Relatório do Trabalho Prático 1

Ana Gonçalves
MEI
pg50180

Francisco Toldy
MEI
pg50379

***Abstract*—Este trabalho tem como objetivo a familiarização com técnicas de otimização de código, incluindo as técnicas/ferramentas de análise de código que vão permitir obter uma melhor performance em termos de tempo de execução. Com este objetivo, foi utilizado um problema exemplo com um grande volume de dados que permite observar melhor as otimizações possíveis.**

I. Introdução

O enunciado deste trabalho apresentava como foco principal o desenvolvimento de uma implementação do algoritmo k-means em C. Para tal, foi fornecido uma pequena explicação sobre o funcionamento do mesmo, deixando a cargo dos alunos a aplicação do algoritmo. Além da implementação básica do algoritmo era ainda esperada a apresentação de algumas otimizações , avaliando os ganhos de performance e o impacto de cada uma delas. Nas secções seguintes, será abordado as várias otimizações que foram implementadas. A explicação do raciocínio no desenvolvimento do algoritmo encontra-se em comentários do ficheiro de código final entregue.

II. Otimizações

TABLE I
Média de 4 resultados com O2 Iniciais

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O2	11,7983	1,58E+08	5,09E+10	3,7E+10

A. Cálculo da Distância entre Ponto e Centroids

A fórmula inicial para cálculo das distâncias de cada ponto a um centroid era a da distância euclidiana, ou seja:

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

No entanto, já que para cada um dos 10 milhões de pontos, é feito esse cálculo 5 vezes (a primeira para declarar a distância para o centroid atual e as seguintes 4 para calcular a distância para todos os centroids) , foi tomada a decisão de remover a operação de raiz quadrada e de exponenciação da formula. Isto não teve qualquer efeito nos resultados uma vez que a distância era usada para fins de comparação, uma vez que, para um dado ponto A e dois centroids (1 e 2):

$$\sqrt{D_{A->1}} > \sqrt{D_{A->2}} \tag{2}$$

Tem o mesmo significado para comparação que :

$$D_{A->1} > D_{A->2} \tag{3}$$

Assim, ao removermos sqrt(), diminuímos o número de chamadas de funções num valor teórico de 10 000 000 * 5 * I , em que I é o número de iterações do programa.

A exponenciação de (x1 - x2) e (y1-y2) através da função pow foi também substituída por uma simples multiplicação.

B. Utilização de Estruturas

Inicialmente foi desenvolvido código recorrendo a listas de estruturas manualmente criadas (sem utilização de structs de C). No entanto, tornou difícil a legilibidade do código, e apesar de estruturas em C terem um pequeno overhead, este não vai ter grandes efeitos sobre o tempo de execução. Com a utilização de estruturas de C foi criada assim uma versão mais legível para qualquer um que tenha acesso ao código.

C. Mudança na Forma como Pontos Mudam de Centroid

Numa primeira fase, a mudança de um ponto de centroid consistia na alteração dos dados do centroid antigo, do centroid atual e do ponto, ao todo, 8 operações por alteração de ponto. No entanto, com a alteração para um reset dos centroids, apesar de um aumento de operações, estas vão ser aplicáveis para a utilização de vetorização, mais notável no ciclo que percorre todos os pontos de modo a obter as novas coordenadas no centroid.

D. Redução de Condições IF

O facto de condições IF não serem vectorizáveis, levou a algumas alterações no código para as retirar e substituí-las por loops que seriam vetorizáveis. Foram consideradas substituíveis 3 condições de IF que estavam presentes na função de verificação de mudança de um ponto, levando à sua remoção. Numa versão final, acabámos com efetivamente uma instrução redundante no caso de não existir alteração do centroid, no entanto, não terá efeitos muito negativos pelo facto de ser só 1 instrução e só ser redundante em uma situação específica (quando não existe a mudança de centroid de um ponto).

TABLE II
Média de 4 resultados com O2 e todas as Otimizações anteriores

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O2	6,6129	1,54E+08	2,93E+10	1,76E+10

TABLE III
MÉDIA DE 4 RESULTADOS COM O3 E TODAS AS OTIMIZAÇÕES ANTERIORES

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O3	5,3148	1,55E+08	2,5E+10	1,69E+10

Com estes resultados, verificamos que o resultado era o esperado, uma redução significativa (22%) em termos de tempo de execução. Como também uma redução muito mais significativa em termos de # I que foi reduzido para metade, como também o # de ciclos.

E. Loop Unrolling

Para permitir o loop unrolling não foi necessária alguma alteração à versão atual, sendo apenas necessário a aplicação da flag que obriga ao unrolling.

TABLE IV
MÉDIA DE 4 RESULTADOS COM LOOP UNROLLING COM O2

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O2	5.1386	1,58E+08	2,44E+10	1,55E+10

Os resultados verificaram-se não muito diferenciados da versão anterior, no entanto ocorreu uma ligeira melhoria na média de tempo de execução.

F. Vectorização

De modo a permitir a vectorização, é necessário que os dados estejam consecutivos na memória e por isso foi desenvolvida uma nova versão com os dados armazenados em estruturas com listas. Ficando os parâmetro dos pontos consecutivos, como também os dos centroids.

Esta alteração equivale à transposição da matriz no caso da multiplicação de matrizes.

TABLE V
MÉDIA DE 4 RESULTADOS COM VETORIZAÇÃO COM O2

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O2	5,078103	1,56E+08	2,93E+10	2,02E+10

TABLE VI
MÉDIA DE 4 RESULTADOS COM VETORIZAÇÃO COM O3

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O3	3,937271	1,54E+08	2,5E+10	1,16E+10

G. Fine Tunning

Com a utilização da flag '-msse4' implica o uso de vetores de 124 bits, no entanto é possível aumentar esse valor para 256 bits se os arrays estiverem alinhados. Com esta alteração, testámos os resultados e depáramo-nos com um resultado não favorável com -O2, mas um bom resultado, com -O3 (Table VIII) em termos de #I e número de L1 misses, mais especificamente, uma redução de 8% em relação ao melhor resultado em termos de # I (Table VI) :

TABLE VII
MÉDIA DE 4 RESULTADOS VETORIZAÇÃO DE 256 BITS COM O2

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O2	6,948227	1,42E+08	2,96E+10	2,22E+10

TABLE VIII
MÉDIA DE 4 RESULTADOS VETORIZAÇÃO DE 256 BITS COM O3

Flag	Tempo de Execução	L1 Misses	#I	#Ciclos
-O3	4,178974	1,41E+08	2,31E+10	1,29E+10

III. ALGORITMO RESULTANTE

Após várias tentativas na obtenção dos resultados desejados, não foi possível encontrar uma forma que evite alguns arredondamentos nos valores obtidos entre iterações e por isso o resultado final ficou perto da solução ótima com a seguinte configuração:

Iterations: 36
N = 10000000, k = 4
Center: (0.249977,0.750091) Size: 2499104
Center: (0.249949,0.250126) Size: 2501263
Center: (0.750015,0.249864) Size: 2499823
Center: (0.749942,0.750037) Size: 2499810

Em relação aos resultados é necessário notar que foi feita uma procura exaustiva, inclusive com docentes, da razão para os resultados estarem diferentes dos fornecidos no enunciado. No entanto, esta não levou a qualquer conclusão sendo assim apresentados resultados com um desvio reduzido em relação ao objetivo.

IV. CONCLUSÕES

Como visto com as tabelas anteriores, foram conseguidos os melhores resultados a nível de tempo com a vetorização e compilação O3. Estes resultados devem-se ao facto da optimização -O3 ser bastante mais agressiva que a optimização -O2. Além disso é importante ainda notar que a nível de vetorização há espaço para melhorias, sendo reportada vetorização apenas numa fração dos ciclos existentes na implementação entregue. Ainda é importante mencionar que ao ser utilizada uma estrutura de arrays (equivalente à transposição de matriz dos guiões práticos), em oposição a um array de estruturas, conseguimos melhorar o aspeto de localidade espacial da solução uma vez que os acessos a memória seriam consecutivos.