Data stream clustering for low-cost machines

Edge Computing / Experimental algorithms / Preliminary Work

christophe.cerin@univ-paris13.fr - June 8, 2021

Outline

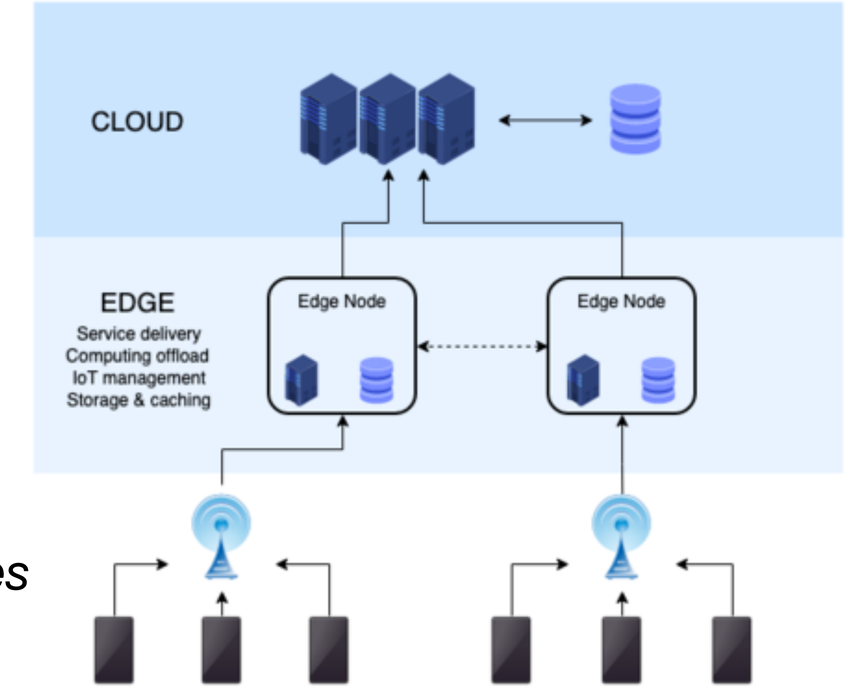
Data stream clustering for low-cost machines

- Context of the work:
    - Edge / Smart building / Machine Learning;
    - Problem statement;
- General framework to solve the problem;
- Instances;
- Experimental results (preliminary);
- Conclusion.

Elements of context

Data stream clustering for low-cost machines

- **Edge computing:** distributed computing paradigm that brings computation and data storage closer to the location where it is needed to improve
    - response times and
    - save bandwidth.
- **Smart Building definition:** *construction with appropriate design and technological supports that allow maximizing the functionalities and comfort offered to the occupants while reducing operational and maintenance costs, and extending the life of the physical structure*;
- Smart Building is an application field of our research and, in our view, an instance of Edge Computing;



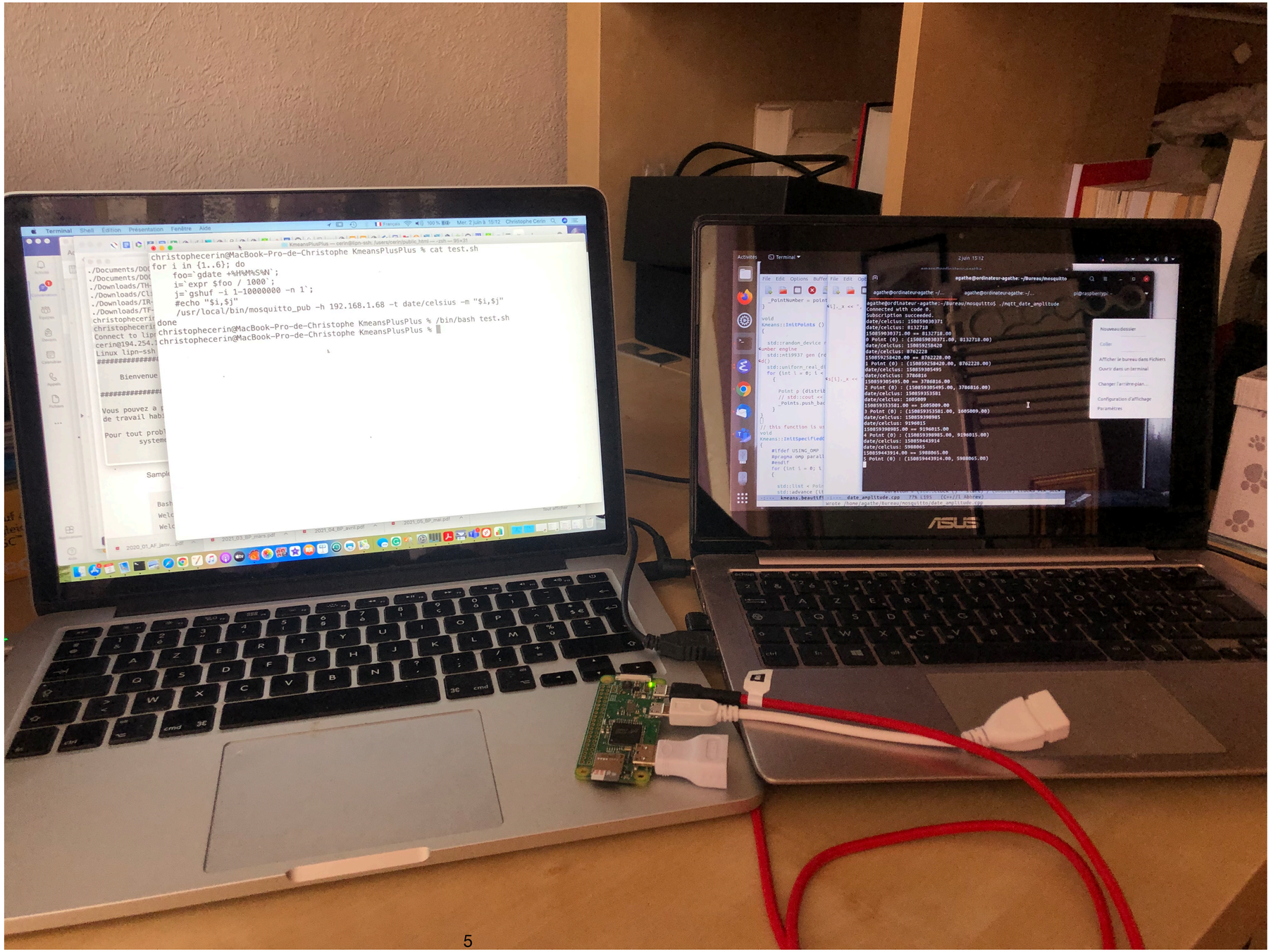
Elements of context

Data stream clustering for low-cost machines

- **Machine learning:** computer algorithms that improve automatically through experience and by the use of data.
  - Supervised: build a mathematical model of a set of data that contains both the inputs and the desired outputs;
  - Unsupervised: take a set of data that contains only inputs, and find structure in the data;
- **Clustering algorithm:** is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar.
- **Data stream (clustering) algorithms:** the clustering of data that arrive continuously such as telephone records, multimedia data, financial transactions etc. Using a small amount of memory and time… for us: using a **small number of resources**.

Context

Data stream clustering for low-cost machines



Elements of context: past findings and issues

Data stream clustering for low-cost machines

- HPC is the field for accelerating computation;
- Scikit-learn parallelism is done:
    - via the joblib library. (for me: mmap to share data - shared memory)
    - via OpenMP, used in C or Cython code.
- Apache Spark study (https://lipn.univ-paris13.fr/~cerin/DATACOM2019_final.pdf):
    - total number of functions in the MLlib library is 1319 whereas the total number of BLAS functions calls in the MLlib library is 113. That makes about 8.6% of the total number of functions present in MLlib library;
    - 8 BLAS functions in the MLlib library.
    - Scala K-means algorithm uses 3 BLAS functions: axpy(), dot() and scal().

Outline

Data stream clustering for low-cost machines

- **Problem statement:** propose efficient (in terms of resources) data stream (clustering) algorithms for edge computing.
- Control the quality of the result (compared to a baseline algorithm);
- Control the execution time, the throughput;
- Control the memory used by the algorithm;
- Implementable with companion IoT technologies i.e., easy to implement because the underlying idea is kept simple;

How to solve: models and assumptions

Data stream clustering for low-cost machines

- **Data stream model:** if the stream has length *n* and the domain has size *m*, algorithms are generally constrained to use space that is logarithmic in *m* and *n*. Make only some small constant number of passes over the stream, sometimes just one.

- **Sliding window model:** in this model, the function of interest is computing over a fixed-size window in the stream. As the stream progresses, items from the end of the window are removed from consideration while new items from the stream take their place.

- others

A baseline algorithm: STREAM

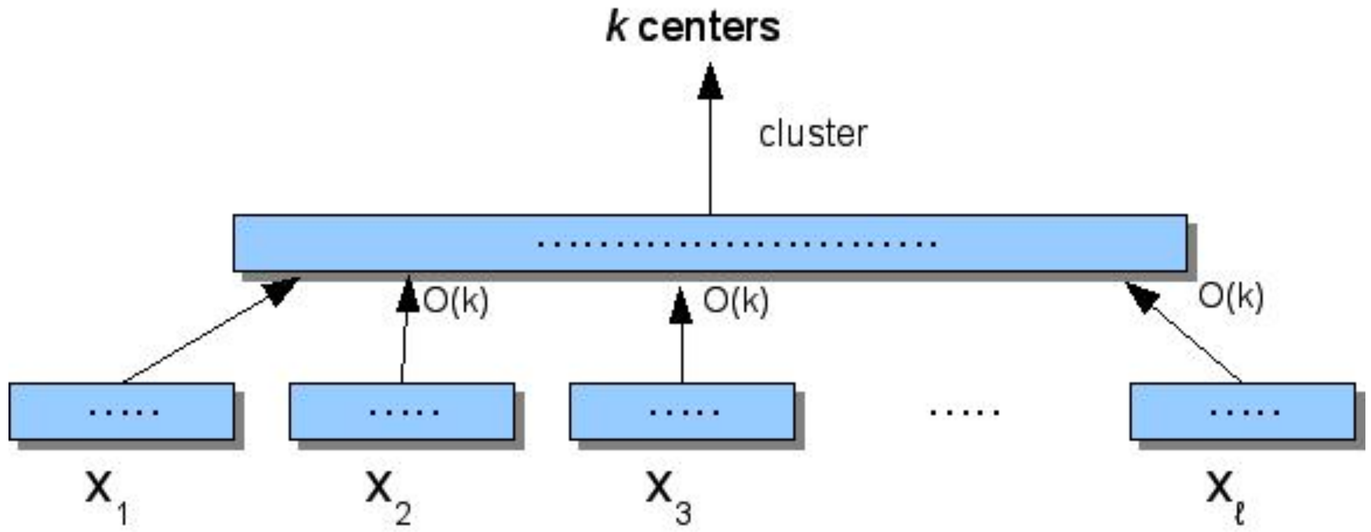
Data stream clustering for low-cost machines

- The problem of data stream clustering is defined as:
  - **Input:** a sequence of *n* points in metric space and an integer *k*.
  - **Output:** *k* centers in the set of the *n* points so as to minimize the sum of distances from data points to their closest cluster centers.
- STREAM achieves a constant factor approximation for the k-Median problem in a single pass and using small space.
- Small space = a divide-and-conquer algorithm that divides the data, *S*, into pieces, clusters each one of them (using *k*-means) and then clusters the centers obtained.

A baseline algorithm: Small-Space(S)

Data stream clustering for low-cost machines

- Require to run a bicriteria - approximation algorithm (to find O(k) centers in X_i) + a *c*-approximation algorithm for the last step;
- Problem: each subset fits in memory, and so that the weighted centers also fit in memory: (n/l)+lk < M. But such an l may not always exist.



STREAM algorithm

Data stream clustering for low-cost machines

- STREAM solves the problem of storing intermediate medians and achieves better running time and space requirements with a randomized alg. + local search + primal dual alg.
- Further readings on data streaming:
    - https://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/stream_icde.pdf (Streaming-Data Algorithms For High-Quality Clustering)
    - https://bdataanalytics.biomedcentral.com/articles/10.1186/s41044-016-0011-3 (State-of-the-art on clustering data streams)
    - https://florentfo.rest/files/Forest2021-manuscrit.pdf (Unsupervised Learning of Data Representations and Cluster Structures: Applications to Large-scale Health Monitoring of Turbofan Aircraft Engines)

Our proposal (1/2)

Data stream clustering for low-cost machines

The algorithm, at the highest level is:

1- Read the first n inputs of the data stream
repeat
    2- Cluster the n inputs with Kmeans++ or kmeans
    3- Sort the data in each of the obtained clusters with respect to the centroid
    4- Firing, in the sorted clusters, a total of k data (for example by considering regular intervals in the sorted vectors and by taking into account the number of samples in each cluster. You have to sample more where there is more data)
    5- Read, from the input data stream, k new data
forever

- Proposal based on well known algorithms (k-Means, Sorting) and methods (regular sampling, sliding window model)
- Weakness (today): analysis of the complexity => approximation alg. => to estimate how far we are from the optimal solution

Our proposal

Data stream clustering for low-cost machines

- The algorithm is based on offline clustering (traditional Kmeans or kmeans++);
- When we redo, round after round, the clustering of step 2, it is only k data that must be inserted (memory control);
- The algorithm is based on sorting and we know how to do parallel sorting (for step 3);
- Each sort in step 3 can be done in parallel (and the more clusters you have, the more parallel work you have to do);
- The idea of eliminating k data by taking them in a regular way corresponds to the idea of preserving diversity in the input.

Our proposal (2/2): what has been implemented

Data stream clustering for low-cost machines

The second algorithm, at the highest level is:

1- Read the first n inputs of the data stream
repeat
    2- Cluster the n inputs with Kmeans++ or kmeans
    3- Sort the data, resulting of a Kmeans
    4- Firing, in the sorted vector, a total of k data (for example by considering regular intervals in the sorted vector) but not the centroids
    5- Read, from the input data stream, k new data
forever

- Again, our proposal based on well known algorithms (k-Means, Sorting) and methods (regular sampling, sliding window model) => **Methodology for other ML streaming Alg**
- Weakness (today): same as previously

Quality of the data stream clustering algorithm

Data stream clustering for low-cost machines

- Experimentally we can estimate the quality of the results;
- Consider the set of centroids obtained after reading N data with our proposal;
- Consider the set of centroids obtained for N data with the corresponding offline algorithm and when the N data are all available at the beginning;
- Compare the two sets with a similarity analysis (Jaccard index for instance)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

If $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ are two vectors with all real $x_i, y_i \geq 0$ then their Jaccard similarity coefficient (also known then as Ruzicka similarity) is defined as

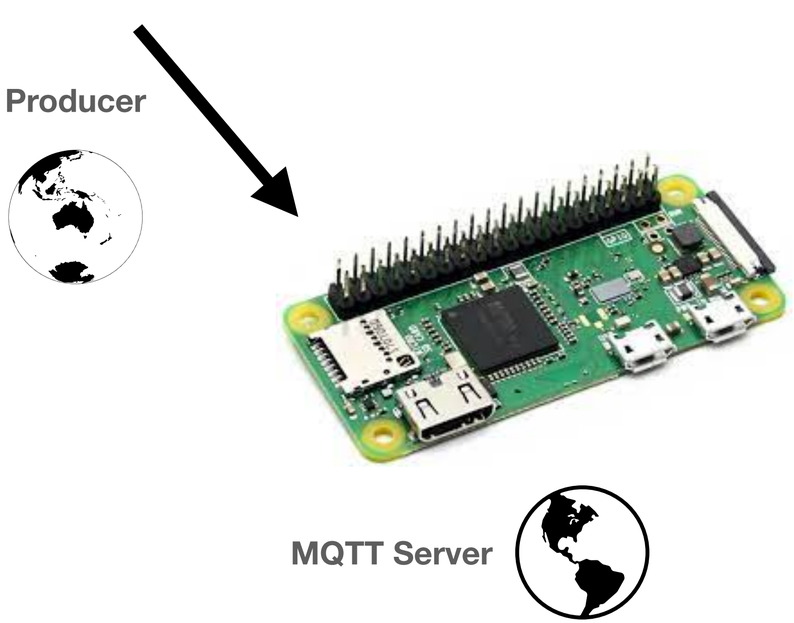
$$J(\mathbf{x},\mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

Experimental results

Data stream clustering for low-cost machines

KmeansPlusPlus — cerin@lipn-ssh: /users/cerin/public_html — -zsh — 95×31

```
christophecerin@MacBook-Pro-de-Christophe KmeansPlusPlus % cat test.sh
for i in {1..6}; do
    foo=`gdate +%H%M%S%N`;
    i=`expr $foo / 1000`;
    j=`gshuf -i 1-10000000 -n 1`;
    #echo "$i,$j"
    /usr/local/bin/mosquitto_pub -h 192.168.1.68 -t date/celsius -m "$i,$j"
done
christophecerin@MacBook-Pro-de-Christophe KmeansPlusPlus % /bin/bash test.sh
christophecerin@MacBook-Pro-de-Christophe KmeansPlusPlus %
```



agathe@ordinateur-agathe: ~/Bureau/mosquitto

agathe@ordinateur-agathe: ~/... | agathe@ordinateur-agathe: ~/... | pi@raspberrypi: ~

```
agathe@ordinateur-agathe:~/Bureau/mosquitto$ ./mqtt_date_amplitude
Connected with code 0.
Subscription succeeded.
date/celcius: 150859030371
date/celcius: 8132718
150859030371.00 == 8132718.00
0 Point (0) : (150859030371.00, 8132718.00)
date/celcius: 150859258420
date/celcius: 8762228
150859258420.00 == 8762228.00
1 Point (0) : (150859258420.00, 8762228.00)
date/celcius: 150859305495
date/celcius: 3786816
150859305495.00 == 3786816.00
2 Point (0) : (150859305495.00, 3786816.00)
date/celcius: 150859353581
date/celcius: 1605009
150859353581.00 == 1605009.00
3 Point (0) : (150859353581.00, 1605009.00)
date/celcius: 150859398985
date/celcius: 9196015
150859398985.00 == 9196015.00
4 Point (0) : (150859398985.00, 9196015.00)
date/celcius: 150859443914
date/celcius: 5988065
150859443914.00 == 5988065.00
5 Point (0) : (150859443914.00, 5988065.00)
```

Experimental results

Data stream clustering for low-cost machines

```
pi@raspberrypi:~ $ more /proc/cpuinfo
processor       : 0
model name      : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS        : 997.08
Features        : half thumb fastmult vfp edsp java tls
CPU implementer: 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xb76
CPU revision    : 7
Hardware        : BCM2835
Revision        : 9000c1
Serial          : 000000000f6f3e50
Model           : Raspberry Pi Zero W Rev 1.1
pi@raspberrypi:~ $ more /proc/meminfo
MemTotal:       440412 kB
```

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz
stepping        : 9
microcode       : 0x21
cpu MHz         : 1796.041
cache size      : 3072 KB
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass…
bogomips        : 3591.74
agathe@ordinateur-agathe:~$ cat /proc/meminfo
MemTotal:       3909456 kB
```

```
processor       : 3
model name      : ARMv7 Processor rev 3 (v7l)
BogoMIPS        : 126.00
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
CPU implementer: 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xd08
CPU revision    : 3
Hardware        : BCM2711
Revision        : d03114
Serial          : 10000000bcf9086d
Model           : Raspberry Pi 4 Model B Rev 1.4
pi@raspberrypi:~ $ cat /proc/meminfo
MemTotal:       8064280 kB
```

Experimental results

Data stream clustering for low-cost machines

K = 5 ; PointNumber = 256 ; Kmeans ; Stop after reading K data ; SSID WiFi 5GHz Livebox-ee94_5GHz

```
agathe@ordinateur-agathe:~/Bureau/mosquitto$ ./mqtt_date_amplitude
Connected with code 0.
Subscription succeeded.
iteration 0 sum 2.45854e+06
iteration 1 sum 666723
iteration 2 sum 640997
iteration 3 sum 720149
iteration 4 sum 360741
iteration 5 sum 325667
iteration 6 sum 251289
iteration 7 sum 208230
iteration 8 sum 252775
iteration 9 sum 207949
iteration 10 sum 210444
iteration 11 sum 168065
iteration 12 sum 203401
iteration 13 sum 46346
iteration 14 sum 0
calculate time 0.008045
^C
```

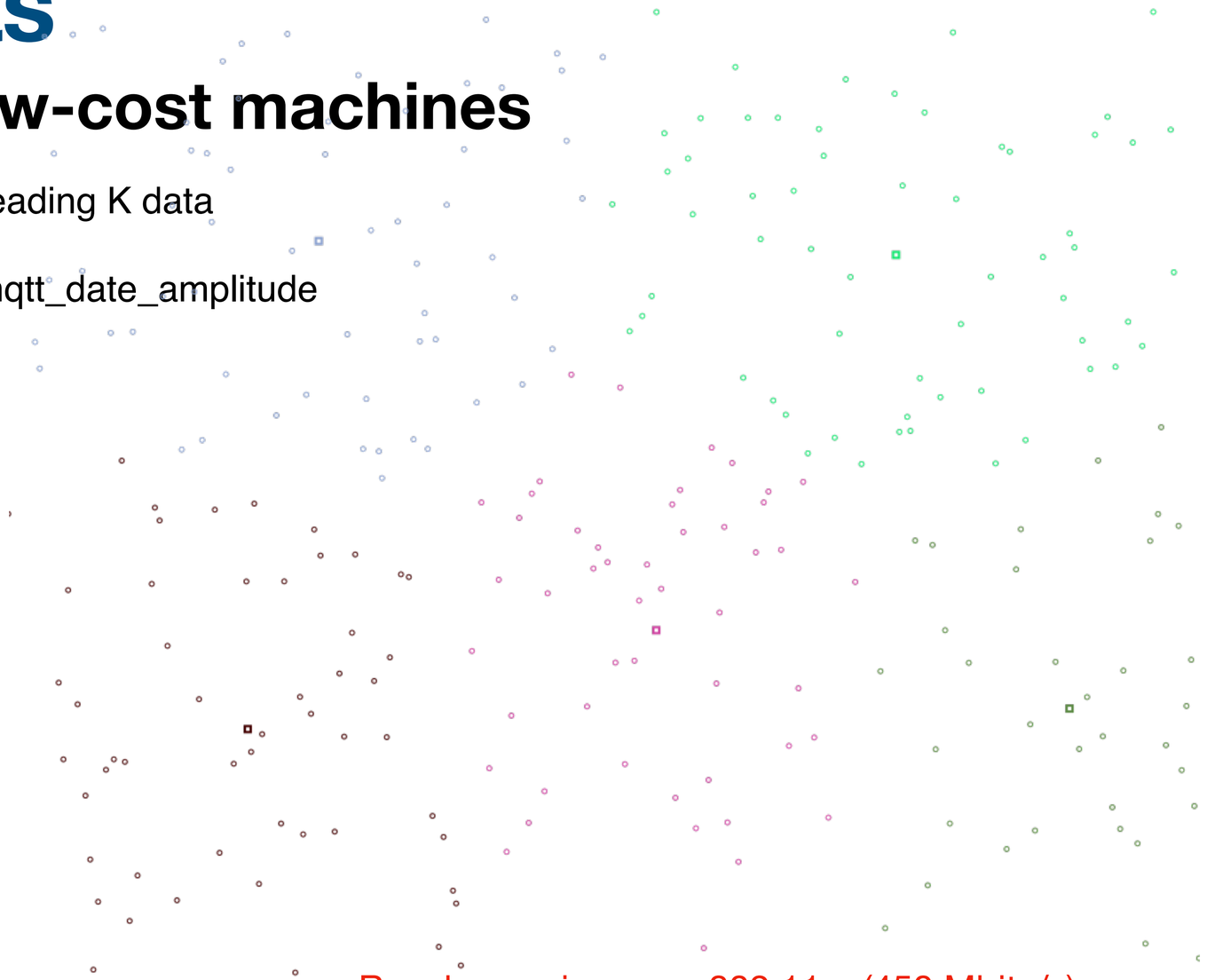
```
$ cat test.sh
for i in {1..256}; do
    foo=`gdate +%H%M%S%N`;
    i=`expr $foo / 1000`;
    j=`gshuf -i 1-10000000 -n 1`;
    #echo "$i,$j"
    /usr/local/bin/mosquitto_pub -h 192.168.1.68 -t date/celsius -m "$i,$j"
done

$ time /bin/bash test.sh
/bin/bash test.sh  2,26s user 3,07s system 40% cpu 13,099 total

$ time /bin/bash test.sh
/bin/bash test.sh  2,22s user 2,97s system 45% cpu 11,404 total
```

Experimental results

Data stream clustering for low-cost machines

K = 25 ; PointNumber = 256 ; Kmeans ; Stop after reading K data

```
agathe@ordinateur-agathe:~/Bureau/mosquitto$ ./mqtt_date_amplitude
Connected with code 0.
Subscription succeeded.
iteration 0 sum 8.3589e+06
iteration 1 sum 2.68866e+06
iteration 2 sum 1.48814e+06
iteration 3 sum 1.02201e+06
iteration 4 sum 822540
iteration 5 sum 363562
iteration 6 sum 232541
iteration 7 sum 683978
iteration 8 sum 185060
iteration 9 sum 115649
iteration 10 sum 0
calculate time 0.007392
^C
```



Raspberry pi zero w: 802.11 n (450 Mbits/s)
Livebox 802.11 ac (1732 Mbits/s)

Experimental results

Data stream clustering for low-cost machines

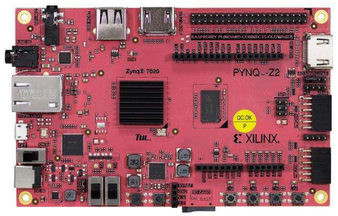
```
Kmeans computation (N=256; K=5):
iteration 0 sum 7593905.000000
iteration 1 sum 3346552.500000
iteration 2 sum 2379568.750000
iteration 3 sum 1774137.500000
iteration 4 sum 1545260.125000
iteration 5 sum 2282675.500000
iteration 6 sum 1570049.000000
iteration 7 sum 1221824.250000
iteration 8 sum 1146263.375000
iteration 9 sum 1365610.875000
iteration 10 sum 1248434.000000
iteration 11 sum 771554.062500
iteration 12 sum 640943.625000
iteration 13 sum 419980.875000
iteration 14 sum 232290.250000
iteration 15 sum 110277.968750
iteration 16 sum 0.000000
```

```
Sampling => sqrt(256)-1 = 15 pivots
Not a center    15     : 155146289152.000000    2750836.000000   1
Not a center    31     : 155147010048.000000    4913787.000000   1
Not a center    47     : 155147862016.000000    8583628.000000   0
Not a center    63     : 155148615680.000000    981059.000000    1
Not a center    79     : 155149352960.000000    2616151.000000   1
Not a center    95     : 155150172160.000000    7569222.000000   0
Not a center    111    : 155151335424.000000    4557428.000000   3
Not a center    127    : 155152039936.000000    8907295.000000   2
Not a center    143    : 155152826368.000000    5584707.000000   3
Not a center    159    : 155153612800.000000    8368685.000000   2
Not a center    175    : 155154317312.000000    3506426.000000   4
Not a center    191    : 155155202048.000000    3006091.000000   4
Not a center    207    : 155155873792.000000    5347767.000000   4
Not a center    223    : 155156512768.000000    7784450.000000   2
Not a center    239    : 155157184512.000000    1511982.000000   4
Centers:
0    155148468224.000000    7623876.500000
1    155147632640.000000    2451648.750000
2    155155431424.000000    7868651.500000
3    155152171008.000000    3123204.250000
4    155155906560.000000    2893622.750000
```

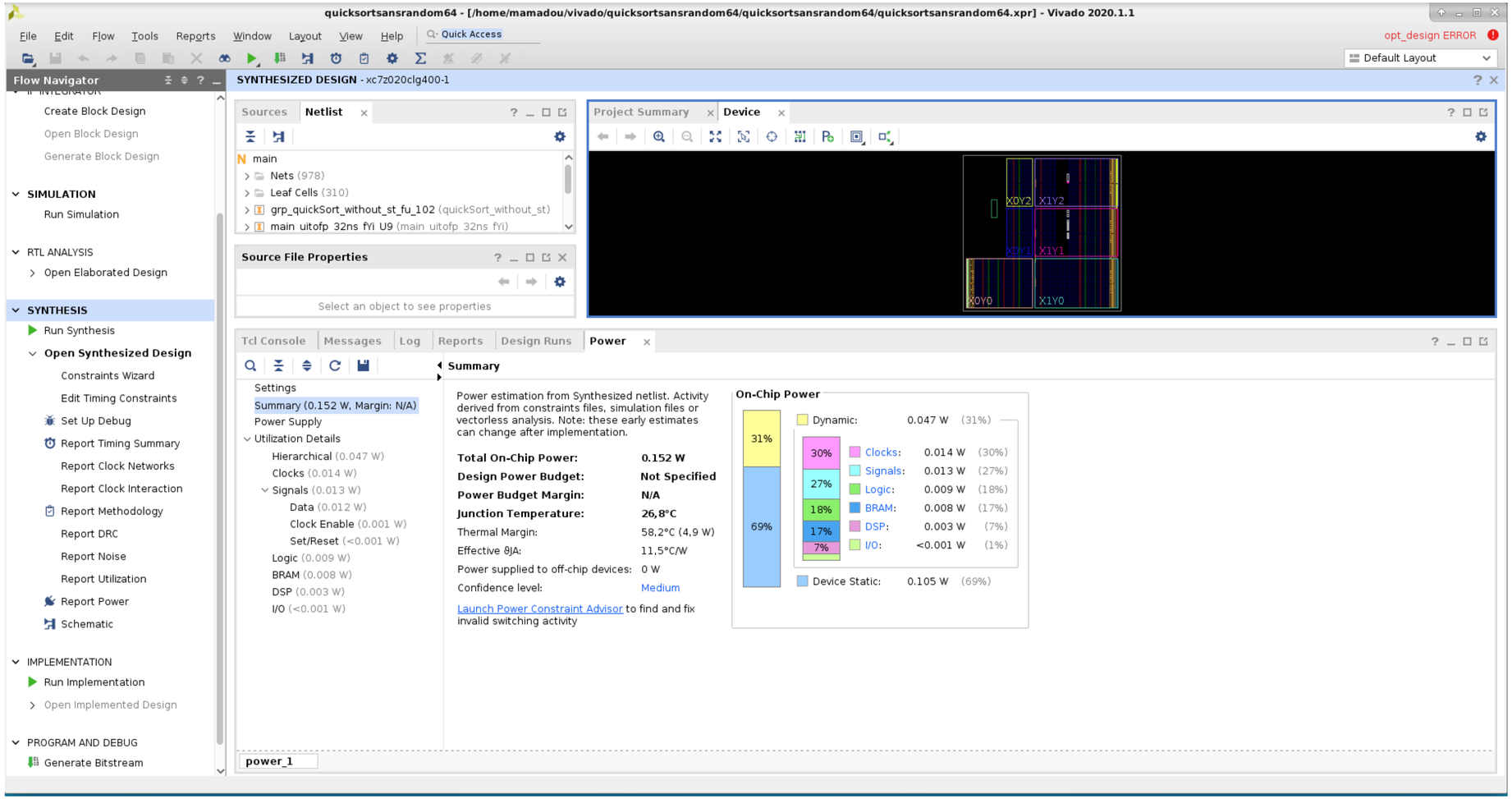
Experimental results: FPGA (Xilinx)

Data stream clustering for low-cost machines

Vivado from Xilinx



PYNQ-Z2 board from Xilinx

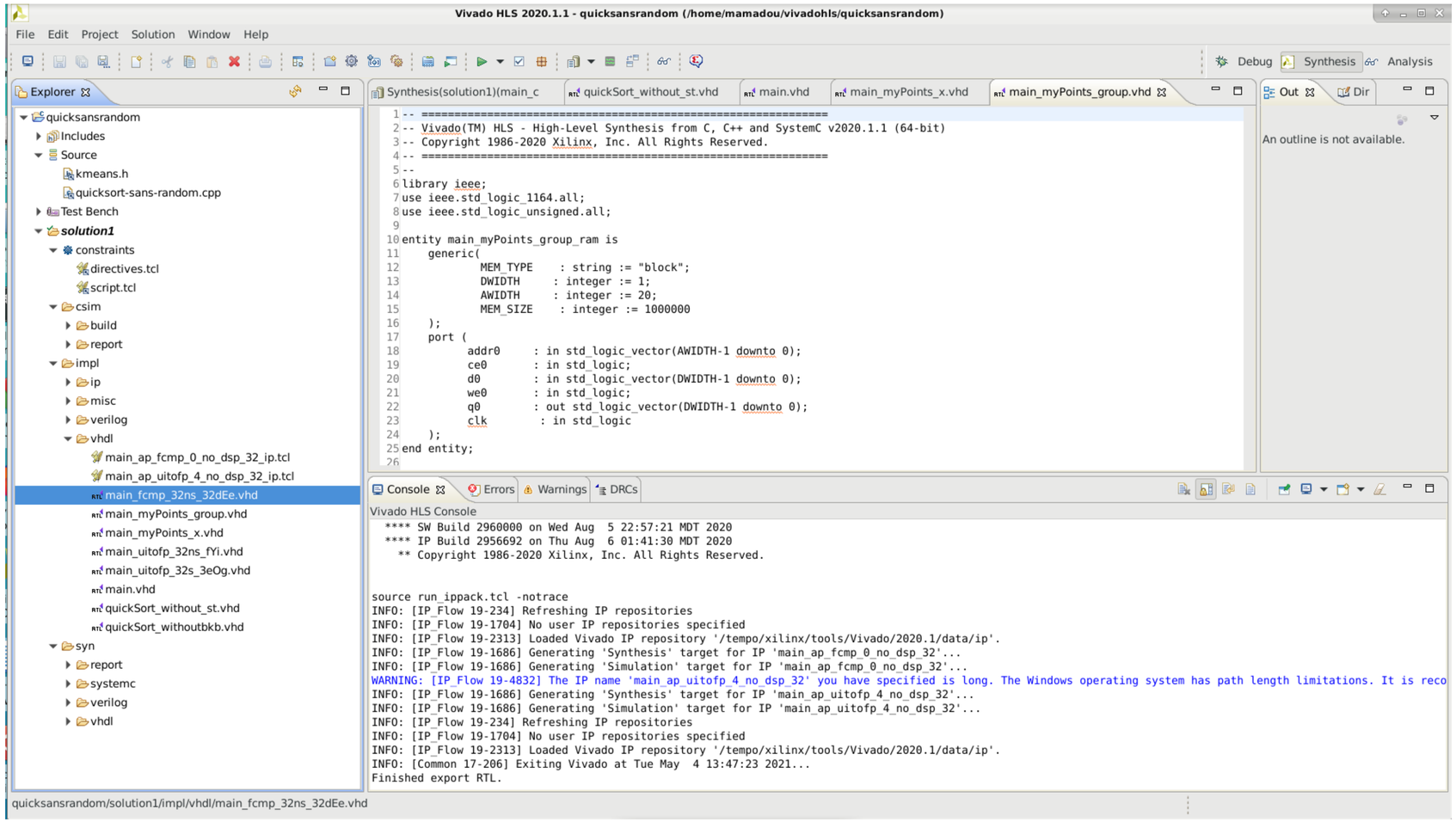


Experimental results: FPGA (Xilinx)

Data stream clustering for low-cost machines

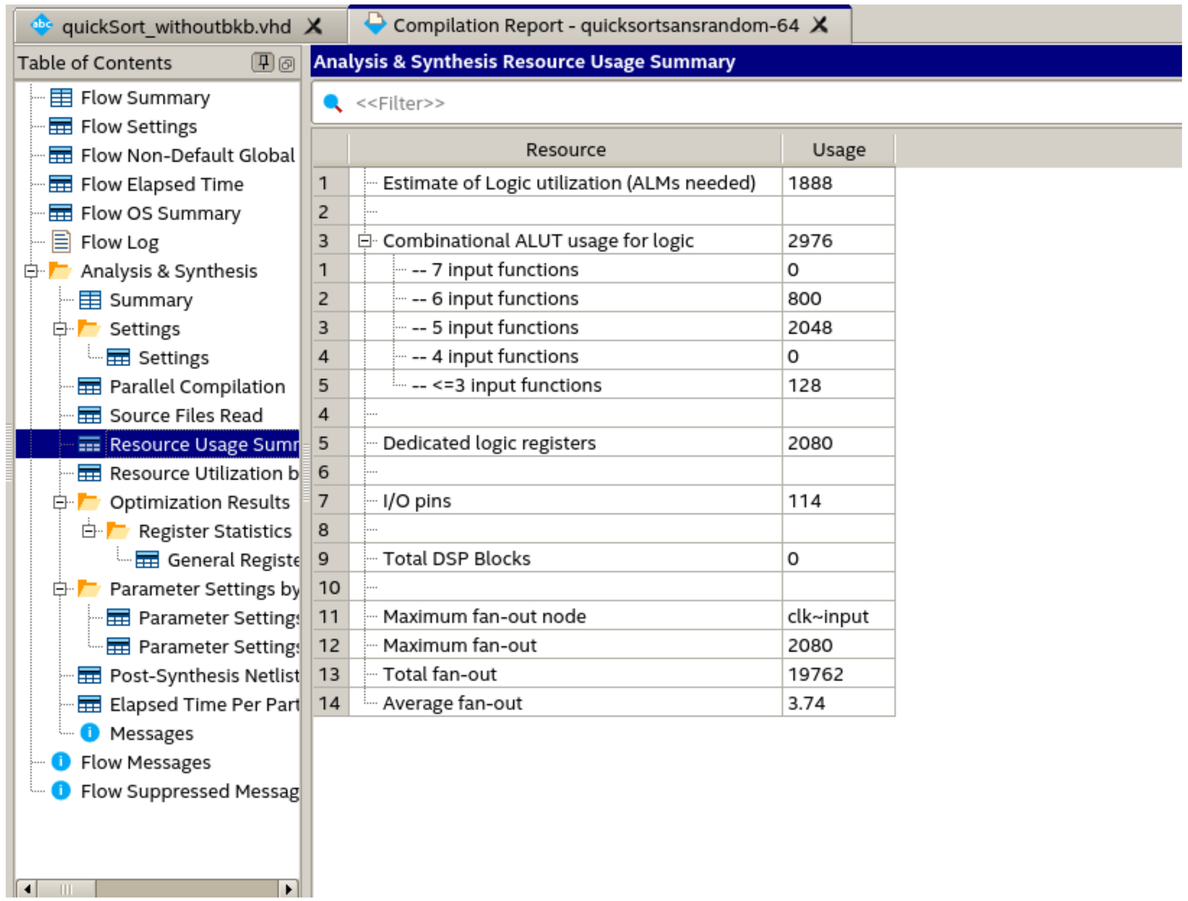
N = 1 000 000

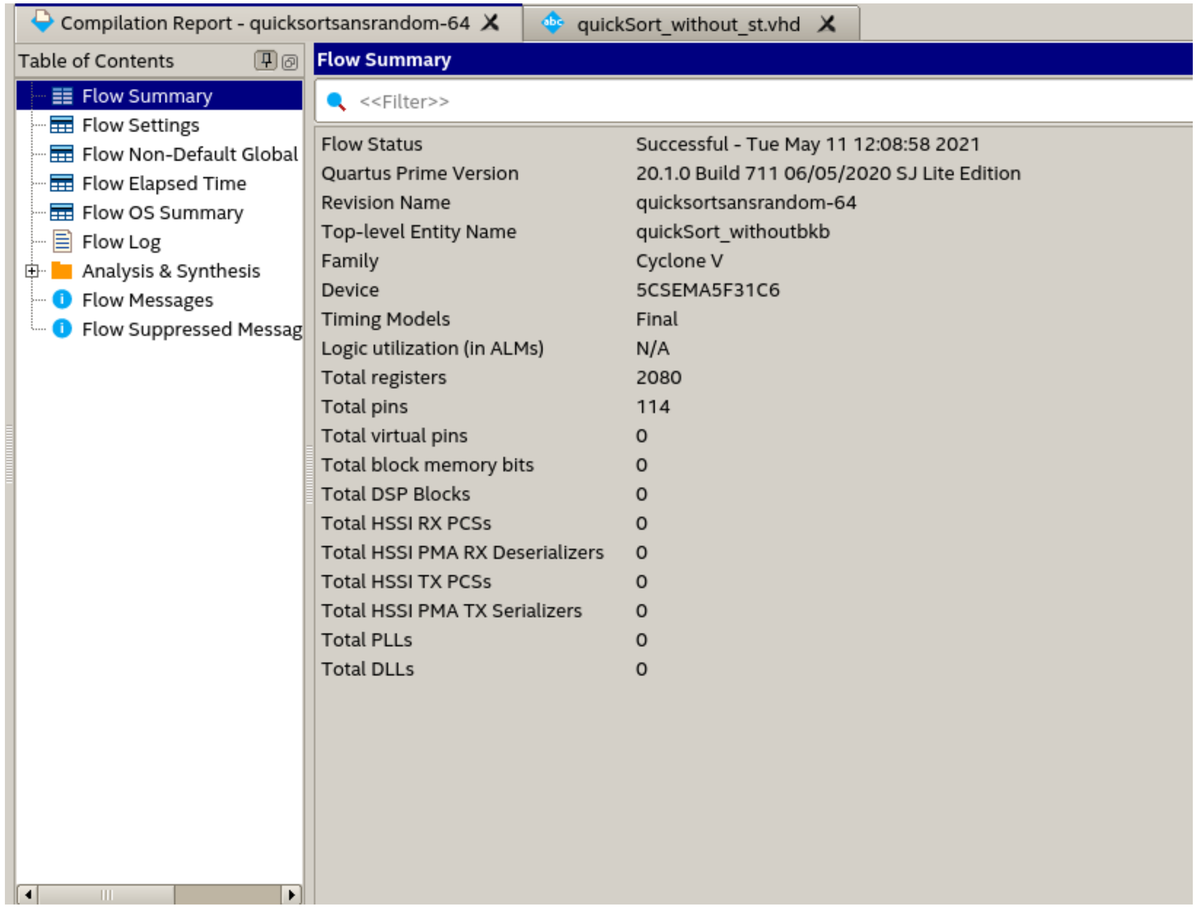
Name	BRAM_18k	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	4	0	163	-
FIFO	-	-	-	-	-
Instance	4096	0	2311	3379	0
Memory	4160	-	128	0	0
Multiplexer	-	-	-	260	-
Register	-	-	279	-	-
Total	8256	4	2718	3802	0
Available	280	220	106400	53200	0
Utilization	2948	1	2	7	0



Experimental results: FPGA (Intel Quartus Lite)

Data stream clustering for low-cost machines





Experimental results: FPGA

Data stream clustering for low-cost machines

bambu-quicksortcerin-pact2019-SIMD-first-cyclone-32768.log

Afficher Maint. Effacer Recharger Partager Rechercher

```
mkdir: impossible de créer le répertoire «hls-quicksortcerinpact2019SIMDfirstcycloneV32768»: Le
fichier existe
 ==  Bambu executed with: bambu --device-name=5CSEMA5F31C6 --simulate -fwhole-program -fno-delete-
null-pointer-checks --clock-period=10 --experimental-setup=BAMBU-BALANCED-MP -fdisable-tree-cunroll
-fdisable-tree-ivopts --param max-inline-insns-auto=1000 ../quicksort-sans-random-derecurs-32768.cpp


******************************************************************************
                 ____                  _
                | __ )  __ _ _ __ ___ | |_   _   _
                |  _ \ / _` | '_ ` _ \| '_ \| | | |
                | |_) | (_| | | | | | | |_) | |_| |
                |____/ \__,_|_| |_| |_|_.__/ \__,_|

******************************************************************************
                        High-Level Synthesis Tool

                        Politecnico di Milano - DEIB
                         System Architectures Group
******************************************************************************
               Copyright (C) 2004-2020 Politecnico di Milano
  Version: PandA 0.9.6 - Revision 5e5e306b86383a7d85274d64977a3d71fdcff4fe-main

Target technology = FPGA
Warning: GCC 8.4.0 has not been tested with the PandA framework
Warning: GCC 8.4.0 has not been tested with the PandA framework
Warning: GCC 8.4.0 has not been tested with the PandA framework
Warning: GCC 8.4.0 has not been tested with the PandA framework
Warning: GCC 8.4.0 has not been tested with the PandA framework
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  The top function inferred from the specification is: main
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

  Functions to be synthesized:
    __internal_bambu_memcpy
    __internal_puts
    __umodsi3
    __uint32_to_float32if
    __float32_leif
    __float32_geif
    _Z8printerrP5Point.constprop.2
```

bambu-quicksortcerin-pact2019-SIMD-first-zynq-32768.log

Afficher Maint. Effacer Recharger Partager Rechercher

```
Time to perform register binding: 0.02 seconds


Register binding information for function main:
  Register allocation algorithm obtains a sub-optimal result: 100 registers(LB:28)
Time to perform register binding: 0.02 seconds


Register binding information for function main:
  Register allocation algorithm obtains a sub-optimal result: 101 registers(LB:28)
Time to perform register binding: 0.02 seconds


Module binding information for function main:
  Number of modules instantiated: 473
  Number of possible conflicts for possible false paths introduced by resource sharing: 140
  Estimated resources area (no Muxes and address logic): 18452
  Estimated area of MUX21: 987
  Total estimated area: 19439
  Estimated number of DSPs: 6
  Slack computed in 0.00 seconds
  False-loop computation completed in 0.00 seconds
  Weight computation completed in 0.03 seconds
  Clique covering computation completed in 0.05 seconds
Time to perform module binding: 0.08 seconds


Register binding information for function main:
  Register allocation algorithm obtains a sub-optimal result: 101 registers(LB:28)
Time to perform register binding: 0.02 seconds


Connection Binding Information for function main:
  Number of allocated multiplexers (2-to-1 equivalent): 92
Time to perform interconnection binding: 0.01 seconds

Total number of flip-flops in function main: 2795
Total cycles             : 12638368 cycles
Number of executions     : 1
Average execution        : 12638368 cycles
```

Conclusion

Data stream clustering for low-cost machines

- Motivations for revisiting ML algorithms for low-cost machines:
    - Edge computing context;
    - Parallel / distributed ML algorithms still challenging, in general.
- Our proposal: a generic methodology;
- Our experiments: based on instances of Kmeans-xxx, sorting, sampling;
- Future works for the coming paper:
    - memory model for FPGA + decide what we externalize on the FPGA chip;
    - experiments and lessons learned.
    - complexity analysis
- Special thanks to Mamadou Sow (LIPN)