

trab2-knnMPI: Determinar conjuntos de K vizinhos mais próximos (KNN) com processos MPI

Histórico:

- v1.0 a versão inicial, - v2.1 adicionado pequenos detalhes, retirado um erro no texto

Data do enunciado: 06/nov/2023

Data da entrega: **19/nov/2023**

PS: note que esse trabalho reaproveita código dos exercícios feitos nas duas últimas aulas bem como parte do trab 1

**Esse trab pode ser em grupo de no MAXIMO 2 alunos
o nome dos autores devem estar no relatório e no progr. Fonte
A entrega será via UFPR Virtual**

Objetivo do trabalho 2:

O objetivo deste exercício é fazer um trabalho que use paralelismo em MPI no cluster com rede ethernet e verificar a aceleração da nossa solução.

.O trabalho:

A entrada para nossa função paralela são dois conjuntos de pontos de D dimensões cada ponto. São pontos de muitas dimensões.

Dados os conjuntos de n pontos P e Q, denominados respectivamente

P = Pontos da base (dataset) (matriz de floats)

Q = Pontos de consulta, send |Q| o tamanho de Q. (matriz de floats)

Dado um número inteiro k

Para cada ponto em Q, queremos determinar quais são seus k vizinhos mais próximos no conjunto P.

Fazer uma função knn que recebe como parâmetros: Q, nq, P, n, D, k

sendo: nq = número de pontos em Q

n = número de pontos em P

D = número de dimensões dos pontos

k tamanho dos conjuntos de pontos vizinhos buscados

Note que o resultado (saída) da função knn deve ter nq conjuntos de tamanho k

Cada conjunto deve ter o índice de seus k vizinhos mais próximos

Ou seja, a saída de knn pode ser uma matriz de nq linhas por k colunas.

A figura na página seguinte ilustra o problema.

Nosso programa MPI deve obter as matrizes Q e P.

O conteúdo dessas matrizes poderia ser obtido de arquivos MAS

como o conjunto P pode ser grande, a leitura pode demorar,

então vamos gerar esses conjuntos com dados de maneira aleatória no processo 0 MPI (raiz).

Usando uma função geraConjuntoDeDados(C, nc, D)

que gera um conjunto qualquer C de nc pontos aleatórios de D dimensões.

Todos as nossas coordenadas dos pontos em Q e P devem ser float.

Somente o processo 0 deve gerar os conjuntos Q e P.

Para nossas experiências queremos um conjunto Q de 128 pontos de 300 dimensões e um conjunto P de 400mil pontos de 300 dimensões

- **SOBRE A IMPLEMENTAÇÃO:**

- Sua solução DEVE usar a adequadamente implementação do HEAP (chave, valor) e operação decreaseMax feitos no trab 1
- Você DEVE usar a operação scatter de MPI na matriz **Q** da raiz para os nodos e a operação broadcast para enviar a matriz **P** para todos os nodos, e gather para “juntar” o resultado R na RAIZ
- todas as distancias calculadas, comparadas e mantidas no heap são **distancias quadraticas (distancia ao quadrado d^2 , ou seja, NAO vamos usar operacoes de raiz quadrada** e nosso programa rodará mais rapido
- Todos os resultados reportados deve ser medidos no nosso cluster XEON e você deve fornecer os scripts para compilacao e para rodar suas experiencias, bem como as planilhas openoffice com seus resultados e graficos ALEM do relatório
- Ordem dos parametros para rodar o programa

`mpirun -np 4 knn-mpi nq npp d k`

onde: nq = numero de pontos em Q (tamanho do conjunto de pesquisa)

npp = numero de pontos em P (dataset)

d = numero de dimensoes dos pontos (dimensionalidade)

k = tamanho de cada conjunto de vizinhos (k vizinhos por ponto)

- ex:

`mpirun -np 4 knn-mpi 128 400000 300 1024`

Calcula os 1024 vizinhos mais próximos de 128 pontos de 300 dimensoes em base de dados de 400mil pontos

- **SOBRE AS EXPERIÊNCIAS:**

- TODAS os resultados reportados DEVEM ser
- Faremos a computacao total para 3 casos apenas:
-
- Experiencia 1:
- Rodar o programa para APENAS 1 processo MPI e medir o tempo da computação de knn
-
- Experiencia 2:
- Rodar o programa para 4 processos MPI no mesmo host e medir o tempo da computação de knn
-
- Experiencia 3:
- Rodar o programa para 4 processos MPI em hosts diferentes e medir o tempo da computação de knn
-
- Reportar a aceleração (speedup) da sua implementação paralela em relação a 1 processo apenas, nos casos da experiencia 2 e 3.

- **SOBRE O RELATÓRIO do trabalho:**

- Fazer um relatório (PDF) com suas conclusões. Descreva as características importantes da sua CPU usada. Inclua em apendice ao relatório o texto da saída do comando `lscpu` bem como a figura do comando `lstopo` para sua CPU usada.

- **SOBRE A função de verificação a ser chamada (próxima página)**

- SOBRE A função de verificação a ser chamada
 - Incluir ao final do seu programa, ANTES de terminar o MPI e ANTES de desalocar suas matrizes, para rodar APENAS no processo 0 MPI, a função verificaKNN(...) que DEVE ter o mesmo prototipo abaixo. Você deve colocar a função inicial de verificação abaixo em um arquivo chamado verificaKNN.c e fazer o include desse arquivo no INICIO do seu programa.
 - A função inicial está abaixo, esta apenas vai imprimir o vetor de resultados. Para a verificação adequada o prof irá substituir essa versão por outra que fará a verificação.

```
void verificaKNN( float *Q, int nq, float *P, int n, int D, int k, int *R ) {
    // note que R tem nq linhas por k colunas, para qualquer tamanho de k (colunas)
    // então é linearizado para acesso como um VETOR
    printf( " ----- VERIFICA KNN ----- " );
    for( int linha=0; linha<nq; linha++ ) {
        printf( "knn[%d]: ", linha );
        for( int coluna=0; coluna<k; coluna++ )
            printf( "%d ", R[ linha*k+coluna ] );
        printf( "\n" );
    }
}
```

- **OBS:** será na página seguinte temos a ilustração do problema

ilustração do problema

