



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Da un secolo, oltre.



HR EXCELLENCE IN RESEARCH

# Progetto Parallel Programming

JobLib – OpenMP – OpenACC

**Elia Matteini - Filippo Zaccari**



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Da un secolo, oltre.



HR EXCELLENCE IN RESEARCH

# Image Augmentation

## Python - JobLib



# Introduzione

- Image augmentation è una tecnica utilizzata per ampliare i dataset di immagini
- Libreria Albumentation per modificare le immagini
- Libreria JobLib per la parallelizzazione
- Libreria OpenCV per lettura/scriittura immagini da/su disco

# Struttura sequenziale e parallela

## Sequenziale

Lettura immagini da disco

Definizione set di trasformazioni

Per ogni immagine:

    Applicazione trasformazioni

    Salvataggio immagini aumentate

## Parallelo

Divisione delle immagini in batch in base al numero di processi

Per ogni batch di immagini:

    Lettura immagini da disco

    Definizione set di trasformazioni

    Applicazione trasformazioni

    Salvataggio immagini aumentate

Divisione delle  
immagini in batch in  
base al numero  
di processi

Parallelizzazione

```
if __name__ == '__main__':
    num_augmentations = int(sys.argv[1])
    num_process = int(sys.argv[2])

    # Cleaning workspace from old executions
    try:
        print("Cleaning workspace...")
        shutil.rmtree(folder_out)
        shutil.os.mkdir(folder_out)
        print("Done")
    except OSError as e:
        print(f"ERROR: {folder_out} - {e.strerror}")
        shutil.os.mkdir(folder_out)
        print(folder_out + " created!")

    try:
        images_from_folder = os.listdir(folder_in)
    except OSError as e:
        print(f"ERROR: {folder_in} - {e.strerror}")
        print("ERROR: import jpg dataset images into " + folder_in)
        sys.exit(1)

    imageBatchSize = math.ceil(len(images_from_folder) / num_process)
    batches_for_process = [images_from_folder[i:i + imageBatchSize] for i in range(0, len(images_from_folder), imageBatchSize)]
    print(f'batches_for_process {batches_for_process}')

    Parallel(n_jobs=num_process)(delayed(imgAugmentation)(batch, num_augmentations) for batch in batches_for_process)
    print("Done")
```

Disattivazione  
parallelizzazione nella  
funzione di OpenCV

```
def imgAugmentation(imagesPath, num_augmentations):  
    cv2.setNumThreads(0)  
    global folder_in  
    global folder_out  
    images = []  
    for path in imagesPath:  
        images.append(cv2.imread(folder_in + path))
```

Lettura immagini per ogni  
batch

Definizione del set di  
trasformazioni

```
transform = A.Compose([
    A.HorizontalFlip(p=1),
    A.VerticalFlip(p=1),
    A.Rotate(limit=360, p=1),
    A.RandomBrightnessContrast(p=1),
    A.GaussianBlur(blur_limit=(3, 9), p=1),
    A.ColorJitter(brightness=0.6, contrast=0.6, saturation=0.6, hue=0.6, p=1),
    A.RGBShift(r_shift_limit=40, g_shift_limit=40, b_shift_limit=40, p=1),
    A.ChannelShuffle(p=1),
    A.RandomGamma(p=1),
    A.Blur(p=1),
    A.ToGray(p=1),
])
```

Per ogni immagine nel  
batch, applica le  
trasformazioni e scrive  
le immagini su disco

```
print(f'Augmenting images by thread number: {os.getpid()}')
j = 0
for image in images:
    i = 0
    for i in range(num_augmentations):
        # Augmenting image
        augmented_image = transform(image=image)['image']
        # Writing augmented image on disk
        cv2.imwrite('./' + folder_out + '/augmented_image' + str(j) + '_' + str(i) + str(os.getpid()) + '.jpg', augmented_image)
    j = j + 1
```

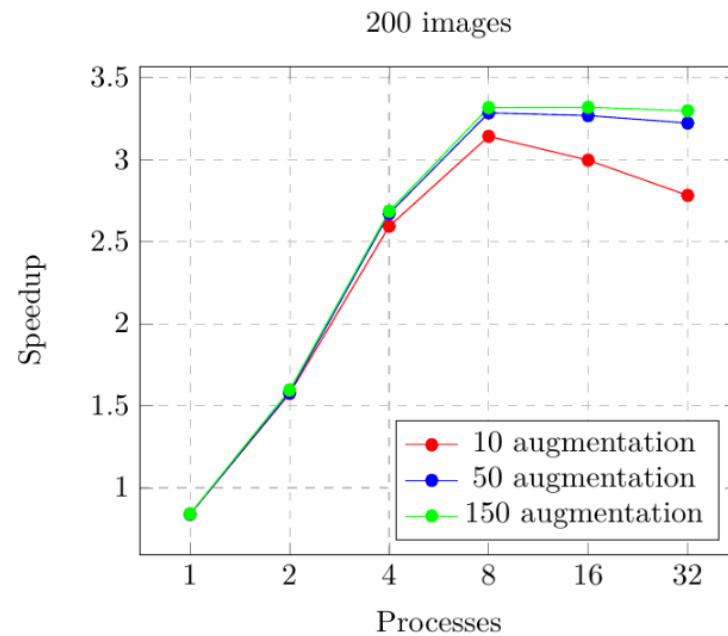
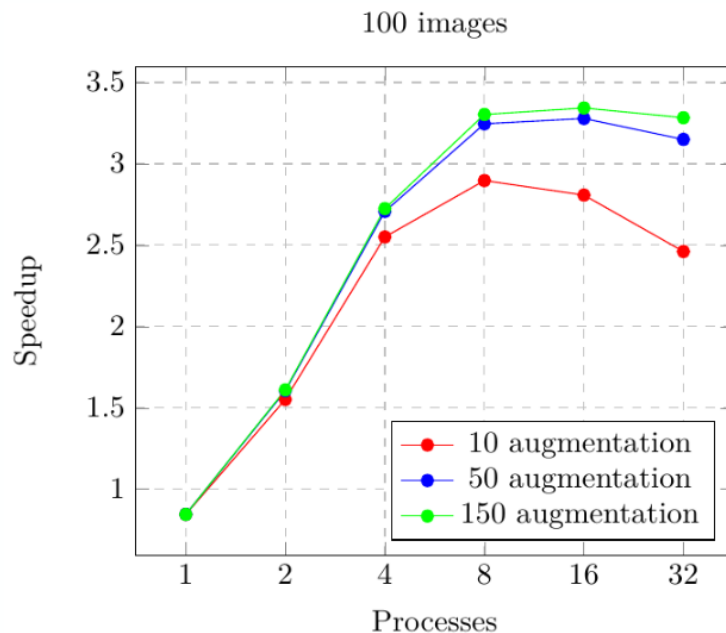


# Specifiche

- Laptop System: ASUSTeK
- product: TUF Gaming FX505DV
- Distro: Ubuntu 22.04.4 LTS
- Kernel: 6.5.0-21-generic x86 64
- CPU: quad core (8 thread) AMD Ryzen 7
- 3750H 2.3 Ghz
- Memory: 16 GB



# Speedup





# Conclusioni

## Image Augmentation

- Speedup positivo nonostante l'utilizzo del multiprocessing invece del multithreading
- Scrittura delle immagini su disco ad ogni trasformazione per evitare la saturazione della RAM



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Da un secolo, oltre.



HR EXCELLENCE IN RESEARCH

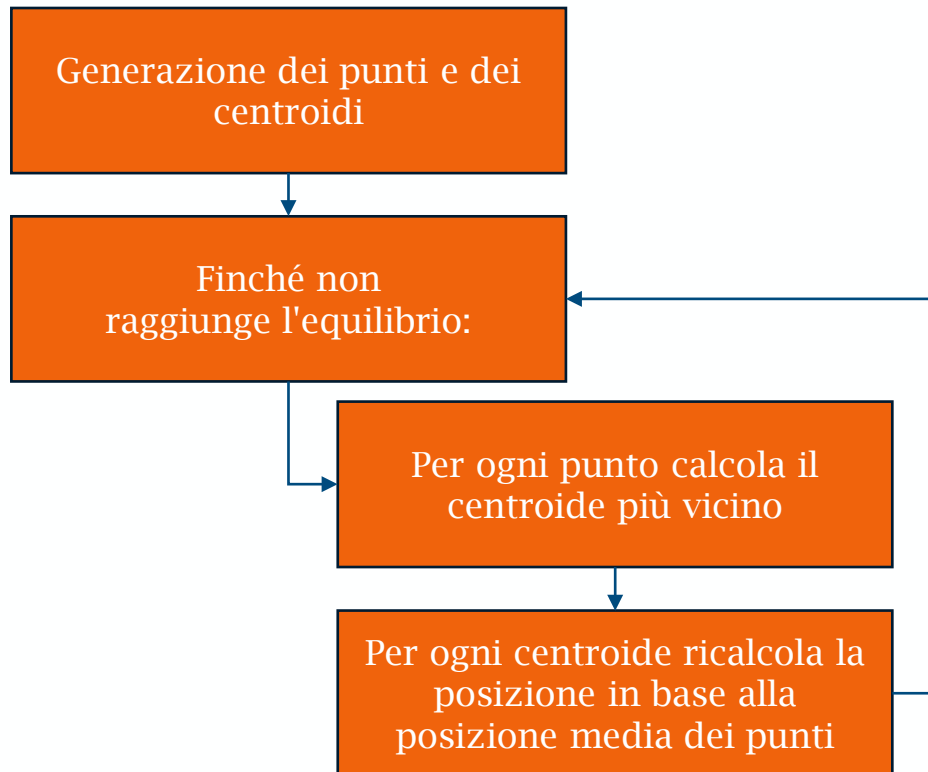
# K-Means

## C++ - OpenMP

## Introduzione

- K-means è una tecnica di machine learning unsupervised per raggruppare un insieme di elementi omogenei intorno a K cluster
- Direttive (pragma) OpenMP per la parallelizzazione
- Utilizzo struttura dati AoS

## Struttura programma



Generazione dei punti

```
points.reserve(numPoints);  
for(int i = 0; i < numPoints; i++) {  
    points.emplace_back(rand() % 6000, rand() % 6000);  
}
```

Generazione dei centroidi

```
centroids.reserve(centroidsNumber);  
for (int i = 0; i < centroidsNumber; i++) {  
    centroids.emplace_back(rand() % 6000, rand() % 6000);  
}
```

Calcolo del centroide  
più vicino

Salvataggio valori nuovi  
assegnamenti dei punti

Ricalcolo della  
posizione dei centroidi

```
#ifdef PARALLEL
#pragma omp parallel for reduction(+:sumXYCount[:3*centroidsNumber])
#endif

for (int i = 0; i < numPoints; i++) {
    float minDistance = __DBL_MAX__; // Reset of minDistance for each iteration
    int cluster = -1;
    for (int j = 0; j < centroidsNumber; j++) {
        int d = pow(points[i].x - centroids[j].x, 2) + pow(points[i].y - centroids[j].y, 2);
        if (d < minDistance) {
            minDistance = d;
            cluster = j;
        }
    }
    if (cluster != points[i].cluster) {
        points[i].cluster = cluster;
        changed = true;
    }
    sumXYCount[3 * cluster] += points[i].x;
    sumXYCount[3 * cluster + 1] += points[i].y;
    sumXYCount[3 * cluster + 2]++;
}

if (changed) {
    /// 3) Redefine the cluster

#ifdef PARALLEL
#pragma omp parallel for
#endif
    for (int j = 0; j < centroidsNumber; j++) {
        centroids[j].x = sumXYCount[3 * j] / sumXYCount[3 * j + 2];
        centroids[j].y = sumXYCount[3 * j + 1] / sumXYCount[3 * j + 2];
    }
}
```

Parallelizzazione

Parallelizzazione

## Specifiche

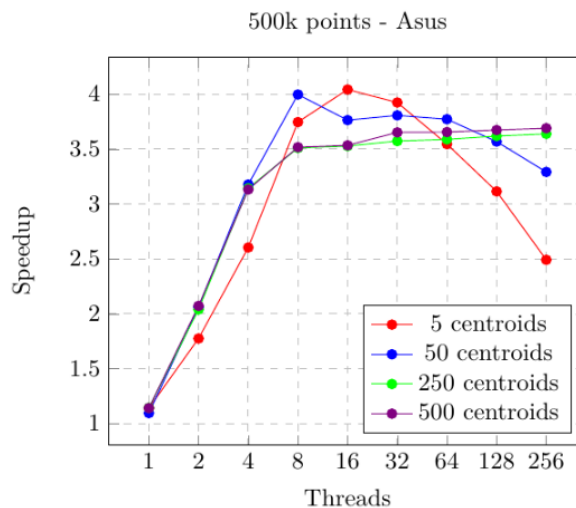
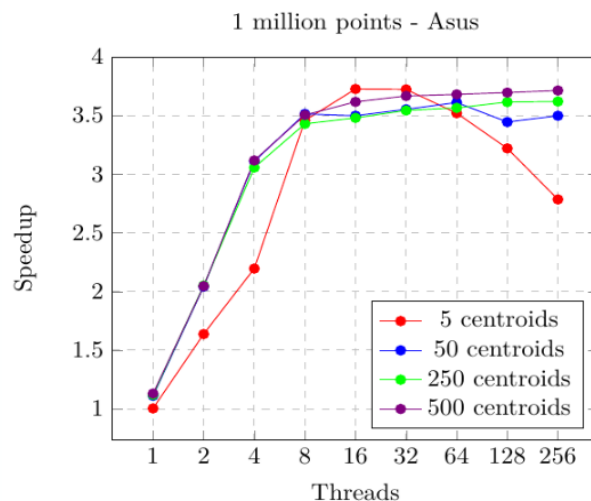
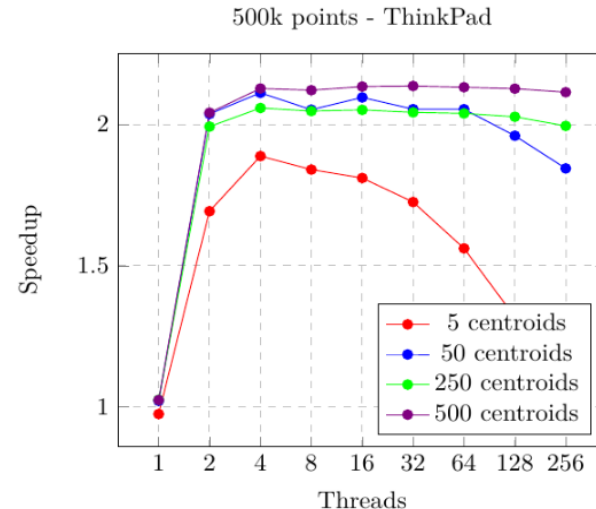
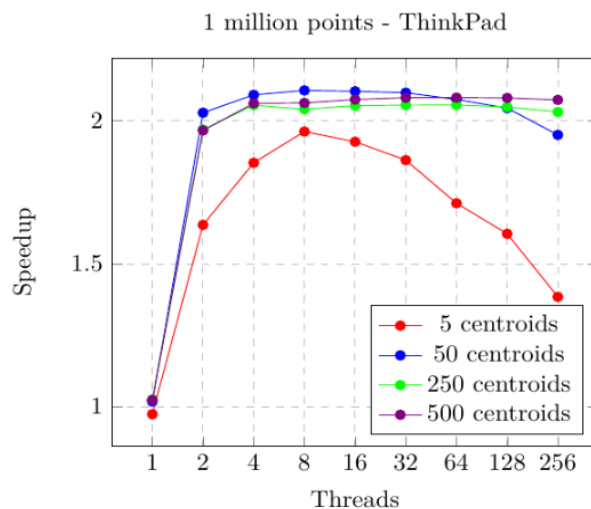
- Laptop System: ASUSTeK
- product: TUF Gaming FX505DV
- Distro: Ubuntu 22.04.4 LTS
- Kernel: 6.5.0-21-generic x86 64
- CPU: quad core (8 thread) AMD Ryzen 7
- 3750H 2.3 Ghz
- Memory: 16 GB

- Laptop System: LENOVO ThinkPad X280
- Distro: Debian GNU/Linux trixie/sid
- Kernel: 6.6.15-amd64
- CPU: dual core (4 threads) model: Intel Core
- i5-7300U 64 bits 2.6 Ghz
- Memory: 8 GB





# Speedup





# Conclusioni

## K-Means - OpenMP

- Programmazione ad oggetti non ideale per questo approccio
- Prestazioni migliorate rispetto alla versione sequenziale



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Da un secolo, oltre.



HR EXCELLENCE IN RESEARCH

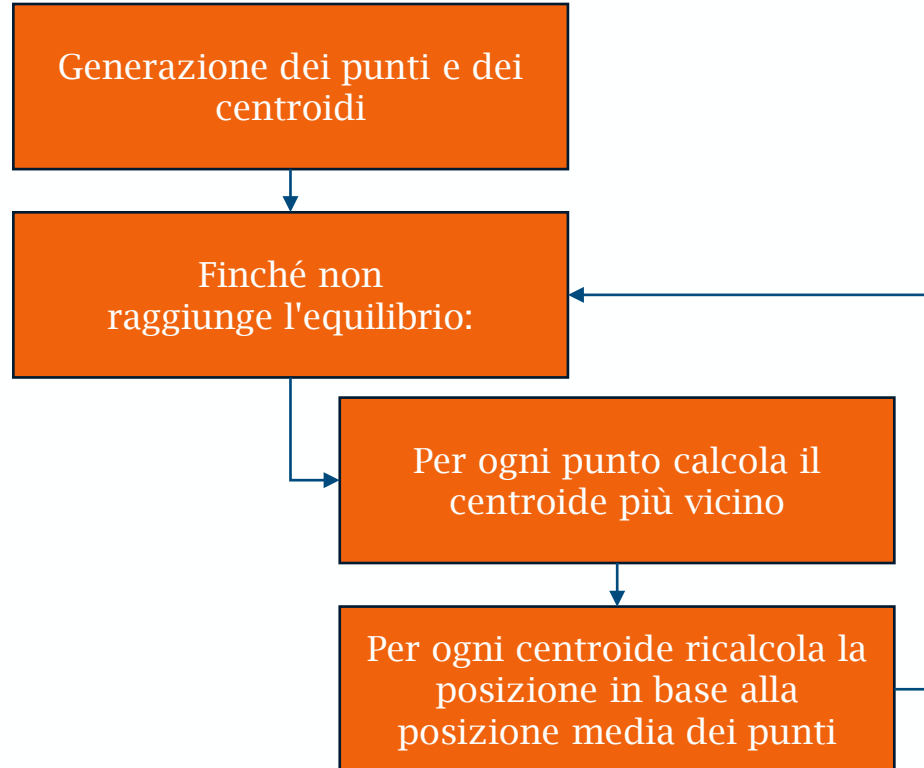
# K-Means

## C - OpenACC

# Introduzione

- K-means è una tecnica di machine learning unsupervised per raggruppare un insieme di elementi omogenei intorno a K cluster
- Direttive (pragma) OpenACC per la parallelizzazione
- Utilizzo struttura dati SoA


## Struttura programma





Generazione dei punti

```
Points points;
points.x = (float *)malloc(numPoints * sizeof(float));
points.y = (float *)malloc(numPoints * sizeof(float));
points.clusters = (int *)malloc(numPoints * sizeof(int));
for(int i = 0; i < numPoints; i++) {
    points.x[i] = rand() % 6000;
    points.y[i] = rand() % 6000;
    points.clusters[i] = -1;
}
```



Generazione dei centroidi

```
Centroids centroids;
centroids.x = (float *)malloc(centroidsNumber * sizeof(float));
centroids.y = (float *)malloc(centroidsNumber * sizeof(float));
for (int i = 0; i < centroidsNumber; i++) {
    centroids.x[i] = rand() % 6000;
    centroids.y[i] = rand() % 6000;
}
```

Trasferimento dati da host a  
device

```
#pragma acc data copyin(points.x[:numPoints], points.y[:numPoints], centroidsNumber, numPoints,  
points.clusters[:numPoints], centroids.x[:centroidsNumber], centroids.y[:centroidsNumber], changed)  
create(sumXYCount[:centroidsNumber*3])  
{  
    do {  
        changed = 0;  
  
        for (int i = 0; i < 3 * centroidsNumber; i++) {  
            sumXYCount[i] = 0;  
        }  
    }
```

Parallelizzazione

Sincronizzazione  
variabile "changed"

```
#pragma acc parallel loop gang worker vector reduction(|:changed) reduction(+:sumXYCount[:3*centroidsNumber])
present(points.x[:numPoints], points.y[:numPoints], centroids.x[:centroidsNumber],
centroids.y[:centroidsNumber], points.clusters[:numPoints], sumXYCount[:3*centroidsNumber], changed)
for (int i = 0; i < numPoints; i++) {
    double minDistance = DBL_MAX; // Reset of minDistance for each iteration
    int cluster = -1;

    for (int j = 0; j < centroidsNumber; j++) {
        int d = pow(points.x[i] - centroids.x[j], 2) + pow(points.y[i] - centroids.y[j], 2);
        if (d < minDistance) {
            minDistance = d;
            cluster = j;
        }
    }
    if (cluster != points.clusters[i]) {
        points.clusters[i] = cluster;
        changed = 1;
    }
    sumXYCount[3 * cluster] += points.x[i];
    sumXYCount[3 * cluster + 1] += points.y[i];
    sumXYCount[3 * cluster + 2]++;
}

#pragma acc update self(changed)
}

if (changed) {
    #pragma acc parallel loop gang worker vector firstprivate(sumXYCount)
    for (int j = 0; j < centroidsNumber; j++) {
        centroids.x[j] = sumXYCount[3 * j] / sumXYCount[3 * j + 2];
        centroids.y[j] = sumXYCount[3 * j + 1] / sumXYCount[3 * j + 2];
    }
}
```



Trasferimento dati da device  
a host

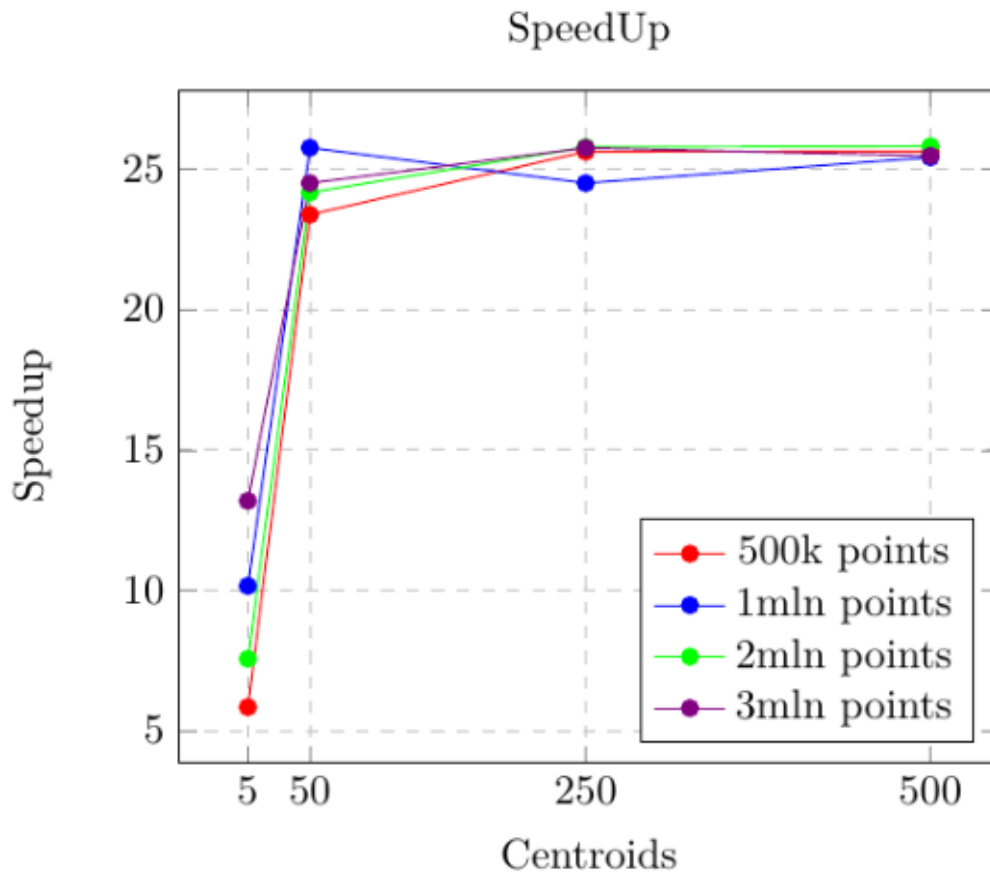
```
    } while (changed);  
#pragma acc data copyout(points.clusters[0:numPoints],  
    centroids.x[0:centroidsNumber], centroids.y[0:centroidsNumber])  
}  
  
free(points.x);  
free(points.y);  
free(points.clusters);  
free(centroids.x);  
free(centroids.y);
```

Liberazione memoria

# Specifiche

- GPU: NVIDIA GeForce RTX 2060
- Cores: 1920
- Memory: 6GB GDDR6
- Architecture: Turing

# Speedup





# Conclusioni

## K-Means - OpenACC

- Linguaggio basso livello
- Trasferimento oneroso dei dati da host a device
- Problemi sincronizzazione dati
- Prestazioni nettamente migliori rispetto alla versione con CPU



# Considerazioni

## OpenMP - OpenACC

- Comodità dei pragma
- Maggior complessità in OpenACC
- Maggior velocità in OpenACC
- Compilazione più difficoltosa su OpenACC