Grupo: Gabriel Passos de Oliveira, Layna Dias, Letícia Dias, Letícia Yuri Hiratsuka.

Relatório paralelização do algoritmo de Dijkstra

Como o algoritmo escolhido necessita visitar por diversas vezes os nós e arestas de um grafo para definir todos os menores caminhos entre um nó fonte e todos os outros nós, o principal problema para uma execução rápida do código é a quantidade de acessos a esse nós e arestas. Assim, a melhor estratégia para sua paralelização seria usar paralelismo de dados, pois então as visitas aos nós, necessárias para calcular as menores distâncias até a fonte, podem ser divididas entre as threads.

Como o código possui diversos loops, usamos a diretiva “for”, esta é uma escolha viável uma vez que todos tinham a quantidade de iterações definidas, ou seja, não há chance de uma parada inesperada no loop devido a uma condição ou a um break.

Inicialmente, dividimos as iterações do primeiro loop, que realiza a inicialização dos valores do vetor de distâncias e do vetor de visitados. Em seguida, o segundo loop também utilizou da diretiva “for” para dividir os nós entre as threads que definem os novos valores do vetor de distâncias com os valores dos custos “w” das arestas correspondentes.

Continuamente, o próximo laço se diferencia por possuir dois laços internos, inicialmente foi testado paralelizar os laços internos, mas o primeiro deles apresentou ter maior dificuldade de paralelização, pois o “omp for” e “reduction” não seriam suficiente para recuperar o valor da variável “min”. Ademais, estes loops internos não seriam vantajosos o suficiente, pois apenas analisam as arestas pertencentes a um único nó. Dessa forma, decidimos testar a paralelização do loop externo, pois seria bem mais vantajosa, pois realiza mais trabalho, nesse caso, há uma iteração para cada nó do grafo.

Assim, usamos a diretiva “for” para o laço externo, porém, devido à concorrência de dados da variável “min”, não foi possível fazê-lo, já que o resultado nem sempre era correto, então, optamos por deixá-lo sequencial, mesmo que o ganho não fosse ser tão grande quanto possível.

Por fim, o segundo laço interno, poderia ser paralelizado facilmente com a cláusula “for”, sendo necessário declarar a variável “dest” privada. Neste loop, há uma região crítica na alteração dos valores de “distances”, portanto a cláusula “critical” foi utilizada para esse bloco

Máquina utilizada:

CPU(s): 8

On-line CPU(s) list: 0-7

Model name: AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx

CPU family: 23

Model: 17

Thread(s) per core: 2

Core(s) per socket: 4

Socket(s): 1

Placa de vídeo: GTX 3050

Memória RAM: 8GB

Memória: 256GB

Testes:

SPEEDUP	**3,60**	Sem Modificação (linha 88)	Com Modificação (linha 88)
i	SEQUENCIAL (ms)	PARALELA (ms)	
1	2061,08	766,850	1066,710
2	2629,54	749,216	1193,800
3	2736,26	765,098	1055,270
4	2744,5	755,918	1255,270
5	2831,17	740,916	1438,680
6	2765,33	746,872	964,273
7	2634,17	773,843	1201,560
8	2635,61	828,835	1047,470
9	2744,69	743,625	1224,770
10	2784,79	776,023	1100,960
MÉDIA	**2740,38**	**760,508**	**1147,380**

Entrada: 20000 vértices, 10000 arestas por vértice, seed: 6

SPEEDUP	**3,72**	Sem Modificação (linha 88)	Com Modificação (linha 88)
i	SEQUENCIAL (ms)	PARALELA (ms)	
1	7926,61	2.103,150	4330,610
2	7414,47	2.162,710	4395,410
3	7071,38	2.098,860	4353,010
4	7186,7	1.934,360	3908,300
5	7153,32	1.882,860	4187,660
6	5509,14	1.906,250	4524,990
7	6012,13	1.906,400	4107,690
8	7248,81	1.924,600	4265,490
9	7236,06	1.683,390	4018,920
10	7152,13	1.930,740	4810,160
MÉDIA	**7170,01**	**1.927,670**	**4298,050**

Entrada: 30000 vértices, 20000 arestas por vértice, seed: 6

OBS: É possível notar que a paralelização do laço iniciado na linha 88, responsável por encontrar as menores distâncias para chegar em cada nó, não mostrou melhoras no tempo de execução.