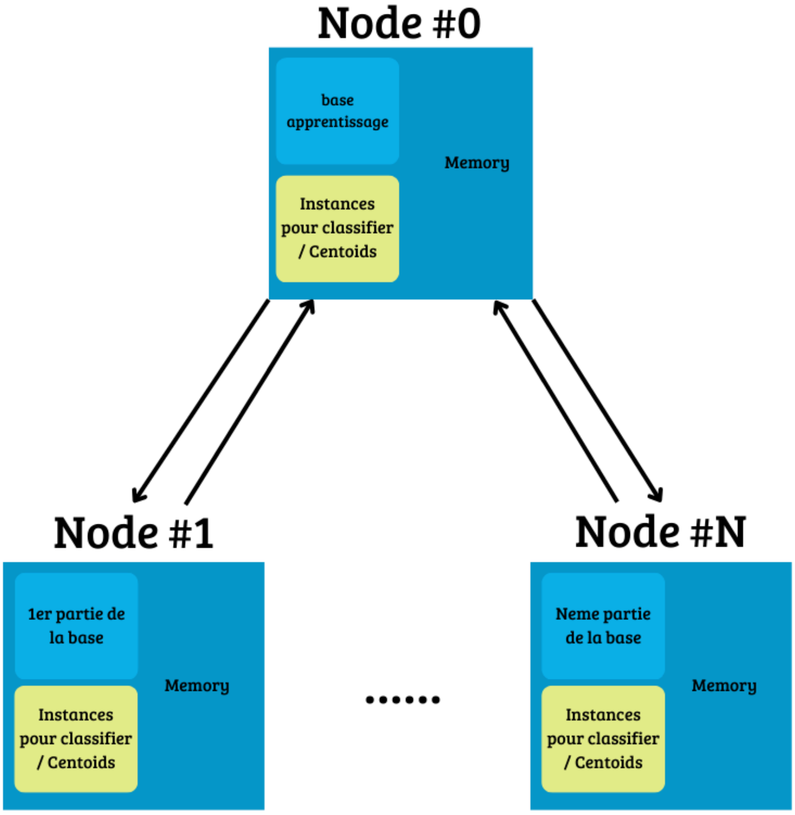
KHENE SORAYA
202031075992
M1 HPC

RAPPORT PROJET TP
(Calcule Parallel)

1 - Parallel Scheme :


Node #0
base apprentissage
Instances pour classifier / Centoids
Memory
Node #1
1er partie de la base
Instances pour classifier / Centoids
Memory
......
Node #N
Neme partie de la base
Instances pour classifier / Centoids
Memory

- The diagram represents k nodes, one master (Node #0), the others slaves
- The master node owns the entire database as well as the instances/centroids
- The slaves receive the data (for the database only a part) from the master, they perform the KNN/K-Means calculations and send the results back to master for results display
- The slaves can have possible communication between them, example : the case of MPI_Allreduce

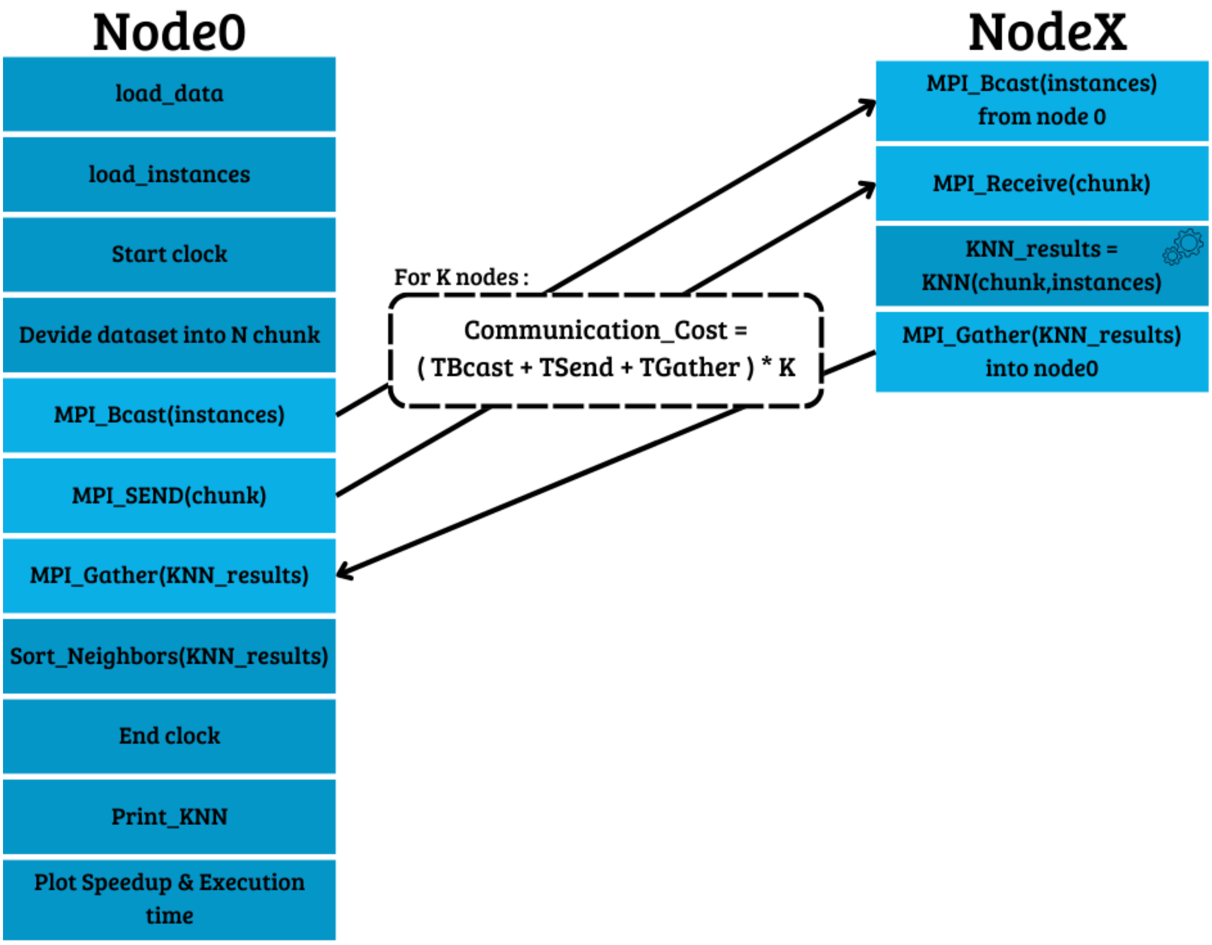
Information of my machine :

CPU : Processor Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 2112 Mhz, 4 Core(s), 8 Logical Processor(s)
Memory : Installed Physical Memory (RAM) 16.0 GB

2 - MPI :

- MPI ⇒ Distributed memory
- Communication cost due to data exchange between processes
- When process 0 sends data, the process receiver will create a copy of that data in his own memory space, which it can modify independently without affecting the data in other processes.

KNN :


Node0
load_data
load_instances
Start clock
Devide dataset into N chunk
MPI_Bcast(instances)
MPI_SEND(chunk)
MPI_Gather(KNN_results)
Sort_Neighbors(KNN_results)
End clock
Print_KNN
Plot Speedup & Execution time
For K nodes :
Communication_Cost =
(TBcast + TSend + TGather) * K
NodeX
MPI_Bcast(instances) from node 0
MPI_Receive(chunk)
KNN_results = KNN(chunk,instances)
MPI_Gather(KNN_results) into node0

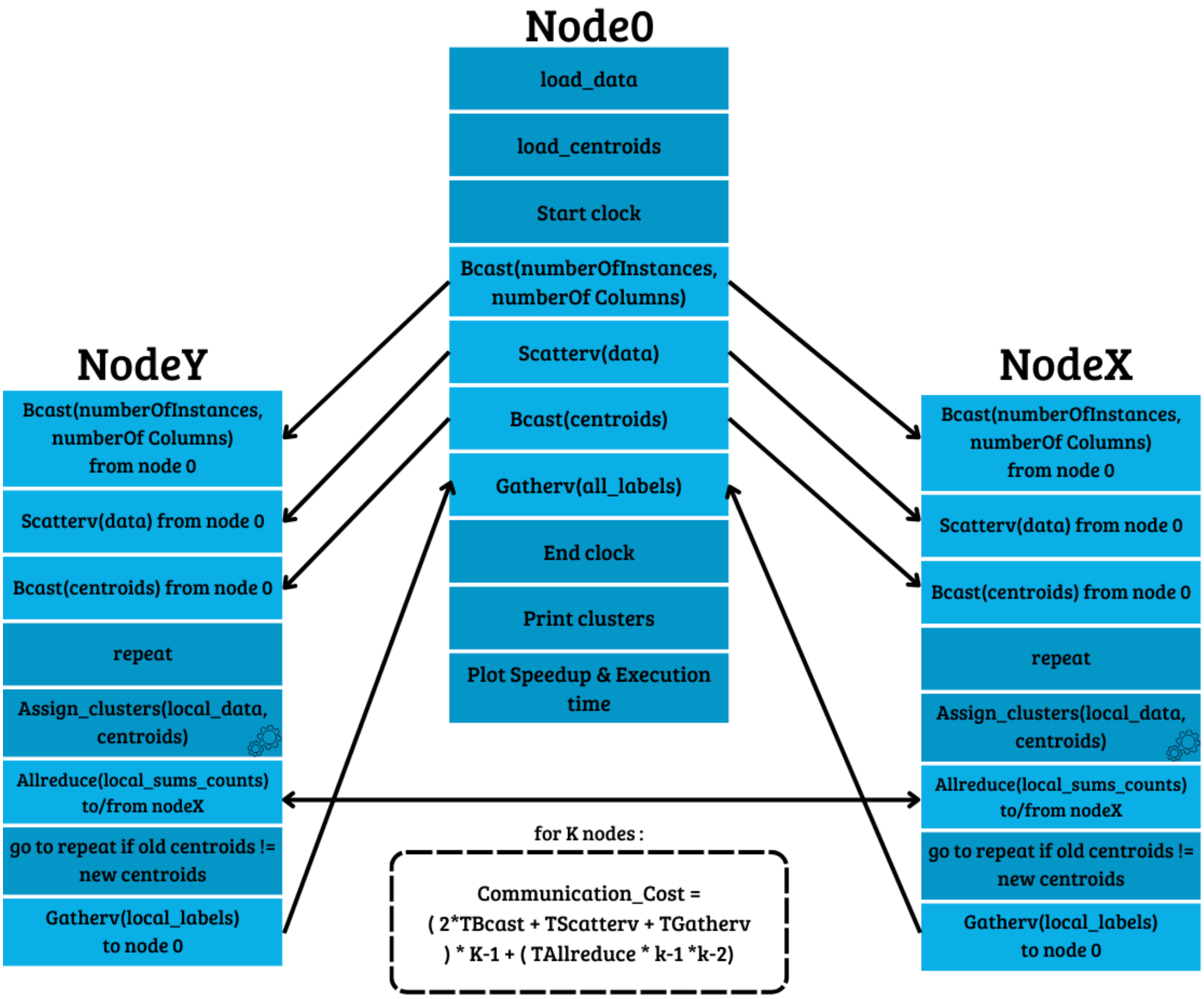
MPI_Bcast	MPI_Send	MPI_Receive	MPI_Gather
Is used to distribute a message (data) from one process (node 0) to all other processes, each receiver will have a copy of the message in his own memory space ⇒ collective communication	Is used to send a message from one process to another ⇒ point-to-point communication	Is used to receive a message sent by another process	Is used to collect data from all processes and gather it into a single process.

Variables description :

Let’s consider a cluster of N nodes :

	Description	**Size**
Instances	Instances to classify	W*1 = nbr of attributes (in my case I classify one instance at a time)
chunk	part of the dataset	L/N*(W+1) L : nbr of instances in the dataset W : nbr of attributes + class label
knn_results	each process writes the k nearest neighbors that he found	K*size size = nbr of processes

K-Means :


Node0
load_data
load_centroids
Start clock
Bcast(numberOfInstances, numberOf Columns)
Scatterv(data)
Bcast(centroids)
Gatherv(all_labels)
End clock
Print clusters
Plot Speedup & Execution time
NodeY
Bcast(numberOfInstances, numberOf Columns) from node 0
Scatterv(data) from node 0
Bcast(centroids) from node 0
repeat
Assign_clusters(local_data, centroids)
Allreduce(local_sums_counts) to/from nodeX
go to repeat if old centroids != new centroids
Gatherv(local_labels) to node 0
NodeX
Bcast(numberOfInstances, numberOf Columns) from node 0
Scatterv(data) from node 0
Bcast(centroids) from node 0
repeat
Assign_clusters(local_data, centroids)
Allreduce(local_sums_counts) to/from nodeX
go to repeat if old centroids != new centroids
Gatherv(local_labels) to node 0
for K nodes :
Communication_Cost =
(2*TBcast + TScatterv + TGatherv
) * K-1 + (TAllreduce * k-1 *k-2)

MPI_Bcast	MPI_Scatterv	MPI_Allreduce	MPI_Gatherv
Same	Is used to send data from the root process to all processes. It's useful when sending data with varying lengths.	Is useful when performing a reduction operation (ex SUM) across all processes and distributing the result back to all processes.	Seme as gather + data with potentially different lengths

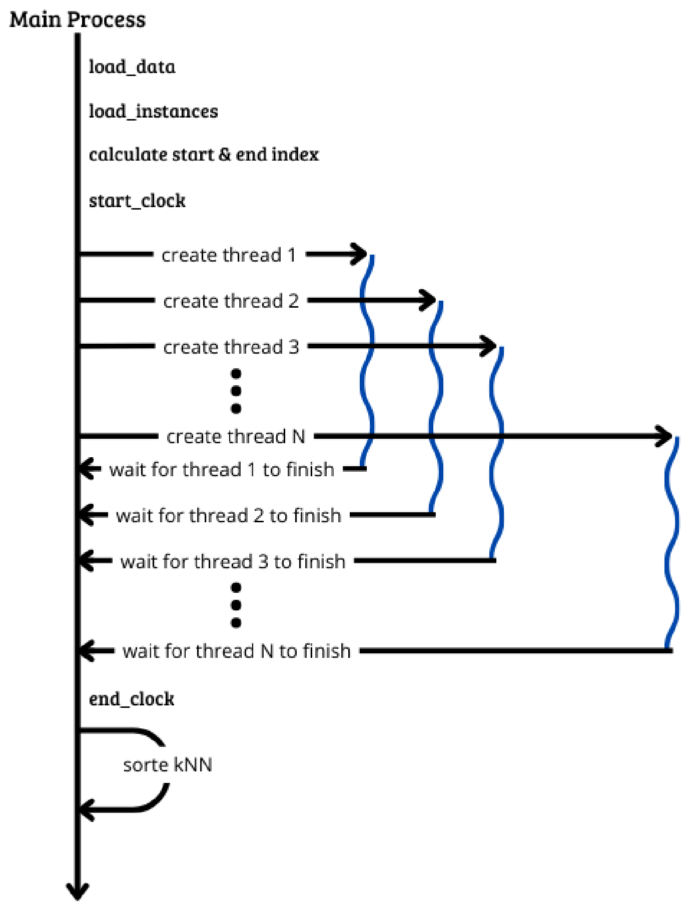
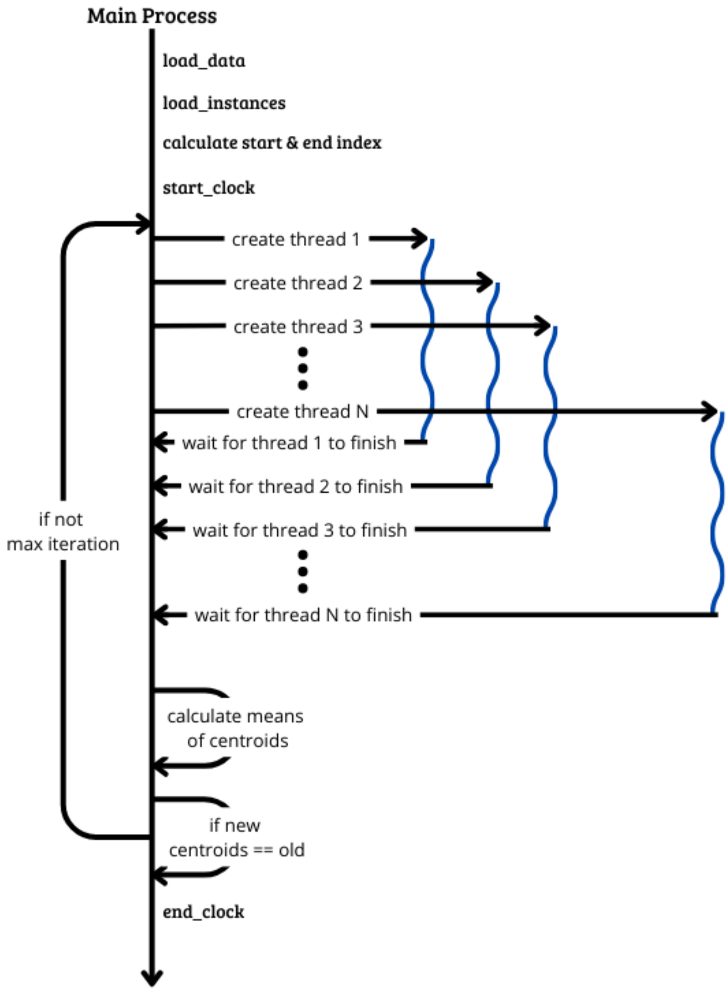
Variables description :

Let's consider a cluster of N nodes :

	Description	**Size**
data	Dataset	L*W L : nbr of instances W : nbr of attributes
centroids	Gravity centers (C)	C*(W+1) +1 because of class label
local_data	Part of data	~L/N*W
local_sums_counts	**local sums** : Each entry in the matrix, local_sums[c][w], represents the sum of the *w*-th attribute of all data points currently assigned to the c-th cluster.	C*W
	local counts : how many instances are in each cluster	C
local_labels	The ith entry represent the class of the ith instance	L/N
all_labels	local_labels all gathered	L

3 - Threads :

- Shared memory
- No communication cost

KNN	KMEANS
 Main Process load_data load_instances calculate start & end index start_clock create thread 1 create thread 2 create thread 3 create thread N wait for thread 1 to finish wait for thread 2 to finish wait for thread 3 to finish wait for thread N to finish end_clock sorte kNN	 Main Process load_data load_instances calculate start & end index start_clock create thread 1 create thread 2 create thread 3 create thread N wait for thread 1 to finish wait for thread 2 to finish wait for thread 3 to finish wait for thread N to finish if not max iteration calculate means of centroids if new centroids == old end_clock

Data passed for each thread :

KNN	K-means
Dataset	Dataset
Starting index	Starting index
Ending index	Ending index
Instances to classify	Centroids
Array to write KNN results (write k nearest neighbors only)	Array to write k-means results (labels)

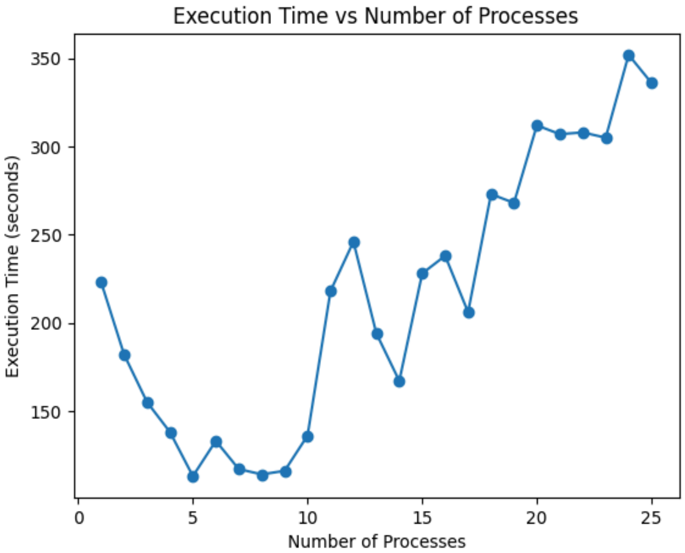
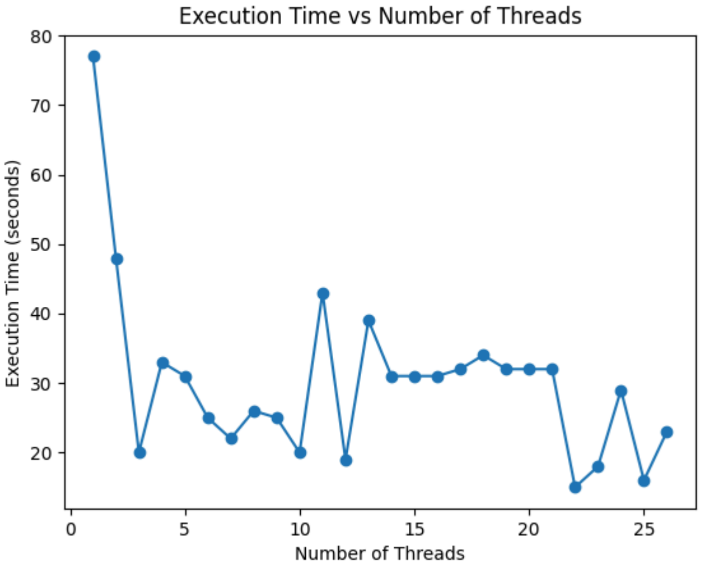
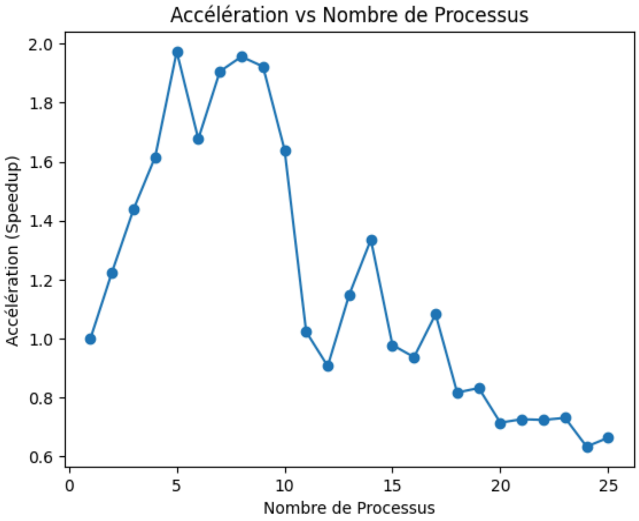
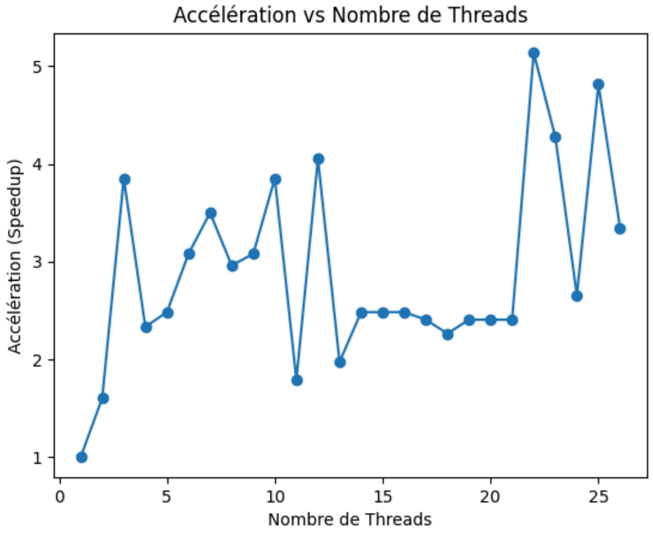
- Each thread executes knn/ k-means and write results in knn/k-means results

4 - Curves :

I’ve worked with “covertype” dataset :

Description :
Classification into 7 forest cover types based on attributes such as elevation, aspect, slope, hillshade, soil-type, and more.
Number of Instances : 581011 (+58k)
Number of Attributes : 54 + 1 class label
7 types of class labels :
Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, Krummholz

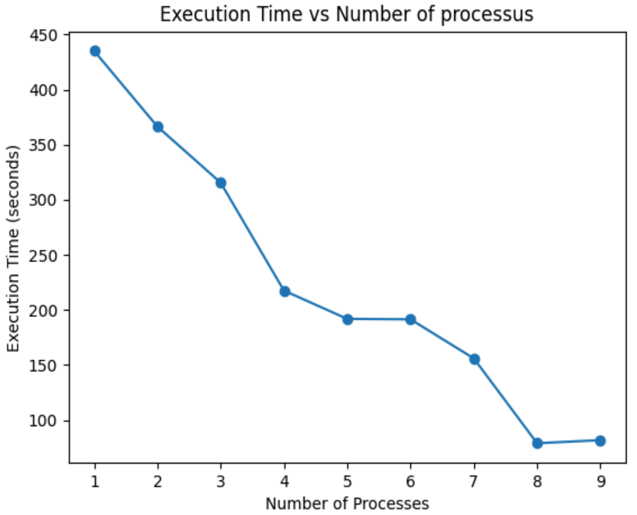
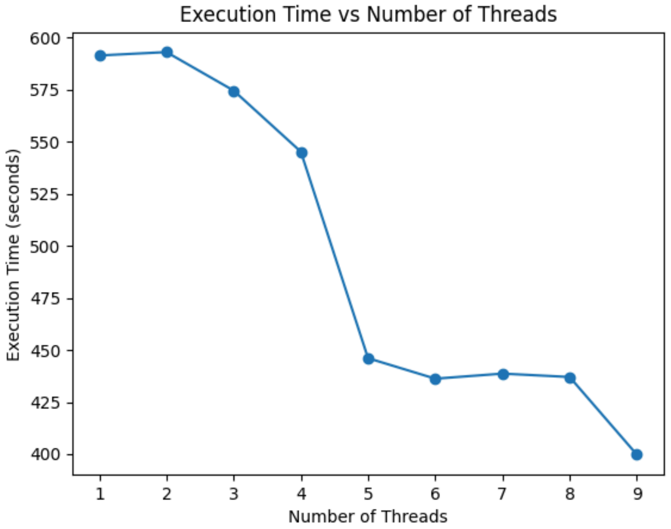
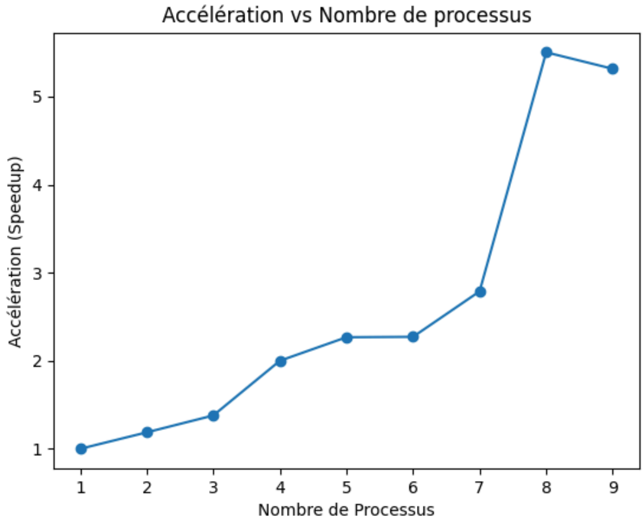
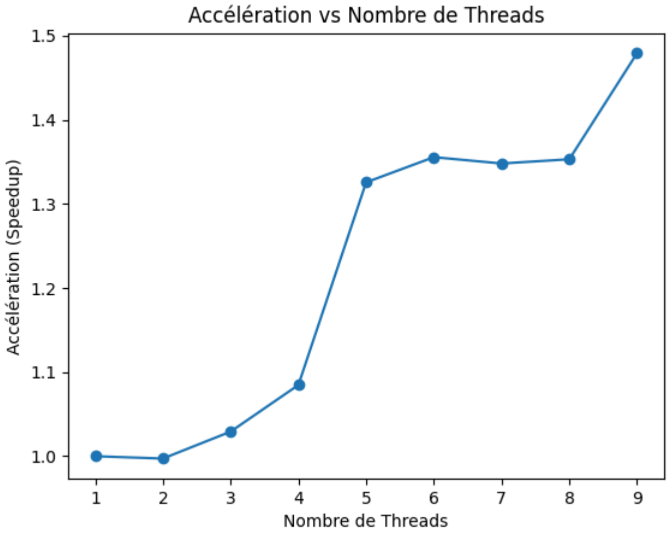
KNN :

MPI	Threads
	
	

	Peak
MPI	5 processes
THREADS	22 processes

Threads vs Processes : Threads show better results than MPI in term of execution time and speedup

K-Means :

MPI	**Threads**
 	
 	

Threads vs Processes : MPI show better results than THREADS in term of execution time and speedup