- Descrição da aplicação e estrutura do código;

A aplicação "*streamcluster*" tem como objetivo fazer o "clustering" ou "agrupamento" de uma "stream" ou "fluxo" de dados. Ou seja, agrupamento de dados em fluxo.

O agrupamento é uma tarefa de segregação de dados em grupos que são similares por algum parâmetro, chamados de "clusters". Esses dados estão em fluxo pois não são categorizados, são produzidos continuamente e, geralmente, devem ser processados em tempo real.

A prática de "clusterização" tem como objetivo a análise exploratória e estatística de dados, sendo relevante a diversas áreas, como reconhecimento de padrões, aprendizagem de máquina, análise de imagens, compressão, entre outros.

O programa segue a estrutura de um algoritmo "k-means" de agrupamento. Ou seja, dado um número K de centros, um arquivo com n pontos d-dimensionais, definidos como argumentos iniciais, é possível encontrar a localização dos centros que melhor separam estes pontos.

Este processo é iterativo, e os centros são recalculados aleatoriamente até que a variância das distâncias centro-pontos não tenha grande alteração. O parâmetro de similaridade é justamente esse, da distância euclidiana.

```
646
647  /* compute Euclidean distance squared between two points */
648  float dist(Point p1, Point p2, int dim)
649  {
650    int i;
651    float result=0.0;
652    for (i=0;i<dim;i++)
653      result += (p1.coord[i] - p2.coord[i])*(p1.coord[i] - p2.coord[i]);
654    return(result);
655  }
656
```

Argumentos passíveis de customização são: K mínimo (k1) , K máximo (k2), número de dimensões (d), número de *data points* (n), número de *data points* processados por iteração (*chunksize*), número máximo de pontos intermediários (*chunksize*), arquivo de entrada (*infile*), arquivo de saída (*outfile*), e número de threads (nproc).

O comando de execução tem formato:
./streamcluster k1 k2 d n chunksize clustersize infile outfile nproc

Caso nenhum arquivo de entrada seja localizado e *n* > 0, serão gerados *n* pontos aleatórios no intervalo real (0, 1), e agrupados.

Para sincronização de *threads* são utilizadas *pthread barriers*, ou barreiras de threads, ao longo de toda execução do código.

Foi feito um teste de agrupamento de 10 pontos bidimensionais em dois grupos:

0.041630 0.454492
0.834817 0.335986
0.565489 0.001767
0.187590 0.990434

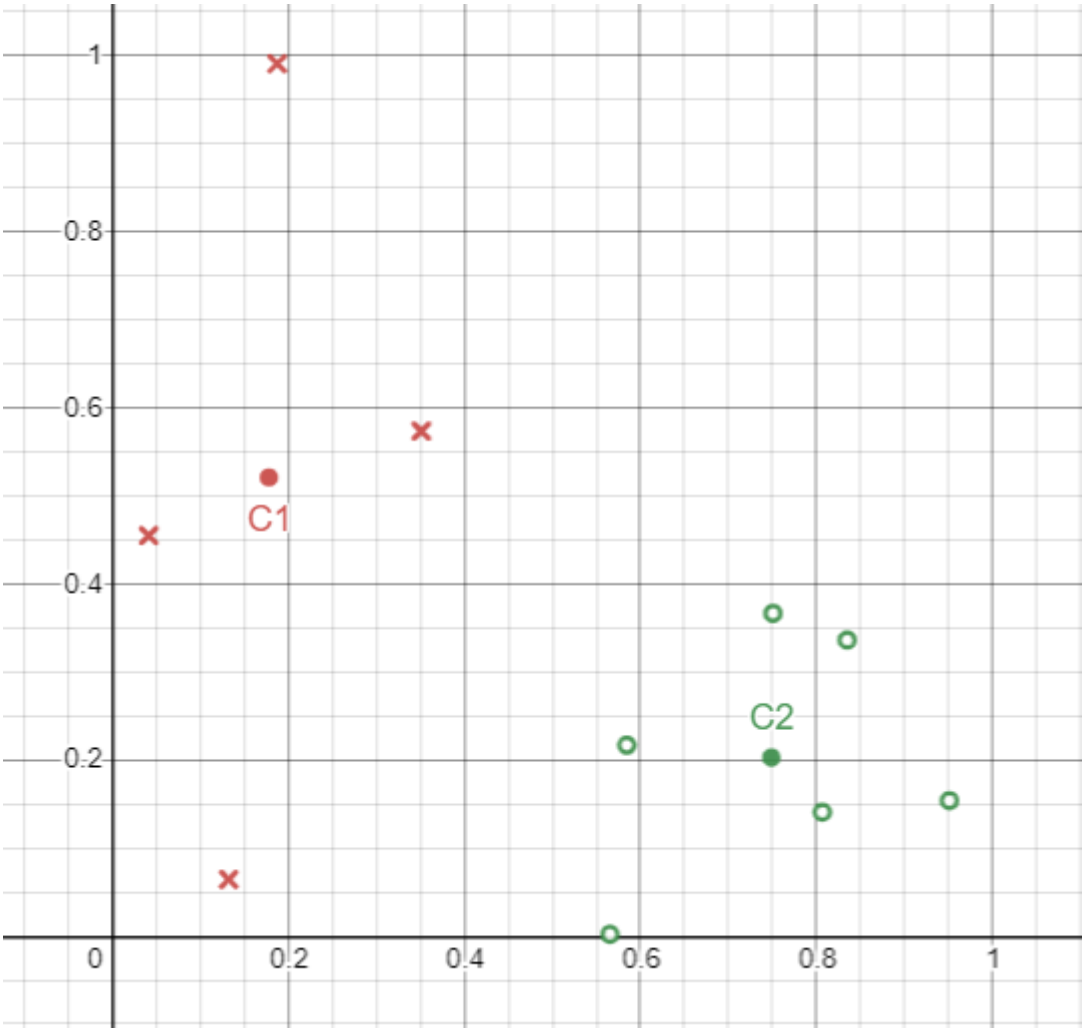
0.750497 0.366274
0.351209 0.573345
0.132554 0.064166
0.950854 0.153560
0.584649 0.216588
0.806502 0.140473

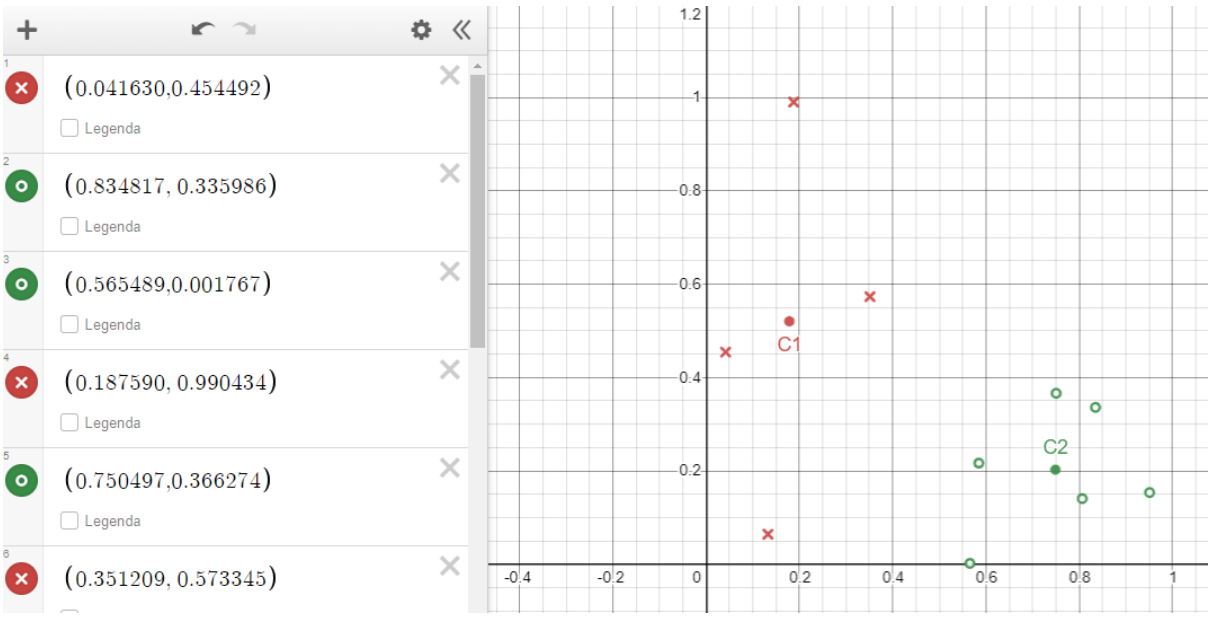
O resultado obtido instantaneamente foi:

4
4.000000
0.178246 0.520609

6
6.000000
0.748801 0.202441

Representa a localização dos dois centros, um com peso 4 e outro com peso 6. Fazendo o mapeamento dos pontos, é possível observar que o resultado foi satisfatório. Dois grupos, um à esquerda e outro à direita foram formados.





Também foi testado para 100 pontos (0,01s), 1000 pontos (0,04s), 10000 pontos (0,8s) e 100000 pontos (15s). Há um crescimento exponencial do tempo, como é esperado.

- Descrição da metodologia de perfilamento (devem ser utilizados o Gprof ou qualquer outro perfilador, ex. Intel VTune Amplifier);

Após verificar que o código estava funcionando, o perfilamento foi realizado com o uso da ferramenta Gprof. O projeto foi então compilado, utilizando a variável de ambiente, através de 3 comandos com a flag “-pg”:

/usr/bin/g++ -pg -O3 -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -c streamcluster.cpp

/usr/bin/g++ -pg -O3 -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -c parsec_barrier.cpp

/usr/bin/g++ -pg -O3 -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC_VERSION=3.0-beta-20150206 -DENABLE_THREADS -pthread -L/usr/lib64 -L/usr/lib streamcluster.o parsec_barrier.o -o streamcluster

Inicialmente foram compilados os arquivos *streamcluster.o* e *parsec_barrier.o*, e por fim o *streamcluster*. A flag “-pg” foi usada para gerar o arquivo *gmon.out*, arquivo de análise Gprof.

Vários testes foram feitos, verificando como diferentes configurações afetam o fluxo do código.

- Resultados do perfilamento em uma CPU e associação dos gargalos com a estrutura do código;

Após análise do perfilamento foi observado que a maior parte das chamadas e tempo de processamento estão distribuídas entre 5 funções específicas, 2 delas tomando mais de 90% do tempo de execução para todas as configurações aferidas: “*pgain*” e “*parsec_barrier_wait*”.

“pgain” é a função mais algoritmicamente exaustiva, já que nela é calculada o ganho na variância entre centros, para todos os pontos. Possui 2 “for” aninhados portanto sua complexidade é de O(n^2). Esta e suas auxiliares são as que mais geram gargalo.

“parsec_barrier_wait” por sua vez implementa barreiras e para sincronização de *threads*. Gera latência pois as threads ficam ociosas enquanto esperam umas às outras. Muito tempo foi perdido mesmo quando *nproc* = 1, ou seja, com uma thread, o que teoricamente não deveria acontecer já que não há o que sincronizar quando se tem apenas um processo. Por fim, a maior parte de suas chamadas vem de “*pgain*”.

O tempo de execução aumenta com o número de pontos, mas não tanto com o aumento da dimensionalidade e número de agrupamentos. Foi observado que o tempo de execução cai drasticamente com o aumento dos chunks por interação (*chunksize*), já que mais pontos eram lidos de uma vez, reduzindo a latência de leitura.

Também foi observado uma maior latência à medida que se aumenta o número de threads, devido especialmente ao aumento de tempo gasto nas barreiras de *threads*, e “*for*”s que escalam com nproc.

```
1076    // at this time, we can calculate the cost of opening a center
1077    // at x; if it is negative, we'll go through with opening it
1078
1079    for ( int i = k1; i < k2; i++ ) {
1080      if( is_center[i] ) {
1081        double low = z;
1082        //aggregate from all threads
1083        for( int p = 0; p < nproc; p++ ) {
1084    low += work_mem[center_table[i]+p*stride];
1085        }
1086        gl_lower[center_table[i]] = low;
1087        if ( low > 0 ) {
1088    // i is a median, and
1089    // if we were to open x (which we still may not) we'd close i
1090
```

- Conclusões.

Com o teste de perfilagem foi possível atestar algumas possíveis causas para um gargalo no agrupamento de números elevados de pontos, para uma CPU. Dentre eles: uso de *chunksize* diminuto, isso leva o núcleo a demorar muito somente na leitura dos pontos. Quando lê em pedaços maiores, reduz-se a latência.

Também, funções como “*pgain”* parecem gastar bastante tempo (aprox. 80%+), especialmente por ter maior parte dos loops no programa, ser a função mais algebricamente dispendiosa e ter complexidade O(n^2). “*pgain”* também chama diversas funções auxiliares, como *“parsec_barrier_wait”*, a segunda mais dispendiosa, responsável por sincronizar threads.

Se observou que com mais *threads* havia mais espera, logo mais atraso, que não deveria ser o caso já que o uso de mais processos deveria diminuir a latência. E mesmo com *nproc* = 1 (uma thread em

execução) muito tempo foi perdido na função, comportamento julgado anômalo, já que não há o que sincronizar tendo um único processo. Tudo isso indicaria que há erro na implementação das barreiras.

REFERÊNCIAS

StatQuest: K-means clustering
https://www.youtube.com/watch?v=4b5d3muPQmA
Data Analysis 7: Clustering - Computerphile
https://www.youtube.com/watch?v=KtRLF6rAkyo
Introduction to barriers (pthread_barrier)
https://www.youtube.com/watch?v=_P-HYxHsVPc
Practical example to barriers (pthread_barrier)
https://www.youtube.com/watch?v=MDgVJVIRBnM
k-means clustering
https://en.wikipedia.org/wiki/K-means_clustering
Prática 1 e 2 de perfilamento
https://www.youtube.com/watch?v=pU3Po2t5A-E
https://www.youtube.com/watch?v=Yc9dLECEOLg