

Una implementación de alto rendimiento de agrupación espectral en plataformas CPU-GPU

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

INTRODUCCIÓN

Presentamos una implementación híbrida del algoritmo de agrupamiento espectral que supera significativamente las implementaciones conocidas, la mayoría de las cuales son puramente basadas en CPUs multi-core.

- 1. Uso de la representación dispersa de los gráficos correspondientes y puede manejar extremadamente grandes tamaños de entrada y generar una gran cantidad de agrupaciones.**
- 2. La implementación híbrida es altamente eficiente y es demostrado que hace un buen uso de los recursos disponibles. Nuestros resultados experimentales muestran un rendimiento superior relativa a la implementación de software científico común en CPUs multinúcleo.**

Algoritmo de Cluster

01

Paso 1: Dado un conjunto de puntos de datos $x_1, x_2, \dots, x_n \in \mathbb{R}$ y alguna medida de similitud $s(x_i, x_j)$, construye una matriz de similitud dispersa W que capture similitudes entre los pares de puntos.

02

Paso 2: Calcular el gráfico normalizado de la matriz Laplace como $L_n = D^{-1} L$ donde L es el gráfico no normalizado Matriz laplaciana definida como $L = D - W$ y $P_n D$ es la matriz diagonal con cada elemento $D_{ii} = \sum_j W_{ij}$.

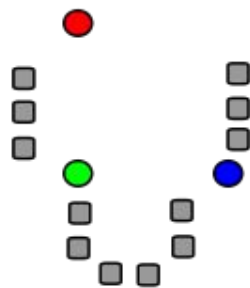
03

Paso 3: Calcular los k vectores propios de la gráfica de la matriz laplaciana L_n correspondiente a los k valores propios distintos de cero.

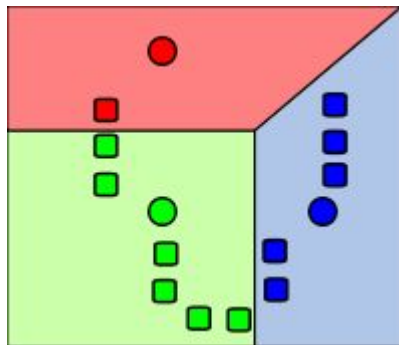
04

Paso 4: Aplicar el algoritmo de agrupamiento de k -means en las filas de la matriz cuyas columnas son las k ; para obtener los clusters finales.

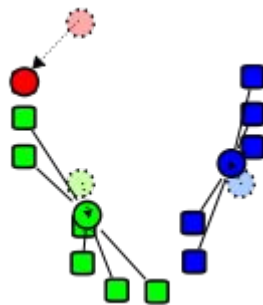
K-MEANS



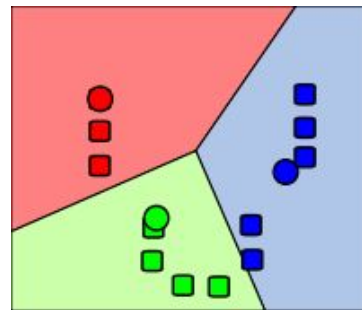
k centroides iniciales (en este caso $k=3$) son generados aleatoriamente dentro de un conjunto de datos (mostrados en color).




k grupos son generados asociándole el punto con la media más cercana. La partición aquí representa el diagrama de Voronoi generado por los centroides.




EL centroide de cada uno de los k grupos se recalcula.



Pasos 2 y 3 se repiten hasta que se logre la convergencia.



```
template<typename T>
int kmeans(int iterations,
           int n, int d, int k,
           thrust::device_vector<T>& data,
           thrust::device_vector<int>& labels,
           thrust::device_vector<T>& centroids,
           thrust::device_vector<T>& distances,
           bool init_from_labels=true,
           double threshold=0.000001) {
    thrust::device_vector<T> data_dots(n);
    thrust::device_vector<T> centroid_dots(n);
    thrust::device_vector<T> pairwise_distances(n * k);
    detail::make_self_dots(n, d, data, data_dots);
    if (init_from_labels) {
        detail::find_centroids(n, d, k, data, labels, centroids);
    }
}
```



```
T prior_distance_sum = 0;

    int i = 0;
    for(; i < iterations; i++) {
        detail::calculate_distances(n, d, k,
                                    data, centroids, data_dots,
                                    centroid_dots, pairwise_distances);

        int changes = detail::relabel(n, k, pairwise_distances, labels, distances);
        detail::find_centroids(n, d, k, data, labels, centroids);
        T distance_sum = thrust::reduce(distances.begin(), distances.end());
        std::cout << "Iteration " << i << " produced " << changes
                    << " changes, and total distance is " << distance_sum << std::endl;

        if (i > 0) {
            T delta = distance_sum / prior_distance_sum;
            if (delta > 1 - threshold) {
                std::cout << "Threshold triggered, terminating iterations early" << std::endl;
                return i + 1;
            }
        }
        prior_distance_sum = distance_sum;
    }
    return i;
```

CONFIGURACIÓN

1. Instalación de Cuda.

Familia de procesador: Intel® Core™ i5-7xxx.

Diagonal de la pantalla: 15.6pulg.

Tarjeta de Video: NVIDIA GeForce GTX 1050

Memoria interna: 8 GB

Capacidad total de almacenaje: 1128 GB

Sistema operativo instalado: Windows 10 Home



1.`sudo dpkg -i cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb`

2.`sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub`

3.`sudo apt-get update`

4.`sudo apt-get install cuda`

5.sudo apt install nvidia-cuda-toolkit

6.nvcc --version



CONFIGURACIÓN

Instalación Arpackpp

git clone <https://github.com/opencollab/arpack-ng>

cd arpack.

Instalación Librerías.

```
$ ./install-openblas.sh
```

```
$ ./install-arpack-ng.sh
```

```
$ ./install-superlu.sh
```

```
$ ./install-suitesparse.sh
```




```
# Defining the machine.
PLAT      = linux
# Defining the compiler.
CPP       = g++
ARPACKPP_DIR = $(HOME)/arpackpp -> Cambiar de direccion de acuerdo a su ubicación
ARPACKPP_INC = $(ARPACKPP_DIR)/include
#SUPERLU_DIR = $(ARPACKPP_INC)
SUPERLU_DIR  = $(ARPACKPP_DIR)/external/SuperLU
UMFPACK_DIR  = $(ARPACKPP_INC)
ARPACK_LIB   = $(ARPACKPP_DIR)/external/libarpack.a
LAPACK_LIB   =
SUPERLU_LIB  = $(ARPACKPP_DIR)/external/libsuperlu.a
BLAS_LIB     = $(ARPACKPP_DIR)/external/libopenblas.a
FORTRAN_LIBS = -lgfortran
SUITESPARSE_DIR = $(ARPACKPP_DIR)/external/SuiteSparse
UMFPACK_LIB    = $(SUITESPARSE_DIR)/UMFPACK/Lib/libumfpack.a \
$(SUITESPARSE_DIR)/CHOLMOD/Lib/libcholmod.a \
$(SUITESPARSE_DIR)/COLAMD/Lib/libcolamd.a \
$(SUITESPARSE_DIR)/CCOLAMD/Lib/libccolamd.a \
$(SUITESPARSE_DIR)/metis-4.0/libmetis.a \
$(SUITESPARSE_DIR)/CAMD/Lib/libcamd.a \
$(SUITESPARSE_DIR)/AMD/Lib/libamd.a \
$(SUITESPARSE_DIR)/SuiteSparse_config/libsuitesparseconfig.a
CHOLMOD_LIB    = $(SUITESPARSE_DIR)/CHOLMOD/Lib/libcholmod.a \
$(SUITESPARSE_DIR)/COLAMD/Lib/libcolamd.a \
```



```
CUDA_CPP = nvcc
```

```
CUDA_ARCH ?= sm_35
```

```
include ../arpackpp/Makefile.inc      -> Configuracion de direccion variable
```

```
CUDA_FLAGS = -arch=$(CUDA_ARCH) -Xptxas -v
```

```
CUDA_LIBS = -lcublas -lcusparse
```

```
spectral_clustering: spectral_clustering.cu timer.o labels.o kmeans.h centroids.h
```

```
    $(CUDA_CPP) $(CPP_FLAGS) $(CUDA_FLAGS) -o spectral_clustering spectral_clustering.cu timer.o labels.o  
$(ALL_LIBS) $(CUDA_LIBS)
```

```
labels.o: labels.cu labels.h
```

```
    $(CUDA_CPP) $(CPP_FLAGS) $(CUDA_FLAGS) -c -o labels.o labels.cu
```

```
timer.o: timer.cu timer.h
```

```
    $(CUDA_CPP) $(CPP_FLAGS) $(CUDA_FLAGS) -c -o timer.o timer.cu
```



CONFIGURACIÓN

Instalación FASTC

git clone <https://github.com/yuj-umd/fastsc>

cd fastsc

\$ make

\$./spectral_clustering input_file_name n k output_file_name

Para finalizar

\$./spectral_clustering input_file_name n k output_file_name



CONCLUSIONES

Combinando las interfaces del software tradicional basado en CPU.

Paquetes ARPACK y biblioteca CUDA basada en GPU.

Se logra una combinación logra buenos incrementos de velocidad en comparación con otros Software basado en CPU.

Implementamos una estrategia de siembra inteligente y utilizar las operaciones BLAS para implementar el rápido k-means de agrupamiento. Nuestra implementación se muestra a lograr una aceleración significativa en comparación con otros paquetes de software, especialmente para problemas a gran escala.

¡Gracias!

