A hybrid clustering algorithm for high-performance edge computing devices (short communication)

Giuliano Laccetti[1], Marco Lapegna[1], Diego Romano[2]

ISPDC 2022
21st IEEE International Symposium on Parallel and Distributed Computing

1) depart. Mathematics and Applications – Univ. of Naples Federico II (Italy)
2) Inst. High Performance Computing and Networking – Nat. Research Council (Italy)

the rising of Edge Computing

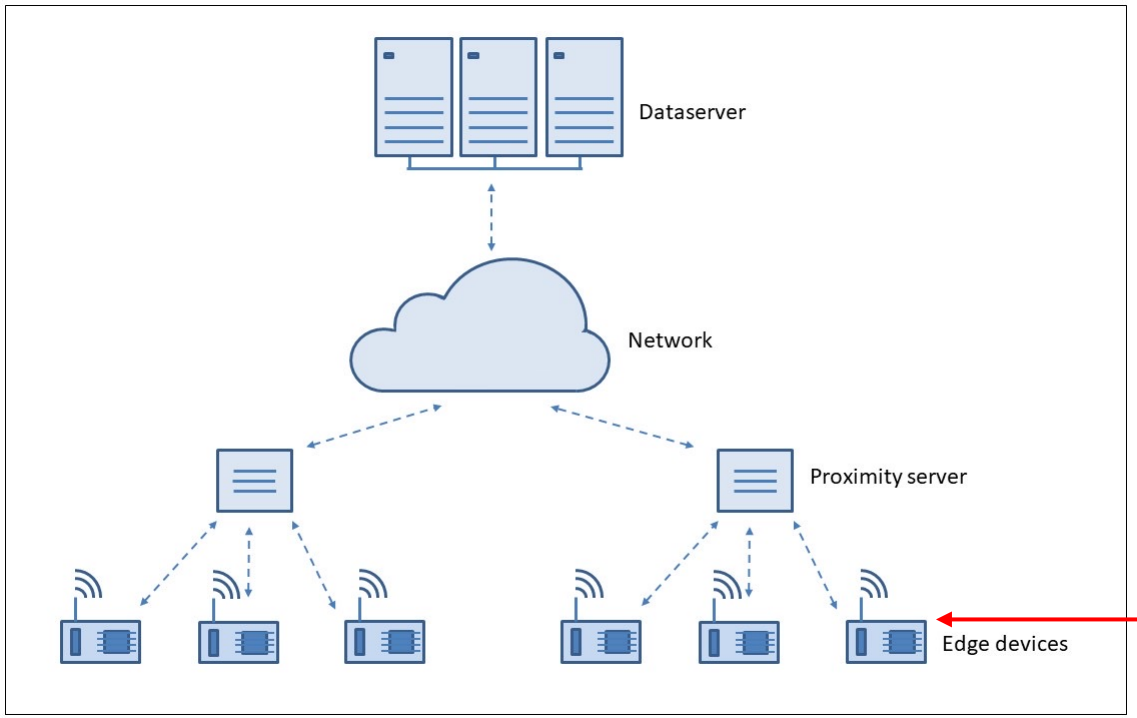
The rising of the Edge Computing (EC) paradigm is motivated by the need to move decision points in real-time applications to the edge of the Internet of Things (IoT) network, so to address the typical issues of centralized data management, where the transferring of large volumes of data can rapidly saturate the network bandwidth.

Further benefits of the EC paradigm are related to

- better fault-tolerant : a single failed device can be easily isolated from the network without interrupting the overall service and replaced at a low cost
- wider open access to knowledge : through the consortia of several companies, a distributed data collection model allows greater democratization
- privacy and security improvement : it is possible to significantly reduce sensitive information from spreading on the network

EC infrastructure

- **edge devices** implement real-time applications processing data collected in the same place. On the other hand, they are featured by small-sized memory and CPU with reduced computing power
- **proximity servers** support the computing boards at the edge with critical tasks such as the workload balancing between devices, intrusion detection, reliability management, up to data caching.
- **Dataservers** exhibit significant latency and are not generally valuable in real-time choices at the edge. In any case, they can support the application with supplementary computing resources and build global models



an example:
Nvidia Jetson nano

- CPU Quad-core ARM Cortex-A57
- GPU Nvidia Maxwell 128 core
- 4 GB RAM LPDD4
- SSD eMCC 16 GB
- 5 / 10 Watt (TDP)
- 69,6 mm x 45 mm

High performance
and
low power

clustering algorithms and EC environments

Artificial Intelligence (AI) and Machine Learning (ML) are among the application areas of

most significant interest for edge computing

- **AI/ML for edge computing**, aimed to improve the efficiency and effectiveness of the infrastructure through AI/ML techniques. For example, it is possible to classify the Edge Devices considering their trustworthiness, in order to improve security
- **AI/ML on the edge**, aimed to run AI/ML models on the edge of the network. For example, it is possible to implement AI model ''at the edge'', so to shift intelligence and decision-making ability for time-critical applications to the edge of the network

Among the most used ML methodologies, **clustering algorithms** play a central role in gathering similar data in the same group according to a precise metric.

Many surveys emphasize the computational peculiarities of the **K-means algorithm**, featured by simplicity, ability to scale with the number of attributes and elements, and a reasonable computational cost mainly in advanced computing systems.

the K-means algorithm

Given

- an integer K
- a set $S = \{ x_n \in R^d , n = 1,..,N\}$ of N vectors in the d-dimensional real space

the K-means algorithms is a clustering algorithm aimed to **collect the items of S in K subset** (called clusters) of a partition $P_K = \{C_k \subset S, k = 1,..,K \}$ **on the basis of their similarity**

the traditional description of the K-mean algorithm is

1. subdivide the N items in K arbitrary clusters, each of them with N_k $(k = 1, \dots, K)$ items
2. compute the center c_k of the clusters with the vector operation
3. for each x_n find the cluster $C_{\bar{k}}$ that minimize the euclidian distance from its center
4. reassign each x_n to the new cluster $C_{\bar{k}}$
5. repeat steps 2 - 4 until there is no change

main problems of the K-means algorithm

- The value of K is an input data, and it must be fixed before the execution (not true for several applications)
  - K too small : dissimilar items can be grouped in the same cluster
  - K too large : similar items can be assigned to different clusters

possible solution

execute the K-mean algorithm several times with increasing values of K

To reduce the computational cost, our K-means algorithm increments the number of clusters, working only on the clusters with the more dissimilar elements (e.g., measured through the standard deviation)

More precisely:

- $K = 1 \quad P_1 = \{C_1\}$ where $C_1 = S$
- $K > 1 \quad P_K = P_{K-1} - \{\widehat{C_{K-1}}\} \cup \{C_\alpha, C_\beta\}$

split in two subset C_α and C_β only the cluster $\widehat{C_{K-1}}$ with the largest standard deviation in the previous iteration

Adaptive K-means algorithm

Algorihtm 1

Algorithm 1: adaptive K-means algorithm

implementation of the adaptive strategy

kernel of the algorithm

1) Set the number of clusters $K = 0$
2) **repeat**:
   2.1) Increase the number of clusters $K = K + 1$
   2.2) find the cluster C_γ with the largest SD
   2.3) define the new partition of clusters $\mathcal{P}_K$
   2.4) **repeat**:
      2.4.1) **for** each cluster $C_k \in \mathcal{P}$:
         update clusters info
      2.4.2) **for** each $\mathbf{x}_n \in S$:
         search the cluster of belonging C_δ
      2.4.3) **for** each $\mathbf{x}_n \in S$:
         displace $\mathbf{x}_n$ to C_δ
      until (no change in the reassignment)
   2.5) update some quality index
until (the variation the quality index is negligible

where is the parallelism ?

where is the parallelism

Algorithm 1: adaptive K-means algorithm

1) Set the number of clusters $K = 0$
2) **repeat**:
   2.1) Increase the number of clusters $K = K + 1$
   2.2) find the cluster C_γ with the largest SD
   2.3) define the new partition of clusters $\mathcal{P}_K$
   2.4) **repeat**:
      2.4.1) **for** each cluster $C_k \in \mathcal{P}$:
         update clusters info
      2.4.2) **for** each $\mathbf{x}_n \in S$:
         search the cluster of belonging C_δ
      2.4.3) **for** each $\mathbf{x}_n \in S$:
         displace $\mathbf{x}_n$ to C_δ
      until (no change in the reassignment)
   2.5) update some quality index
until (the variation the quality index is negligible

implementation of the adaptive strategy

kernel of the algorithm

parallelism at the cluster level:the degree of parallelism if given by the **number of clusters** K

parallelism at the element level:the degree of parallelism if given by the **number of items** N

parallelism at cluster level

Usually $K \ll N$, so that this level of parallelism can be well managed by a **multi-core CPU according a SPMD multithreading programming model**

More precisely, given P the number of threads $t_1 \ldots, t_P$ running on the computing units of a multi-core CPU, this step defines P subpartitions $P_k^j \subset P_k$ with $j = 1, \ldots, P$, obtained assigning the clusters C_k to the thread t_j in a round-robin mode.

Step 2.4.1 can be redefined as follows

2.4.1-a) **for** each $\mathcal{P}_K^j \ (j = 1, \ldots, P)$ **in parallel**
2.4.1-b) **for** each cluster $C_k \in \mathcal{P}_K^j$
update clusters info
end for
end parallel for

on multicore CPU

parallelism at element levels

In this case, it is possible to define a parallelism at the element level that can be addressed **both by a multi-core CPU with the SPMD programming model, as well as by a GPU-based accelerator with a SIMD programming model**. For such reason, this step can be implemented with **a hybrid procedure** able to distribute the N elements to both the computing devices

More precisely, let be:

$$Q = \frac{PP_{GPU}}{PP_{CPU}}$$

the ratio between the peak performances of the GPU and of the CPU, it is possible to distribute the workload between the two devices according to their performance.

The N elements of S are split in two subsets S_{CPU} and S_{GPU} according to

$$N = N_{CPU} + N_{GPU} = \frac{N}{1+Q} + \frac{NQ}{1+Q}$$

the workload assigned to each device reflects the ratio between the peak performances Q

refinement of the step 2.4.2

Given, P be the number of threads running on the multi-core CPU, Step 2.4.2 defines P subsets $S^j \subset S_{CPU}$ $(j = 1,..,P)$ with N_{CPU}/P elements assigned in charge to each thread t_j according to the SPMD programming model.

The remaining elements in S_{GPU}, composed of N_{GPU} elements, can be processed with a SIMD programming model implemented by a GPU device. Each thread in the kernel can process a single element x_n .

Step 2.4.2 can be redefined as follows

2.4.2-a) **for** each $\mathbf{x}_n \in S^{GPU}$ **in parallel**
search the cluster of belonging C_δ
end parallel for

2.4.2-b) **for** each S^j $(j = 1, ..., P)$ **in parallel**
for each $\mathbf{x}_n \in S^j$
search the cluster of belonging C_δ
end for
end parallel for

on GPU

on multicore CPU

remark: the two steps 2.4.2-a) and 2.4.2-b) are executed concurrently on the CPU and the GPU

test problems

We used 4 dataset taken from Univ. California Machine Learning repository with different features

- **Letters**. This dataset is characterized by N = 20000 elements, each of them representing a black-and-white rectangular image of a letter in the English alphabet. This problem aims to classify the letters through d = 16 numerical attributes describing some specific feature (dimension of the image, edge counts, number of pixels, ...). For this test, the number of clusters is K = 26.

- **Wines**. In this problem, the N = 4898 items are samples of Portuguese wines described through d = 11 numerical attributes related to some organoleptic features (e.g., acidity, sweetness, alcohol content). According to their quality, the wines are clustered in K = 11 groups.

- **Cardio**. This problem arises to classify N = 2126 fetal cardiotocographic reports in K = 10 morphologic patterns. Each report is described by d = 21 diagnostic features (e.g., acceleration, pulsation, short- and long-term variability, and other physiological features.).

- **Clients**. The items in this dataset are related to a marketing campaign of a banking institution. Each of the N = 45211 elements is the numerical description of possible clients through d = 16 features (e.g., job, education, age, marital status). The clients are classified in K = 2 clusters.

computing environment and performance test

We conducted the performance experiments on the **NVIDIA Jetson Nano board**

- CPU Quad-core ARM Cortex-A57
- GPU Nvidia Maxwell 128 core
- 4 GB RAM LPDD4
- SSD eMCC 16 GB
- 5 / 10 Watt (TDP)
- 69,6 mm x 45 mm

We report performance results of Algorithm 2 with several execution mode:

a) P = 1 : basic execution using only 1 core on the CPU

b) P = 4 without GPU : execution using 4 cores of the multicore CPU

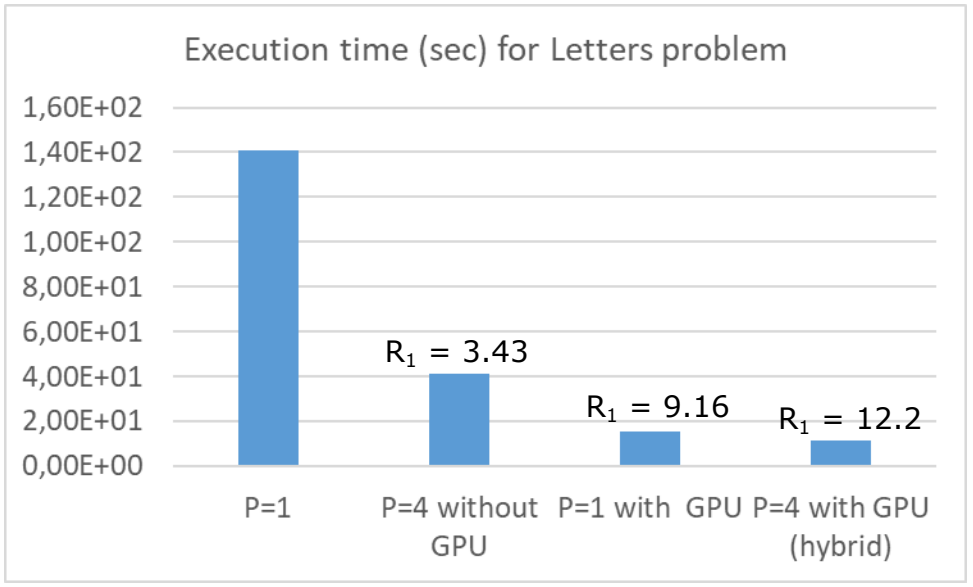
c) P = 1 with GPU : execution using only 1 core on the CPU and the GPU

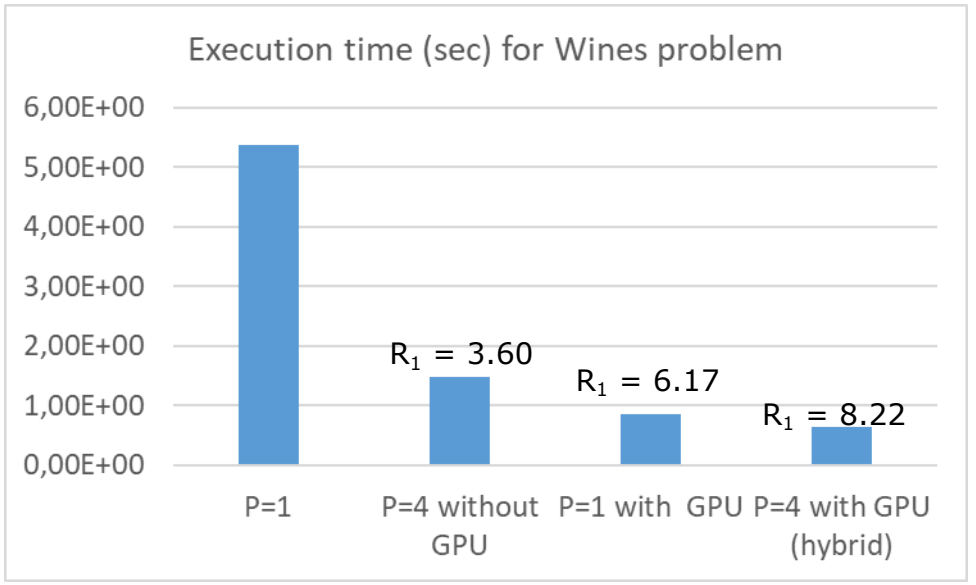
d) P = 4 with GPU : execution using full capability of the board (4 cores on the CPU and the GPU)

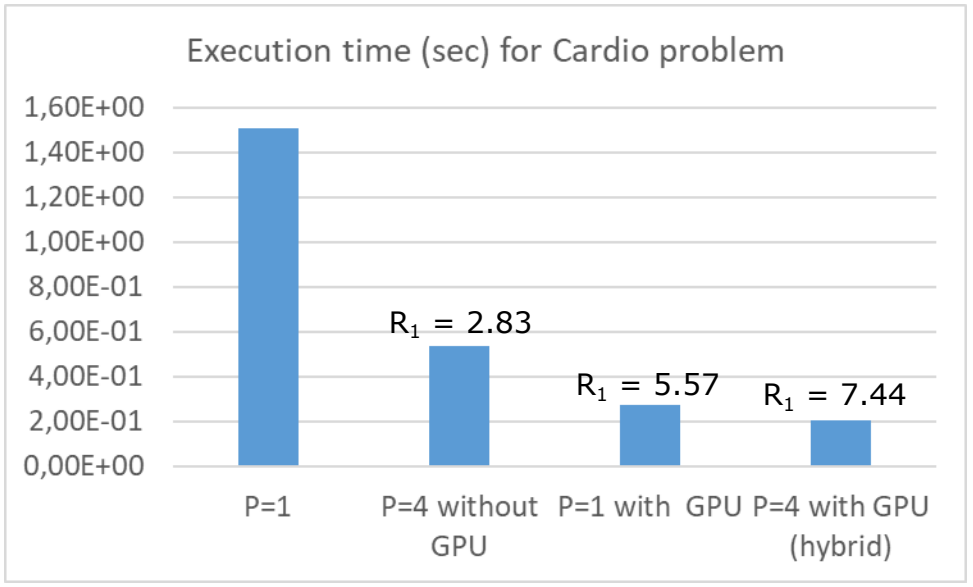
We report the execution time T and the performance gain of the mode b), c) and d) with respect to the mode a)

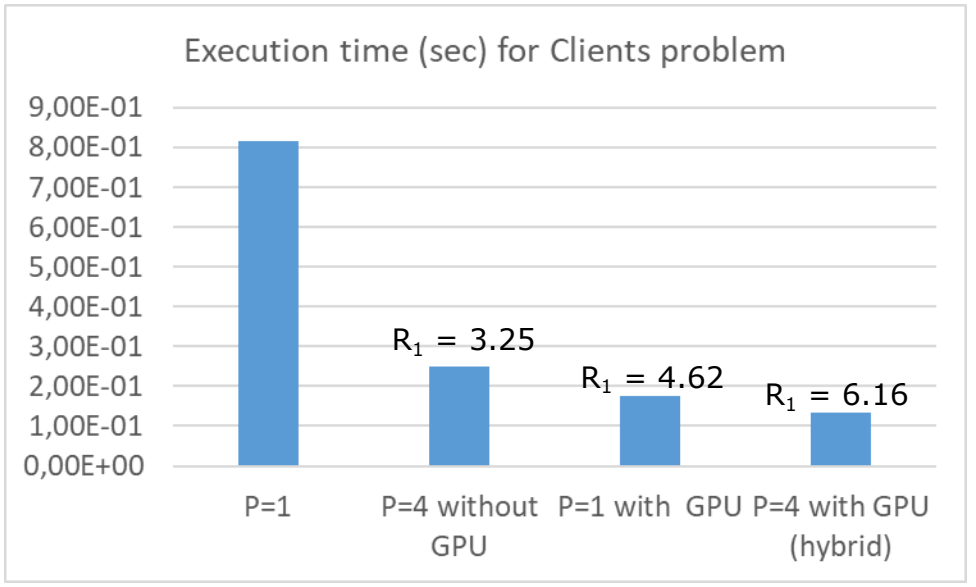
$$R_1 = \frac{T^{(a)}}{T^{(b,c,d)}}$$

execution time (step 2.4.2) and performance gain

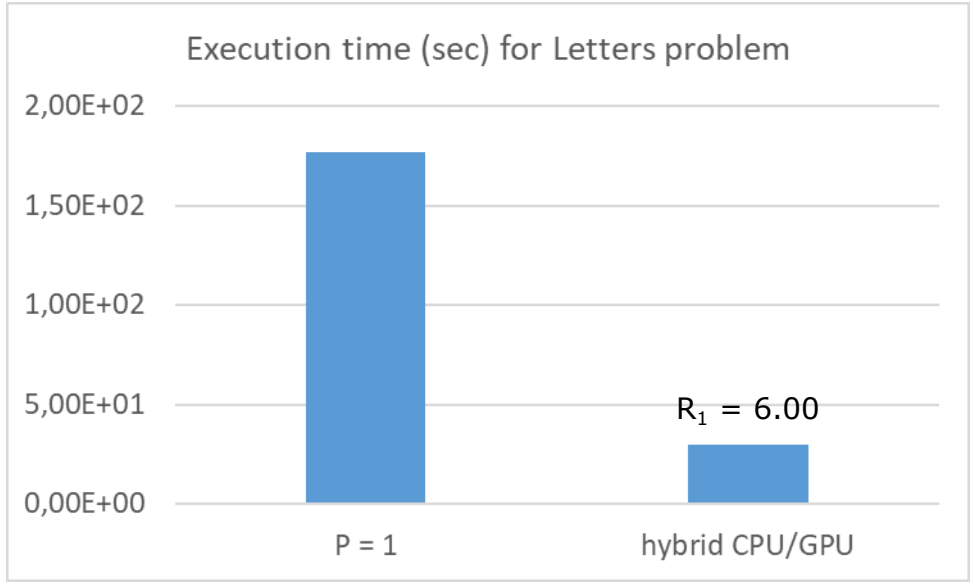


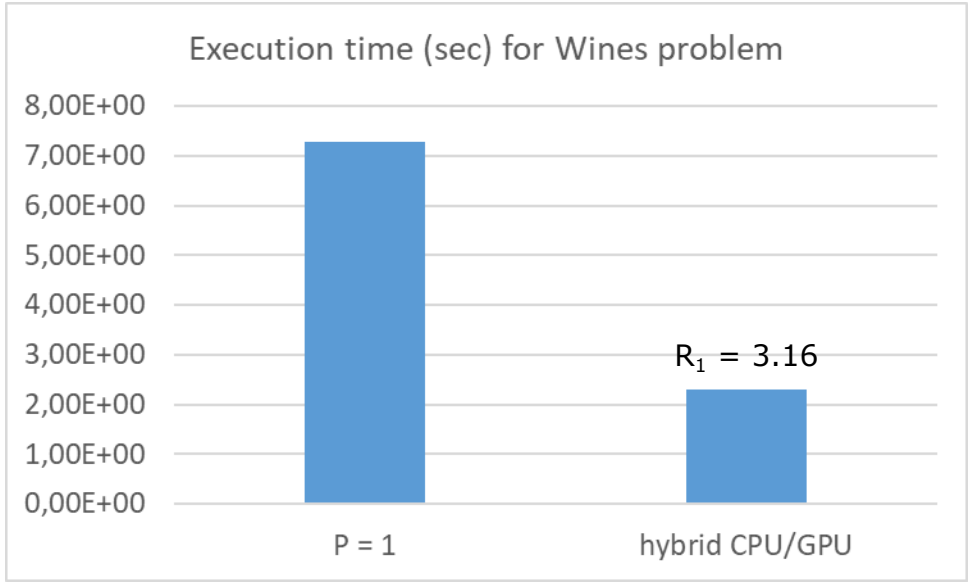


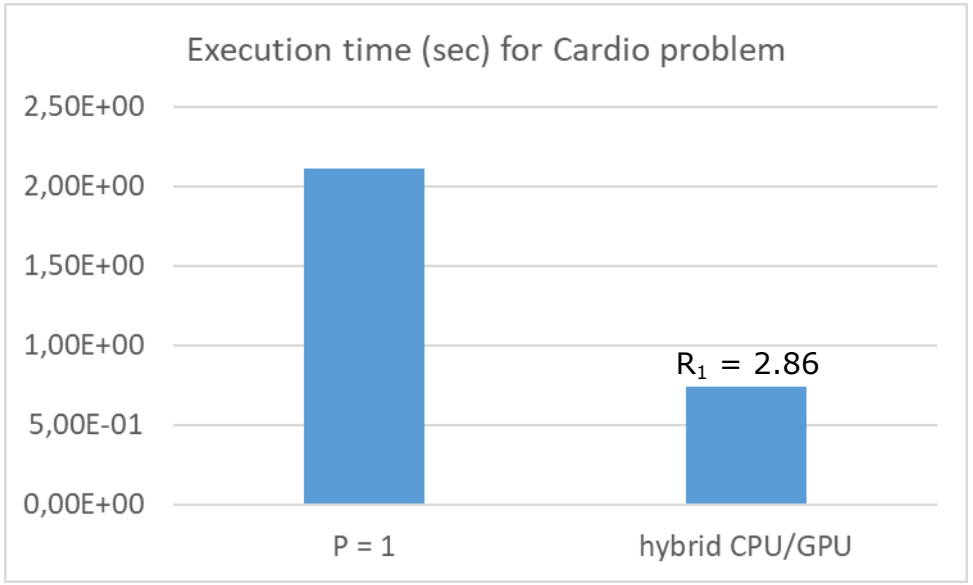


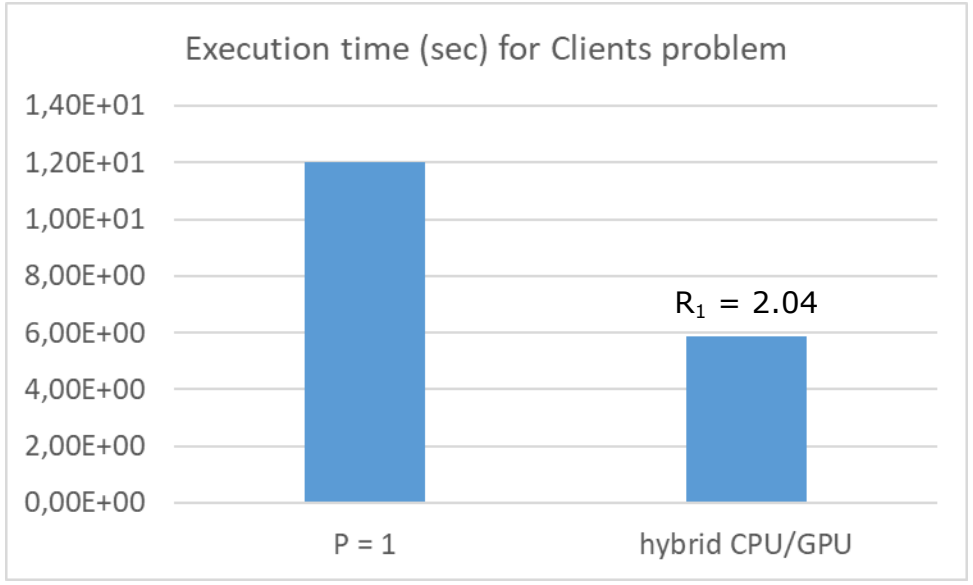


execution time (total) and performance gain









energy efficiency tests

We compared the energy efficiency of Algorithm 2 on the Nvidia Jetson Nano board with two other systems with high-end computing devices:

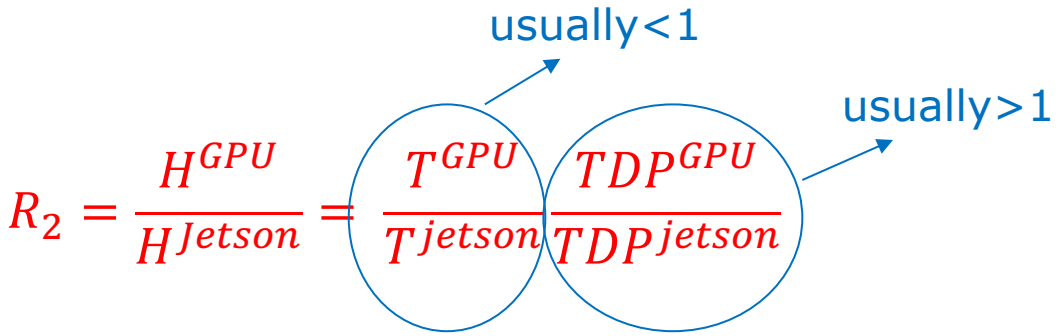
- (TK20): a system based on an NVIDIA Tesla K20c GPU (2012), with 2496 CUDA cores running at 0.706 GHz with a global memory of 5 GBytes and a TDP = 225 Watt. The host CPU is a 4-core Intel Core i7-950 running at 3.07 GHz
- (RTX): a system based on an NVIDIA GeForce RTX 3070 GPU (2020), with 5888 CUDA cores running at 1.75 GHz with a global memory of 8 Gbytes and a TDP = 220 Watt. The host CPU is an 8-core Intel Core i9-9900K running at 3.6 GHz

The energy efficiency has been measured through

$$R_2 = \frac{H^{GPU}}{H^{Jetson}}$$

$H = T_4 \cdot TDP = [sec][Watt] = [Joule]$ evaluates the total energy consumption for solving the problem

energy efficiency test



R_2 > 1 is profitable for Jetson nano

R_2 measures how much higher a GPU's total energy consumption is than that of the Nvidia Jetson nano board.

	Tesla K20	RTX 3070
Letters	11.1	5.47
Clients	8.5	4.63
Cardio	10.1	5.17
Wines	11.6	4.74

conclusions

- clustering algorithm are essential tools for shifting intelligence and decision-making ability for time-critical applications to the edge of the network
- we introduced a high-performance parallel adaptive approach for improving the performance of dynamic data clustering with the K-means algorithm.
- new edge devices integrate sensor pins and high-performance computing resources and can execute complex algorithms with greater energy efficiency than high-end CPU and/or GPU.