Relatório do processo Kmeans paralelizável

Escrito no âmbito do Projeto Final de Fundamentos de Computação de Alto Desempenho

Trabalho realizado por:
Jorge Couto Nº58656
Miguel Grilo Nº58387

Algo importante a mencionar

Antes de começarmos, de facto, a explicar o trabalho, o funcionamento de cada função e outros, achamos pertinente explicar o porquê da ausência da entrega do trabalho antes do dia 5. Como ficou esclarecido, o prazo de entrega foi alterado para dia 5 de Julho. Todavia, pelo modo como foi dito, achávamos que o prazo final de entrega era até à meia noite (23:59) de dia 5, ou seja, que ainda tínhamos algumas horas antes do prazo final de entrega de dia 5. Infelizmente, só percebemos mais tarde que dia 5 não contava para a entrega. Esperemos que compreenda a entrega tardia, e que não nos penalize por este pequeno engano. Tendo explicado esse percalço, podemos seguir para a explicação do trabalho.

Preparação do programa: prep.sh

O script de bash prep.sh é responsável pela preparação do ambiente do trabalho. Primeiro, o script certifica-se de atualizar a lista de pacotes e os pacotes, para o sistema. Depois, certifica-se de instalar/atualizar o libopenmpi-dev, openmpi-bin, libhdf5-dev, libjansson-dev e o build-essential. Primeiro, verifica se cada um dos pacotes já foi instalado previamente e, se não, instá-la-os. Naturalmente, colocamos verificações de 'else' para caso houvesse algum erro a instalar os pacotes. Para executar a preparação do ambiente, é preciso executar o comando ./prep.sh.

Compilar e correr o programa: build.sh

O script de bash build.sh permite compilar e correr o programa, tendo sido criado por dois motivos, um deles pedido no projeto e outro para ultrapassar um obstáculo que surgiu na criação do trabalho: Primeiro, a necessidade de executar o programa através de uma linha de código específica remeteu-nos à necessidade de criar um script que permitisse executar o kmeans_paralelo.c do modo pedido. Todavia, acabava por não dar para executar 'normalmente'. Um problema irregular no programa fazia com que as bibliotecas de leitura de ficheiros JSON e H5 não fossem identificadas, como se nunca tivessem sido instaladas. Para isso, optamos por correr a compilação do programa através do mpicc (também nos permitindo temporizar o programa automaticamente). Por isso, fizemos a linha de execução de um modo alterno: Os argumentos da linha de execução são variáveis no build.sh, as quais podem ser alteradas diretamente no script dependendo da vontade do utilizador. Por exemplo, se quiser correr um só processo de kmeans para o ficheiro CSV de 20 pontos em 2 dimensões com 2 clusters, pode mudar a variável datafile para "points_20_2.csv" (dependendo do nome e da localização do ficheiro csv) e a variável K para 2. O build.sh irá correr o programa automaticamente usando mpi.

O ficheiro .h: kmeans_paralelo.h

Não existe muito a dizer sobre este ficheiro, sendo apenas a chamada das funções que serão usadas no .c. O que existe de importante neste ficheiro são os defines e os structs: Os defines permitem dar limites à quantidade de clusters, dimensões e pontos a se mexer. Para atualizar algum dos limites pode-se editar por completo (caso queira mais que 10 clusters, ou caso queira mexer com um ficheiro que tenha mais que 50000 pontos). Por outro lado, os structs definem informações importantes, essas sendo as coordenadas dos pontos (definida no struct Point) e as informações do 'Dataset' em si (o número de pontos, as dimensões e as coordenadas dos pontos, importante para se usar com os defines já definidos).

Kmeans_paralelo: A leitura de CSV's

De um modo aproximado ao feito pelo professor no exemplo disponibilizado, a leitura de ficheiros CSV é feita através de duas funções: A função read_line, que simplesmente lê cada linha, servindo de verificação para outra, e a função read_csv que, como o nome indica, lê o ficheiro csv em si. A leitura de linha cria um buffer temporário de espaço limitado e guarda o conteúdo da linha, lendo quantas dimensões (quantos valores separados por vírgulas) existem, encerrando caso haja mais dimensões do que o máximo definido. Já o read_csv abre o ficheiro dado para leitura, passando à frente a primeira linha (que é nula, tendo apenas x0,x1…), e aloca a memória para os pontos de dados. Depois de alocar a memória, lê o arquivo linha por linha (através de read_line) e armazena os pontos de dados em Dataset, para serem depois usados no kmeans. Adicionamos algumas verificações, encerrando o código se, por exemplo, forem lidos mais pontos que o máximo permitido.

A leitura de H5 e JSON

A leitura de ficheiros H5 e JSON já se faz em uma só função. No caso de JSON, são primeiras feitas as verificações sobre se surgiu algum erro a analisar o ficheiro, se trata-se de um array ou não, se têm mais pontos que o máximo definido e outros. Depois disso, aloca-se a memória dos pontos e, em seguida, começa-se o processamento dos pontos. Para cada ponto no array, obtem-se o objeto JSON correspondente ao ponto, faz-se as verificações necessárias, aloca-se memória das coordenadas, itera-se sobre cada chave-valor no objeto do ponto para armazenar (se for um número) nas coordenadas e verifica-se, também, se o número de dimensões lidas é consistente com as dimensões lidas anteriores. No final, é liberta a memória ocupada pelo objeto JSON raíz. A função de H5 primeiro abre o ficheiro H5, abre o dataset no ficheiro, obtêm o dataspace do dataset, verificam e obtêm as dimensões do dataspace. Depois das verificações do costume (se excede as dimensões máximas), configura o dataset e aloca a memória para as coordenadas dos pontos, antes de ler os dados do dataset e copiá-los para a estrutura dataset. Tal e qual as outras funções, no fim liberta a memória.

Escrita do Output

A escrita do Output é feita pela função write_output que, usando os nomes para os ficheiros de output fornecidos nos argumentos de build.sh, escreve o ficheiro de output dos centroides e dos clusters. Para os clusters, abre o arquivo dos clusters (e cria se não existir) em modo de escrita e itera sobre os pontos do dataset, escrevendo, para cada ponto, o número do respetivo cluster, e em frente (através do segundo for) cada dimensão do ponto, assim garantindo que são fornecidas as coordenadas de cada ponto para cada cluster (ex: 1:57,3), o cluster 1 apresenta nele o ponto (57,3). Depois disso, fecha-se o arquivo dos clusters e abre-se o dos centroides que, por uma lógica semelhante, percorre pelo primeiro for todos os centroides (k sendo o número de clusters, logo, de centroides) e, para cada centroide, escreve o respetivo índice. No final, é fechado o arquivo.

O Kmeans, feito por 4 funções

O Kmeans é feito por uma lógica semelhante àquela do exemplo: Apresenta 4 funções centrais que tratam do kmeans (descontando a função de distância, igual à do exemplo por não haver onde variar, e a de inicialização que, simplesmente, inicializa a semente do gerador de números aleatórios).
A função initialize_centroids usa a função initialize_random_seed para, iterando sobre o número de clusters (quantos centroides serão precisos), definir as coordenadas iniciais dos centroides e as dimensões. A função assign_clusters define os clusters em si, avaliando a distância entre os centroides e os pontos e dando, a cada centroide, o ponto mais próximo. A variável changes é usada para verificar se ocorrem mudanças nos centroides. Se não ocorrerem, é porque o centroide está definido, e o kmeans pode encerrar. A função update_centroids torna todos os centroides 0, e calcula as suas novas coordenadas com base nos clusters. Por fim, a função kmeans simplesmente corre o kmeans como um ciclo, primeiro inicializando os centroides, as iterações e as mudanças, e parando apenas quando os centroides deixam de sofrer mudanças ou são feitas 100 iterações.

A função Main

A função Main é responsável por, com uso do MPI, permitindo realizar o programa k-means com paralelização. Primeiro, a função Main verifica se foi dada a quantidade de argumentos correta e, depois, inicializa o ambiente MPI com MPI_Init. As variáveis rank e size são usadas para obter o rank do processo atual e o número total de processos, respetivamente. Usando os argumentos dados, são definidos K (número de clusters) e N (número de processos), com N sendo um só processo (sem paralelização). O Main também se responsabiliza por verificar qual o ficheiro lido, analisando qual a extensão do arquivo a ser lido fornecido nos argumentos: Se é .csv, .json, .h5 ou nenhum desses, nesse caso encerrando o programa mais cedo. Dependendo da extensão do arquivo, é chamada a respetiva função de leitura para preencher uma variável dataset, do tipo de dado Dataset.

Depois disso é feita a alocação de memória para os centroides e atribuições e, depois de tudo isso, são divididos os dados locais para os diferentes processos MPI. Apenas depois de toda essa preparação é que se executa o kmeans, sendo os dados recolhidos com MPI_Gather. No final, são descritos os resultados através de write_output, liberada a memória alocada e finalizado o MPI, antes do return 0 que encerra o programa.

Erros no Programa

Infelizmente, sofremos diversos problemas no programa, especialmente quando se trata de Segmentation Fault's. O maior problema, que permanece ainda no trabalho e não conseguimos resolver a tempo (tendo ficado até hoje a tentar tratar deste erro), trata-se de um estranho Segmentation Fault que ocorre quando tentamos paralelizar o programa. Estranhamente, o erro parece tornar-se mais comum quando cuidamos de dados maiores, e dependendo de quantos clusters temos: Por exemplo, o erro raramente ocorre com o ficheiro points_20_2.csv, podendo nós paralelizar com até mesmo 4 programas sem grandes problemas. Com 100 pontos também corremos o programa sem dificuldades dependendo de quantos clusters usamos para os processos. Até mesmo com 500 pontos o erro surge, dependendo de quantos clusters usemos, entretanto não encontramos nenhum padrão por detrás do segmentation fault. Queríamos terminar de corrigir o erro, tendo percebido um possível problema. Entretanto, a falta de tempo não nos permite corrigir a tempo. Esperemos que entenda e que, tendo em conta que a paralelização é ainda possível com dados menores, não desconte muito.