuro

Com o objetivo de analisar melhor as soluções expostas, seria possível analisar as cliques obtidas a partir do método de Monte Carlo em comparação às obtidas pela heurística gulosa. A partir dessa comparação seria possível determinar se a heurística é melhor que a solução aleatória média ou não.

Extras

- `status.txt`, no repositório