Algoritmo KNN paralelo em cluster com MPI

Henrique L. Rieger[1], **Wagner M. Nunan Zola** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{hlr19, wagner}@inf.ufpr.br

Resumo. *Diversas aplicações de aprendizado de máquina e mineração de dados dependem de algoritmos bastante custosos em bases de dados de larga escala. No entanto, esses algoritmos podem obter ótimos ganhos de desempenho ao paralelizar sua operação, como é o caso do KNN (k-nearest neighbors). Neste trabalho, será apresentada uma implementação paralela em cluster do KNN, utilizando a biblioteca Open MPI e medindo seu tempo e aceleração.*

1. Introdução

Por vezes, o processamento de grandes conjuntos de dados leva a operações de alto custo computacional, que podem tirar bastante proveito de paralelizar a carga de trabalho para obter desempenho. Um destes algoritmos, bastante usado no contexto de mineração de dados e aprendizado de máquina, é o KNN (k-*nearest neighbors*, ou k-vizinhos mais próximos). Diversas pesquisas foram realizadas para otimizar o algoritmo KNN e suas variações utilizando computação paralela. Em [Zhang et al. 2012], foi implementada uma versão em cluster do KNN usando Hadoop MapReduce, particionando os conjuntos P e Q, bem como utilizando aproximações e árvores para melhorar o desempenho. Em [Shahvarani and Jacobsen 2021], é proposta uma solução denominada ADS-kNN, usando multiprocessador de 56 núcleos sobre dados dinâmicos. [Meyer et al. 2021] e [Meyer et al. 2022] apresentam uma solução denominada RSFK para aproximar o resultado de KNN utilizando GPUs.

Neste trabalho, será investigada a escalabilidade do algoritmo KNN exato em um *cluster* de computadores utilizando a biblioteca Open MPI. Serão analisados o tempo de execução e aceleração (*speedup*) em relação ao número de processos utilizados em cada uma das máquinas, tamanhos de conjunto de teste (P) e amostra (Q), e tamanho de k.

2. Fundamentos teóricos e Implementação

O algoritmo exato KNN tem como entrada dois conjuntos de pontos P e Q, em que cada ponto pertence a $\mathbb{R}^d$. A saída é um conjunto resultado R contendo $|Q|$ outros conjuntos de tamanho k, onde k é algum inteiro pré-definido. Cada elemento de R corresponde aos k pontos de P mais próximos do ponto em Q usando alguma função de comparação, normalmente a distância euclidiana. A Figura 1 ilustra o resultado descrito.

O algoritmo KNN foi implementado na linguagem C, utilizando diretivas da biblioteca Open MPI para permitir a paralelização. A versão sequencial, usada como base para a medição do *speedup*, consiste apenas da execução desse código para um único processo MPI. Os conjuntos P, Q e R foram implementados como matrizes de *floats* de dimensões $n \times d$, $n_q \times d$ e $n_q \times k$, respectivamente. As matrizes P e Q foram criadas artificialmente diversas vezes a partir de um gerador de números pseudo-aleatórios variando

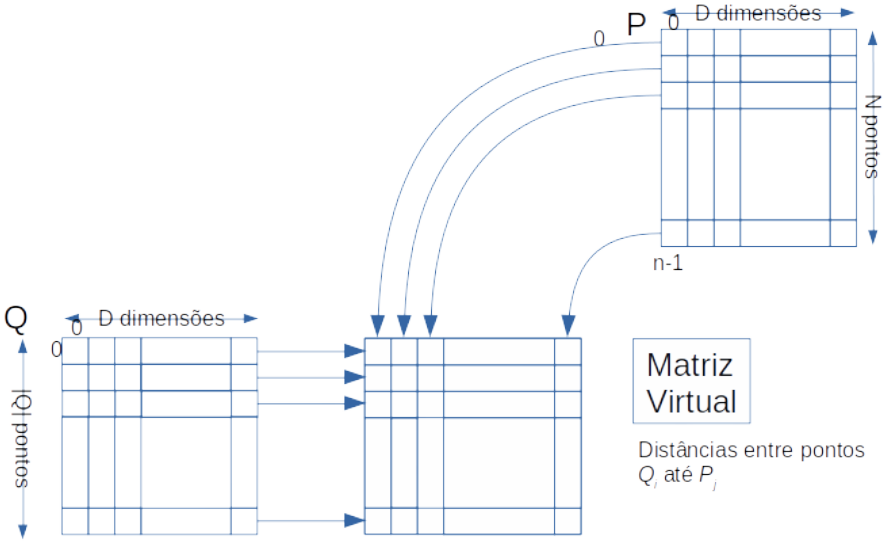


Figura 1. Ilustração do funcionamento do algoritmo KNN.

Nome	Número de pontos	Número de dimensões
MNIST	70000	784
ImageNet	1275219	128

Tabela 1. Datsets utilizados

a semente a cada experimento, a fim de reduzir tempos de leitura de potenciais bancos de dados externos. Os tamanhos escolhidos para as experiências simulam duas bases de dados existentes: MNIST e ImageNet. As dimensões de cada conjunto estão listadas na Tabela 1. As dimensões dos pontos em Q em cada experimento foram ajustadas para corresponder às bases geradas. A matriz R contém em cada linha os índices dos k pontos em P mais próximos do ponto representado na linha correspondente em Q.

O paralelismo foi aplicado na divisão do cálculo das distâncias de cada ponto do vetor Q em *chunks* designados a cada processo MPI. Através da diretiva `MPI_Scatter`, o vetor de pontos foi distribuído em pedaços menores, e os resultados depois agrupados usando a função `MPI_Gather`. Para o cálculo das distâncias, foi aplicada uma matriz virtual, que mantinha apenas os resultados das k distâncias mais próximas calculadas até o momento, indicando a maior delas. Esses valores eram então substituídos conforme novas distâncias eram calculadas. A matriz de distâncias é virtual pois representa o cálculo das distâncias entre os pontos de Q aos pontos em P. Ou seja, é uma matriz de valores $Q \times P$, mas esses não são efetivamente mantidos em memória em forma de matriz. Cada nova distância $D(Q_i, P_j)$ é calculada e mantida em variáveis locais, sendo usada para manter os conjuntos KNN. Assim, seja P_m o ponto de maior distância $D(Q_i, P_m)$ que está no conjunto KNN_i. Se uma nova distância calculada a um ponto P_j é menor que a distância a esse ponto P_m, então o ponto P_j substitui P_m no conjunto KNN_i.

3. Metodologia e resultados experimentais

Os experimentos foram feitos em quatro computadores idênticos, com processadores Intel i7-4770, com 4 núcleos, 2 *hyperthreads*/núcleo, *clock* máximo de 3,9 GHz e 8GiB de

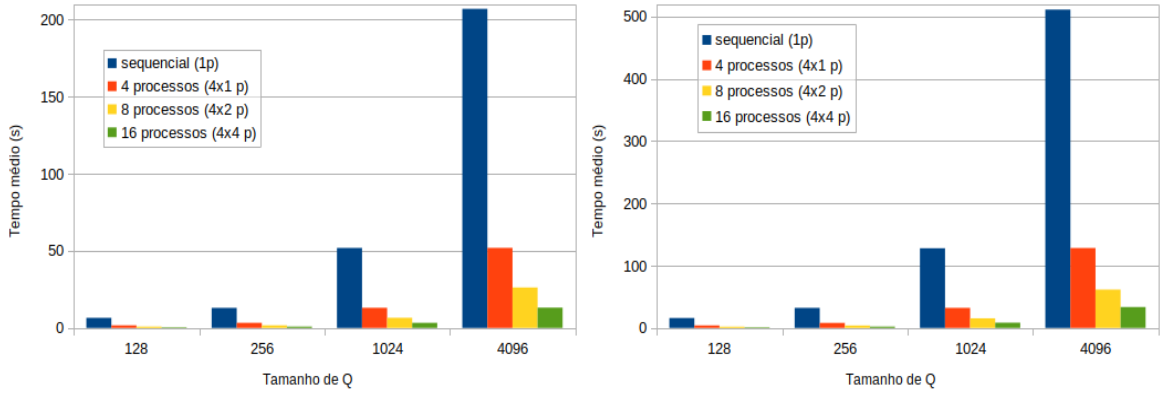
MNIST – Speedup				
| Q |	128	256	1024	4096
sequencial (1p)	1,00	1,00	1,00	1,00
4 processos (4x1 p)	3,99	3,99	3,99	3,99
8 processos (4x2 p)	7,87	7,90	7,90	7,90
16 processos (4x4 p)	15,49	15,73	15,76	15,76

IMAGENET – Speedup				
| Q |	128	256	1024	4096
sequencial (1p)	1,00	1,00	1,00	1,00
4 processos (4x1 p)	3,99	4,01	3,99	3,98
8 processos (4x2 p)	8,24	8,33	8,31	8,31
16 processos (4x4 p)	15,26	15,31	15,30	15,30

Tabela 2. Aceleração média de execução para cada base de dados com os mesmos experimentos representados na Figura 2

memória RAM. Para a comunicação dentro do *cluster*, foi utilizada a biblioteca Open MPI versão 4.1.0. As experiências foram feitas para vetores de pontos Q de tamanho 128, 256, 1024 e 4096, e k de tamanho 32, 128 e 256. O tempo foi medido apenas no processo principal (*rank = 0*) e apenas para o cálculo da matriz R. Para garantir a medição correta, os tempos iniciais e finais foram mensurados apenas após barreiras com todos os processos. O tempo de execução sequencial também foi medido com outra versão do algoritmo sem diretivas MPI. Esses resultados não foram inclusos pois, em média, a sobrecarga de tempo da versão com apenas 1 processo MPI não ultrapassou 1,4%.

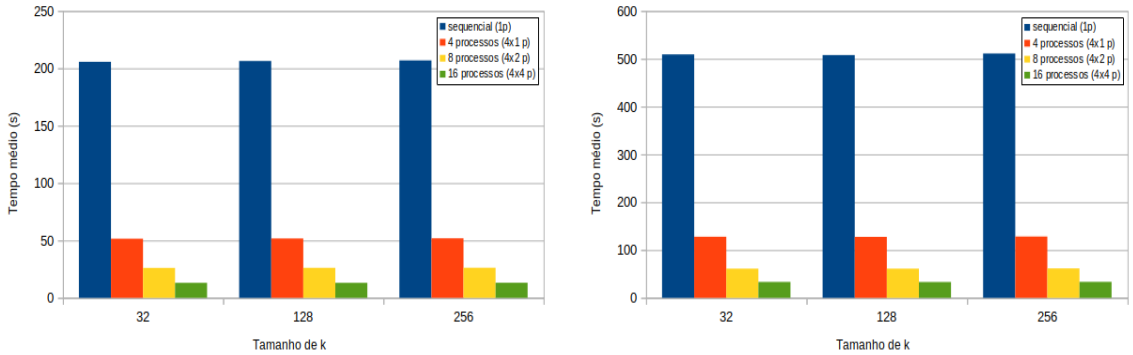
Foram analisados os tempos de execução de ambas as bases artificiais listadas acima, variando os tamanhos de Q e k conforme descrito no início da seção. Foram comparados os tempos médios de 30 execuções para o caso sequencial e com a carga de trabalho dividida entre as quatro máquinas, avaliando também o desempenho para cada uma rodando 1, 2 e 4 processos. Vale ressaltar que os computadores utilizados são compartilhados por diversos usuários e, embora tentativas tenham sido realizadas para minimizar a interferência externa no experimento, os tempos de execução podem variar por usos não esperados. A Figura 2 apresenta o tempo médio de cada execução para cada base de dados com diferentes tamanhos de Q, fixando $k = 256$. A Tabela 2 mostra a aceleração média nos mesmos experimentos para cada divisão dos processos no *cluster*. Podemos notar que o algoritmo apresentou excelente escalabilidade. Por fim, a Figura 3 mostra as diferenças de tempo entre os experimentos para $|Q| = 4096$, variando apenas k. É possível notar que a variação de k afeta em muito pouco a execução do algoritmo.



(a) Base de dados MNIST

(b) Base de dados ImageNet

Figura 2. Tempos médios de execução para cada base de dados e variando-se o tamanho do conjunto Q de pesquisa e o número de processos MPI, com k fixado em 256.



(a) Base de dados MNIST

(b) Base de dados ImageNet

Figura 3. Tempos médios de execução para cada base de dados e variando-se o tamanho de k e o número de processos MPI. Para esses experimentos, $|Q|$ foi fixado em 4096.

4. Conclusões

Neste artigo, foi apresentado uma opção de paralelização em cluster do algoritmo KNN exato, que proporciona aceleração praticamente igual e por vezes, superior (superlinear), ao número de processos em que foi dividido, de maneira praticamente independente da quantidade k de vizinhos. O resultado é bastante semelhante ao apresentado em [Zhang et al. 2012], no qual o algoritmo exato escala de maneira direta com a quantidade de máquinas no *cluster*, embora sejam notáveis diferenças ao variar k. Vale ressaltar que, neste trabalho, o tamanho de k é relativamente pequeno em comparação à quantidade de pontos em P. Os resultados demonstram que a biblioteca Open MPI apresenta excelente desempenho para comunicação entre processos em *cluster*, mesmo considerando as comunicações em rede necessárias para o desempenho correto do algoritmo.

Como sugestões de trabalhos futuros, as operações do cálculo das distância poderiam ser implementadas utilizando instruções SIMD, além das otimizações feitas automaticamente pelo compilador. Também podem ser utilizados outros conjuntos maiores de dados, como o GoogleNews300 utilizado em [Meyer et al. 2021] e tamanhos de k mais próximos do tamanho de P.

Referências

Meyer, B., Pozo, A., and Nunan Zola, W. M. (2021). Warp-centric k-nearest neighbor graphs construction on GPU. In *50th International Conference on Parallel Processing Workshop*, ICPP Workshops '21, New York, NY, USA. Pub. ACM.

Meyer, B., Pozo, A., and Zola, W. (2022). ANN-RSFK: Busca genérica de similaridade em GPU. In *Anais da XXII Escola Regional de Alto Desempenho da Região Sul*, Porto Alegre, RS, Brasil. SBC.

Shahvarani, A. and Jacobsen, H.-A. (2021). Distributed stream kNN join. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1597–1609, New York, NY, USA. Association for Computing Machinery.

Zhang, C., Li, F., and Jestes, J. (2012). Efficient parallel kNN joins for large data in MapReduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, New York, NY, USA. Pub. ACM.