Enunciado do trabalho prático 1

Objetivo

Este trabalho tem como objetivo avaliar a aprendizagem das técnicas de otimização de código, incluindo as técnicas/ferramentas de análise de código.

Introdução

Os alunos deverão desenvolver, em C, uma versão sequencial otimizada de um algoritmo *k-means* simples, baseado no algoritmo de Lloyd (https://datasciencelab.wordpress.com/tag/lloyds-algorithm/). O algoritmo classifica um conjunto de amostras em k grupos (*clusters*), podendo ser resumido nos seguintes passos:

```
1. [Criar as amostras e] iniciar os "clusters"
   a. [Iniciar um vetor com valores aleatórios (N amostras no espaço (x,y) )]
   b. Iniciar os K clusters com as coordenadas das primeiras K amostras
   c. Atribuir cada amostra ao cluster mais próximo usando a distância euclidiana
2. Calcular o centroide de cada "cluster" (também conhecido como centro geométrico)
3. Atribuir cada amostra ao "cluster" mais próximo usando a distância euclidiana
4. Repetir os passos 2 e 3 até não existirem pontos que mudem de "cluster"
```

Neste primeiro trabalho deverá ser construída uma função para criar com conjunto de amostras para teste (passo 1a) do algoritmo; ver anexo I). Os alunos têm a liberdade para fazer otimizações ao algoritmo proposto tendo como objetivo a minimização do tempo total de execução do algoritmo (otimização da implementação das funções, estruturas de dados, etc.). No entanto deverão manter a filosofia do algoritmo original (calcular os centroides, atribuir amostras aos "clusters" mais próximos, etc; o resultado deve ser o indicado no anexo I.)

Devido à grande variedade de otimizações que podem ser realizadas no trabalho, sugere-se que todas as otimizações sejam precedidas de uma análise do impacto das mesmas, ou da obtenção de métricas que indiciem um impacto significativo no desempenho.

Resultados a apresentar

Na submissão do trabalho deverão incluir os tempos de execução obtidos na máquina do Search, usando 10 000 000 amostras (pontos no espaço X,Y representados por dois valores do tipo *float*) e 4 clusters (cada cluster representado pelo seu centroide (x,y), dois valores do tipo *float*). O código deverá ser genérico podendo ser facilmente modificado para usar mais amostras e/ou "clusters".

Grupos, estrutura do código/relatório e datas

Os grupos poderão ser constituídos por um ou dois alunos (não são permitidos grupos de três alunos!).
O código e respetiva `Makefile` têm que cumprir as regras definidas no Anexo I.
O relatório será em formato *pdf*, com um máximo de 2 páginas, excluindo anexos (os anexos só serão consultados se o docente achar relevantes). Deverão usar o *template* IEEE (em https://www.ieee.org/conferences/publishing/templates.html).
O relatório e um ficheiro *zip* com o código e respetiva `Makefile` deverão submetidos na plataforma de *elearning* até ás **23:59 horas do dia 4-Nov-22**.
A defesa deste primeiro trabalho será realizada juntamente com a entrega do TP3, em Janeiro de 2023. Nessa defesa poderá ser revista a nota desta primeira entrega.

Critérios de avaliação

A nota deste trabalho, terá em consideração: **(i)** o desempenho do código desenvolvido **(50%)**; **(ii)** as otimizações realizadas no código e a correspondente justificação (valoriza-se a utilização de modelos/métricas que expliquem as opções/resultados) **(30%) (iii)** Legibilidade do código e relatório **(20%).**

Anexo I - Estrutura do código e *Makefile* a submeter

1. Inicialização dos valores dos pontos e dos *clusters*

As amostras (pontos no espaço (x,y)) e os *clusters* deverão ser inicializados de forma aleatória, através de uma função com a seguinte estrutura:

```
void inicializa(...) {
    ...
    srand(10);
    for(int i = 0; i < numero_de_pontos; i++) {
        <<ponto_i_coordenada_x>> = (float) rand() / RAND_MAX;
        <<ponto_i_coordenada_y>> = (float) rand() / RAND_MAX;
    }
    for(int i = 0; i < numero_de_clusters; i++) {
        <<cluster_i_coordenada_x>> = <<ponto_i_coordenada_x>>
        <<cluster_i_coordenada_y>> = <<ponto_i_coordenada_y>>
    }
}
```

É importante a inicialização dos valores pela ordem indicada, caso contrário o resultado será diferente do indicado no ponto seguinte. O número de amostras e de "clusters" será indicado diretamente no código usando `#define N 10000000` e `#define K 4`. A função `main` não deverá ter argumentos.

2. Validação do resultado e *output* do programa

O algoritmo corretamente implementado converge ao fim de 39 iterações, sem contar o passo 1 (`gcc 7.2.0` no cluster Search, noutras máquinas o resultado pode ser diferente). Neste caso, o *output* do programa deverá ser **EXATAMENTE** o seguinte (N-número de prontos, K-número de *clusters*):

```
N = 10000000, K = 4
Center: (0.250, 0.750) : Size: 2499108
Center: (0.250, 0.250) : Size: 2501256
Center: (0.750, 0.250) : Size: 2499824
Center: (0.750, 0.750) : Size: 2499812
Iterations: 39
```

3. Organização do código e *Makefile*

Deverá ser incluída uma `Makefile` que gere um executável com o nome `k_means` numa subdiretoria `bin`. Os ficheiros fonte devem estar localizados na subdiretoria `src`. O programa pode ser executado com `make run`. Os alunos poderão indicar as "CFLAGS" que entenderem, desde suportadas pelo `gcc 7.2.0` disponível no Search. Exemplo:

```
CC       = gcc
BIN      = bin/
SRC      = src/
INCLUDES = include/
EXEC     = k_means

CFLAGS = ...

.DEFAULT_GOAL = k_means

k_means: $(SRC)k_means.c $(BIN)utils.o
 $(CC) $(CFLAGS) $(SRC)k_means.c $(BIN)utils.o -o $(BIN)$(EXEC)

$(BIN)utils.o: $(SRC)utils.c $(INCLUDES)utils.h
 $(CC) $(CFLAGS) -c $(SRC)utils.c -o $(BIN)utils.o

clean:
 rm -r bin/*

run:
 ./$(BIN)$(EXEC)
```