DAMDID/RCDL'2017, Data Analytics and Management in Data Intensive Domains
Moscow, Russia, October 10–13, 2017

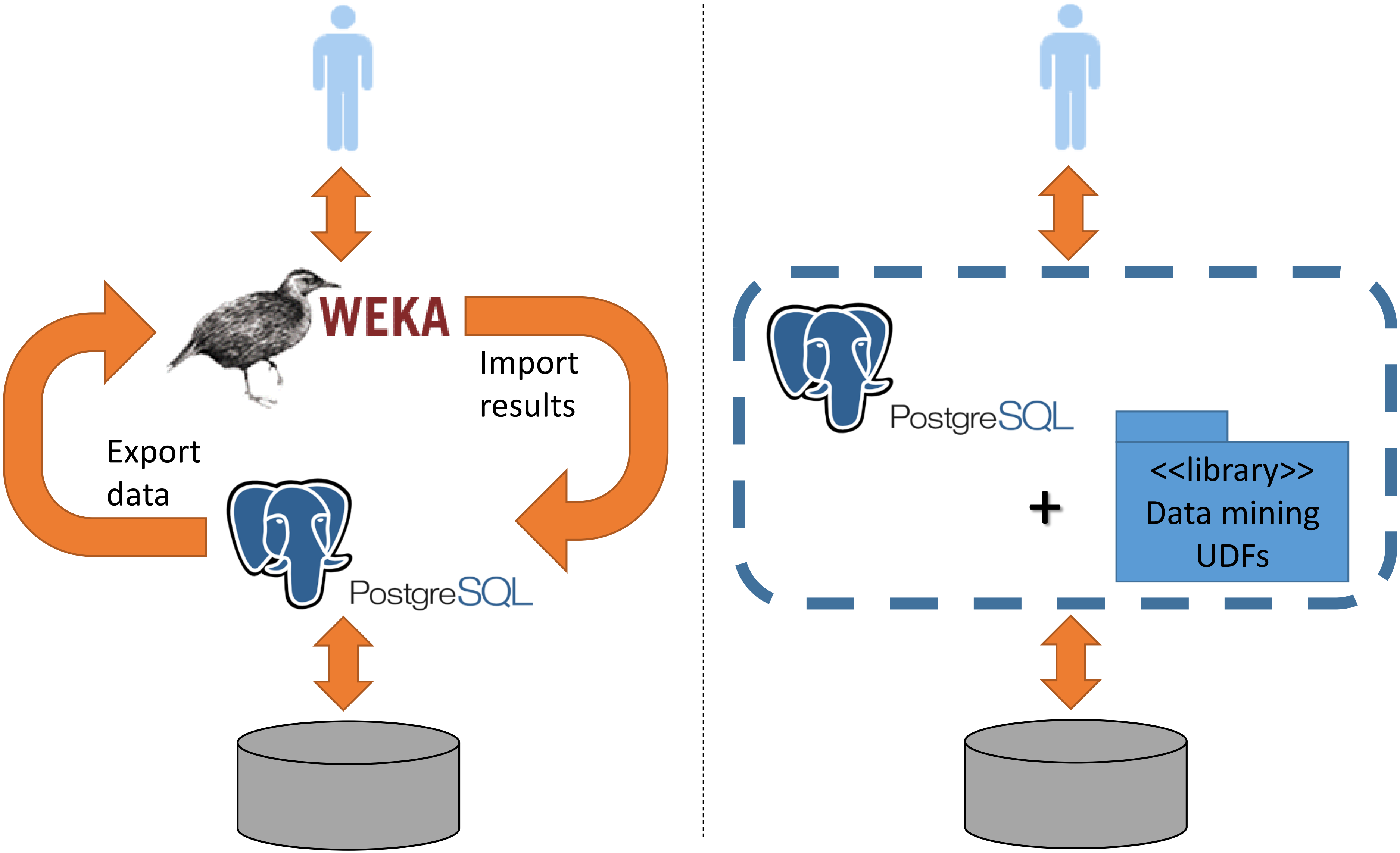
An approach to data mining inside PostgreSQL based on parallel implementation of UDFs

Timofey Rechkalov and Mikhail Zymbler
South Ural State University, Chelyabinsk, Russia
trechkalov@yandex.ru, mzym@susu.ru

This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463),
by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and
by the Ministry of education and science of Russian Federation (government order 1.9624.2017/7.8).

Authors thank RSC Group (Moscow, Russia) for the Intel Xeon Phi Knights Landing for experimental evaluation.

Data mining outside *vs.* inside DBMS



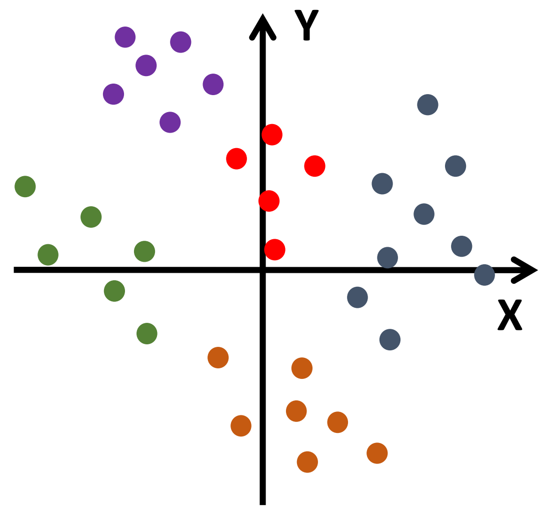
Data mining inside DBMS

Points

X	Y	Z	I	J	K	…
…						
…						

dim=2 k=5

```
procedure Clustering (
  inpTable text,  -- Input table name
  dim int,        -- # of columns
  k int,          -- # of clusters
  Eps real        -- Accuracy
) return text
begin
…
end
```




```
sql> exec Clustering('Points', 2, 5, 0.001);
```

X	Y	CLUSTER
…	…	
…	…	
…	…	
…	…	
…	…	

Data mining inside PostgreSQL

```
#include <libpq-fe.h> // API of PostgreSQL
#include "pgmining.h" // API of pgMining library

void main (void)
{
   char *inpTable = "Points“;// Table with data to be mined
   int dim = 3;                // Number of columns
   int k = 5;                  // Number of clusters
   float Eps = 0.001;          // Accuracy
   char *outTable = "Clusters"; // Table to save mining results

   // Connect to server
   char *conninfo = "user=postgres port=5432 host=localhost";
   PGconn *conn = PQconnectdb (conninfo);

   // Call mining UDF
   pgClustering (conn, inpTable, dim, k, Eps, outTable);

   PQexec (conn, strcat ("SELECT * FROM ", outTable));// Show results
   PQfinish (conn); // Cleanup
}
```

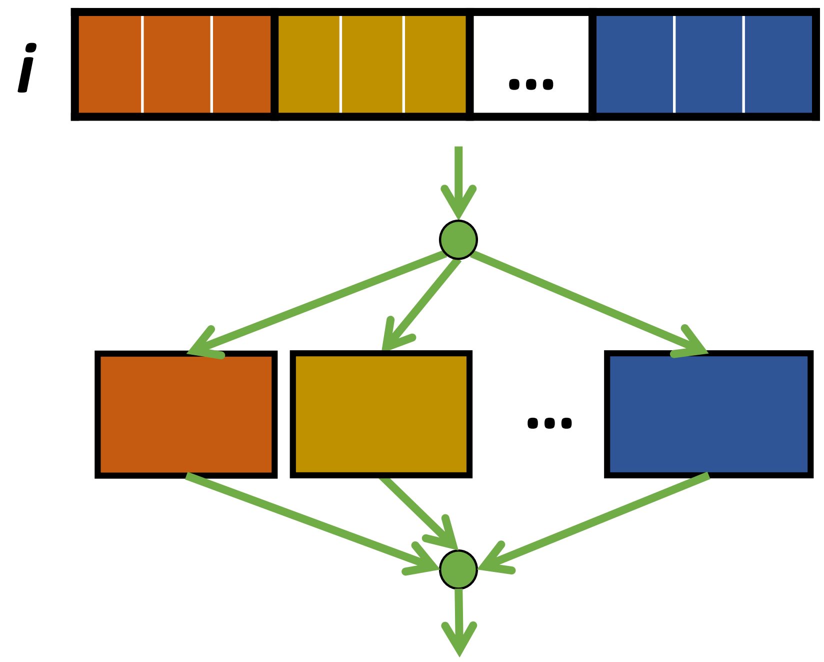
What mining UDF encapsulates

- Parallel implementation by OpenMP

```
#pragma omp parallel for
for (i=0; i<n; i++) {
  ...
  ...
}
```

fork

join



What mining UDF encapsulates

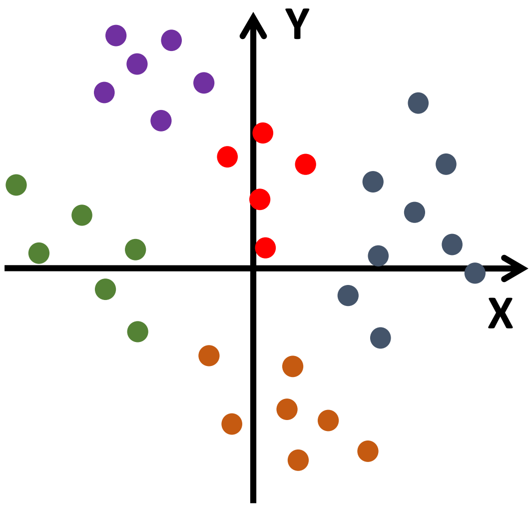
- … for Intel MIC (Many Integrated Core) platforms

Device / Feature	Intel Xeon X5680	**Intel Xeon Phi, Knights Corner** SE10X	**Intel Xeon Phi, Knights Landing** 7250
# of physical cores	6	**61**	**68**
Hyper threading factor	2	**4**	**4**
# of logical cores	12	**244**	**272**
Frequency, GHz	3.33	**1.1**	**1.4**
Vector processing unit	No	**512 bit**	**512 bit**
Bootable	Yes	**No**	**Yes**
Peak performance, TFLOPS	0.371	**1.076**	**3.046**

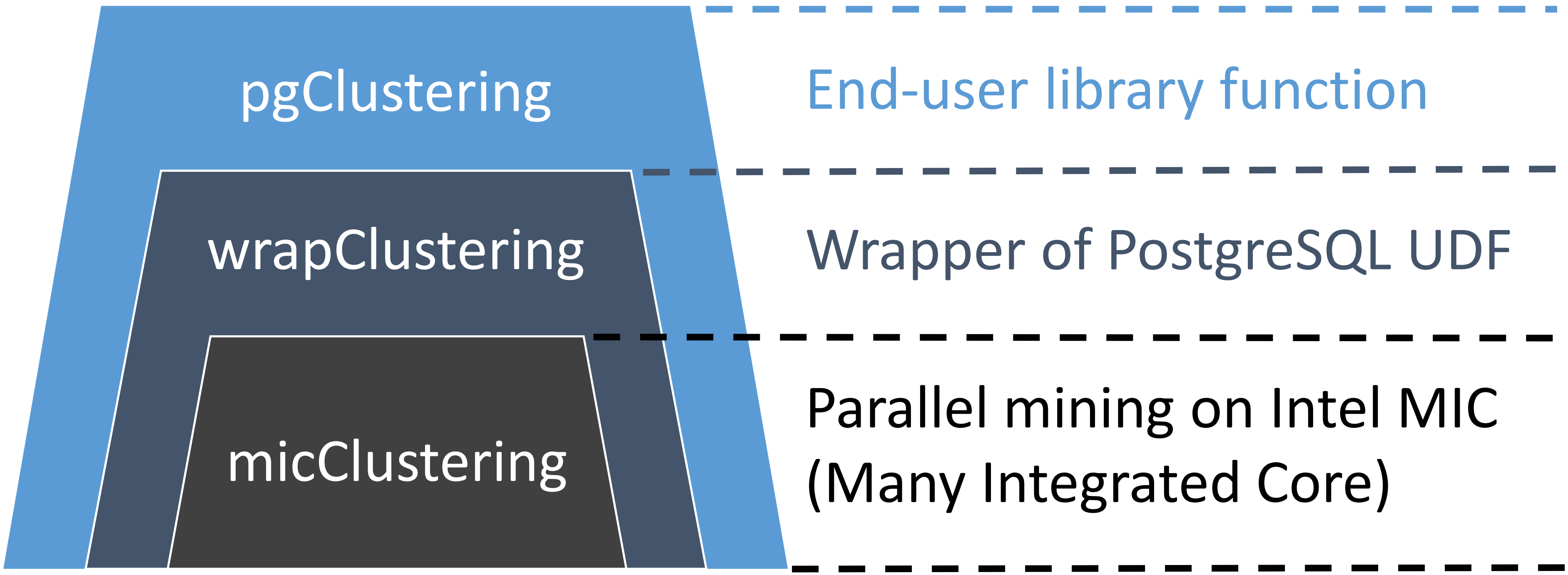

What mining UDF encapsulates

- Using cache of pre-computed mining structures
  - E.g. distance matrix to perform clustering

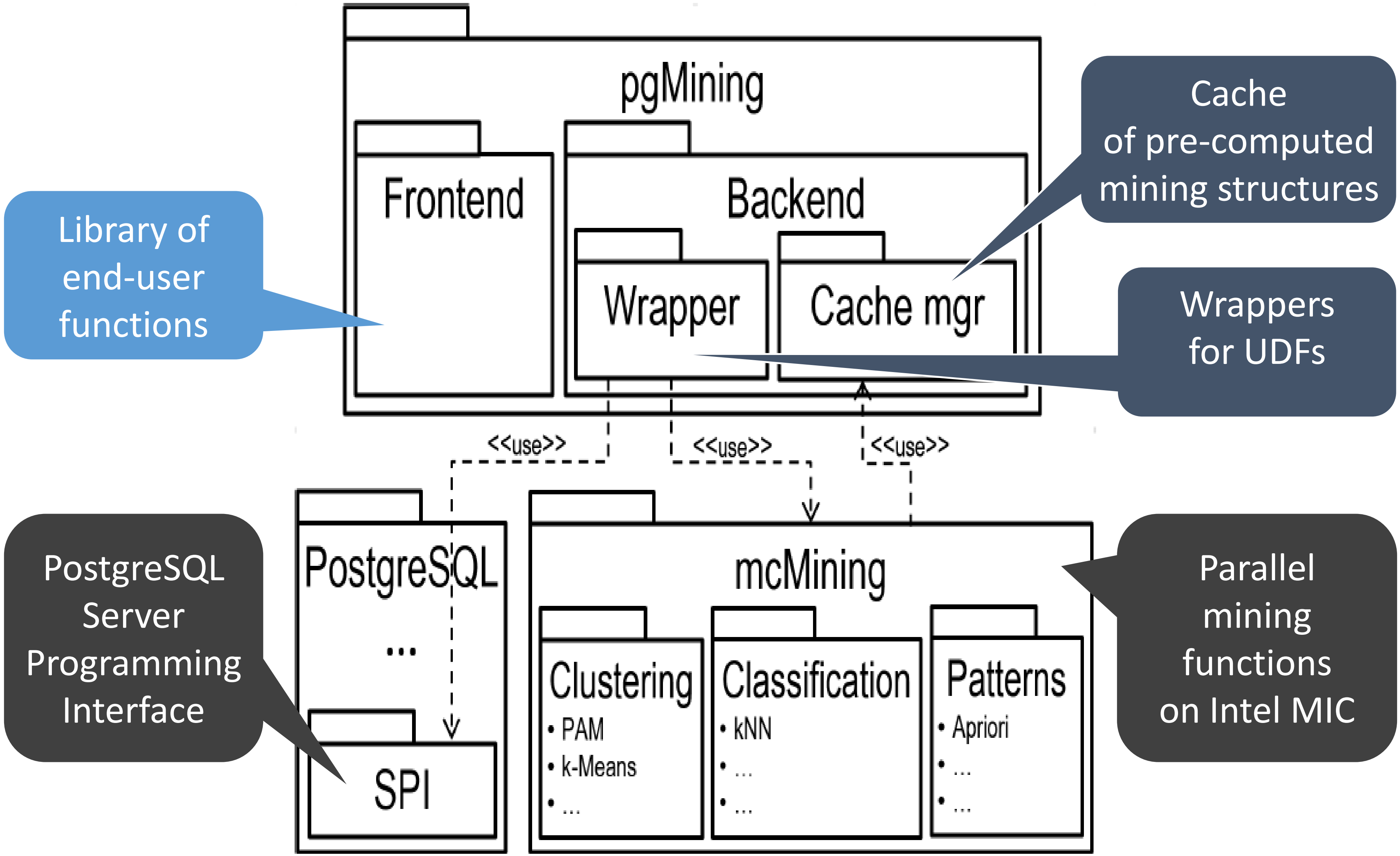
	0	1	...	*n*-1
0	0	dist(a_0, a_1)	...	dist(a_0, a_{n-1})
1		0	dist(a_1, a_j)	dist(a_1, a_{n-1})
...			0	dist(a_j, a_{n-1})
***n*-1**				0



How mining UDF is designed



Module structure



End-user library function

```
int pgClustering (
   PGconn * conn,     // ID of PostgreSQL connection
   char * inpTable,   // Name of input table
   int dim,           // Number of coordinates in data point
   int k,             // Number of clusters
   float Eps,         // Accuracy
   char * outTable)   // Name of output table
{
   // Register UDF
   PQexec (conn, "CREATE OR REPLACE FUNCTION
   wrapClustering (text, integer, integer, real) RETURNS text AS
       ' pgmining ', 'wrapClustering' LANGUAGE C STRICT;");
   // Create resulting table
   PQexec (conn, "CREATE %s TABLE IF NOT EXISTS (data text)",
              outTable);
   // Execute UDF
   return PQexec (conn, "INSERT INTO %s
       SELECT wrapClustering (%s, %d, %d, %f);",
              outTable, inpTable, dim, k, Eps);
}
```

Wrapper of PostgreSQL UDF


```
Datum wrapClustering(PG_FUNCTION_ARGS)
{
    // Extract input parameters of clustering
    // from the PostgreSQL parameters
        char * inpTable = text_to_cstring (PG_GETARG_TEXT_P (0));
        int dimension = PG_GETARG_INT32 (1);
        int k = PG_GETARG_INT32 (2);
        float Eps = PG_GETARG_FLOAT4 (3);
        int N;

    // Check if we have pre-calculated mining structures
    // in our cache, else calculate (in parallel) and load it
        void * distMatr = cache_getObject (strcat (inpTable, "_distMatrix" ));
        if (distMatr == NULL) {
                // Check if input table is in our cache
                void * inpData = cacheGet (inpTable);
                if (inpData == NULL) {
                        // Allocate memory and load input table to cache
                        inpData = (float4 *) palloc (dimension * sizeof (float4));
                        wrapTabRead (inpData , inpTable, dim, &N);
                        cachePut (inpTable, inpData, sizeof (inpData));
                }
                distMatr = micCalcDistMatr (inpData, dim, N);
                cache_putObject (strcat (inpTable, "_distMatrix"), distMatr, sizeof (distMatr));
        }

    // Cluster data (in parallel) using mining structures
    // and save results in output table
        micClusteringRes * outData = micClusteringResCreate ();
        micClustering (N, k, Eps , outData , distMatr);
        PG_RETURN_TEXT (data2String(outData));
}
```

Wrapper of PostgreSQL UDF

```
Datum wrapClustering(PG_FUNCTION_ARGS)
{
   // Extract input parameters of clustering
   // from the PostgreSQL parameters
   char * inpTable = text_to_cstring (PG_GETARG_TEXT_P (0));
   int dim = PG_GETARG_INT32 (1);
   int k = PG_GETARG_INT32 (2);
   float Eps = PG_GETARG_FLOAT4 (3);
   int N;

   // Check if we can use pre-calculated mining structures
   void * distMatr = cacheGet(strcat (inpTable , "_distMatr" ));
   if (distMatr == NULL) {
      // Check if input table is in our cache
      void * inpData = cacheGet (inpTable);
      if (inpData == NULL) {
         // Allocate memory and load input table to cache
         inpData = (float4 *) palloc (dim * sizeof (float4));
         wrapTabRead (inpData, inpTable, dim, &N);
         cachePut(inpTable, inpData, sizeof (inpData));
      }
      distMatr = micCalcDistMatr (inpData, dim, N);
      cachePut (strcat (inpTable, "_distMatr"), distMatr, sizeof (distMatr));
   }
   micClustering_res * outData = micClustering_resCreate ();
   micClustering (N, k, Eps, outData , distMatr); // Perform clustering
   PG_RETURN_TEXT (data2String(outData)); // Write results to the output table
}
```

Wrapper of PostgreSQL UDF

```
Datum wrapClustering(PG_FUNCTION_ARGS)
{
    char * inpTable = text_to_cstring (PG_GETARG_TEXT_P (0));
    int dim = PG_GETARG_INT32 (1);
    int k = PG_GETARG_INT32 (2);
    float Eps = PG_GETARG_FLOAT4 (3);
    int N;
    // Check if we can use pre-calculated distance matrix
    void * distMatr = cacheGet (strcat (inpTable, "_distMatr"));
    if (distMatr == NULL) {
       // Check if input table is in our cache and load if not
       void * inpData = cacheGet (inpTable);
       if (inpData == NULL) {
           inpData = (float4 *) palloc (dim * sizeof (float4));
           wrapTabRead (inpData, inpTable, dim, &N);
           cachePut (inpTable, inpData, sizeof (inpData));
        }
       // Calculate distance matrix in parallel and load it to cache
       distMatr = micCalcDistMatr (inpData, dim, N);
       cachePut (strcat (inpTable, "_distMatr"), distMatr,
                 sizeof (distMatr));
    }
    micClusteringRes * outData = micClusteringResCreate ();
    micClustering (N, k, Eps , outData , distMatr); // Perform clustering
    PG_RETURN_TEXT (data2String(outData)); // Write results to the output table
}
```

Wrapper of PostgreSQL UDF

```
Datum wrapClustering(PG_FUNCTION_ARGS)
{
    // Extract input parameters
    char * inpTable = text_to_cstring (PG_GETARG_TEXT_P (0));
    int dim = PG_GETARG_INT32 (1);
    int k = PG_GETARG_INT32 (2);
    float Eps = PG_GETARG_FLOAT4 (3);
    int N;
    // Check if we can use pre-calculated distance matrix
    void * distMatr = cacheGet (strcat (inpTable, "_distMatr"));
    if (distMatr == NULL) {
       // Check if input table is in our cache and load if not
       void * inpData = cacheGet (inpTable);
       if (inpData == NULL) {
            inpData = (float4 *) palloc (dim * sizeof (float4));
            wrapTabRead (inpData, inpTable, dim, &N);
            cachePut (inpTable, inpData, sizeof (inpData));
        }
       // Calculate distance matrix in parallel and load it to cache
       distMatr = micCalcDistMatr (inpData, dim, N);
       cachePut (strcat (inpTable, "_distMatr"), distMatr,
                    sizeof (distMatr));
    }

    // Cluster data (in parallel) using mining structures
    micClusteringRes * outData = micClusteringResCreate ();
    micClustering (N, k, Eps, outData, distMatr);
    // Save results in output table
    PG_RETURN_TEXT (data2String(outData));
}
```

Methods of Cache manager

- `void * cacheGet(char * objName)`

  // Searches for a specified object in the cache.

  // Updates internal statistics of the mined object
  // (number of calls, timestamp of recent call, etc.).

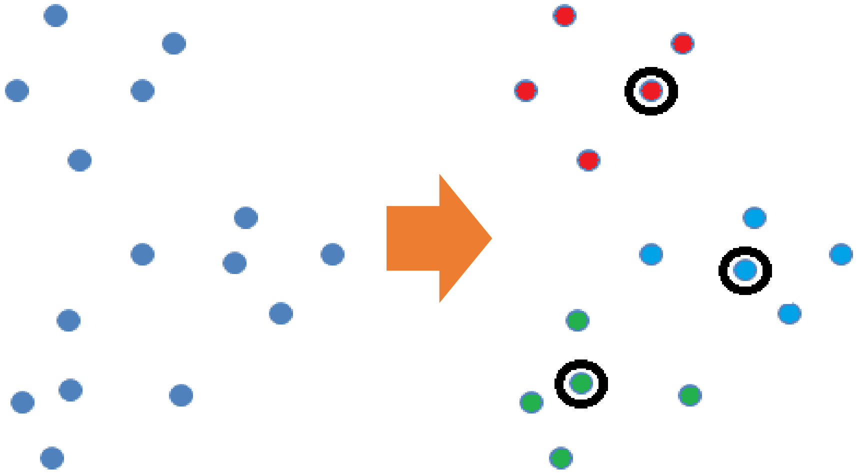
- `int cachePut(char * objName)`

  // Loads a specified object into the cache.

  // Pops out a victim object if it is not enough space in the cache
  // (according to a cache management policy,
  // e.g. Least Recently Used, Least Frequently Used, etc.).

PAM: Partition Around Medoids

- Goal
  - Organize *n* objects of data set in *k* clusters
- Key idea
  - Cluster centers are chosen as objects of data set (*medoids*)
- Method
  - BUILD phase
    - Initially choose cluster centers
  - SWAP phase
    - Iteratively move objects across clusters to improve value of an objective function



Parallel PAM: basic optimizations

```
void calcDistMatr (const float* rowData, const float* colData,
        float* distances, const int n, const int pointWidth){
  const int vecLen = 32;
  #pragma omp parallel
  {
    float point[pointWidth] __attribute__((aligned(64)));
    float result[vecLen]    __attribute__((aligned(64)));
    #pragma omp for
    for(int i=0; i<n; ++i){
      point[:]=rowData[i*pointWidth:pointWidth];
      for(int ii = 0; ii < n; ii += vecLen){
        result[:]=0;
        for(int j=0; j < pointWidth; ++j){
          const float* restrict point2 = colData+ii*pointWidth;
          result[:]+=(point[j]-point2[j*vecLen:vecLen])*
                     (point[j]-point2[j*vecLen:vecLen]);
        }
        distances[i*n+ii:vecLen] = sqrtf(result[:]);
      }
    }
  }
}
```

OpenMP
Rewriting loops
Loop tiling
Data alignment

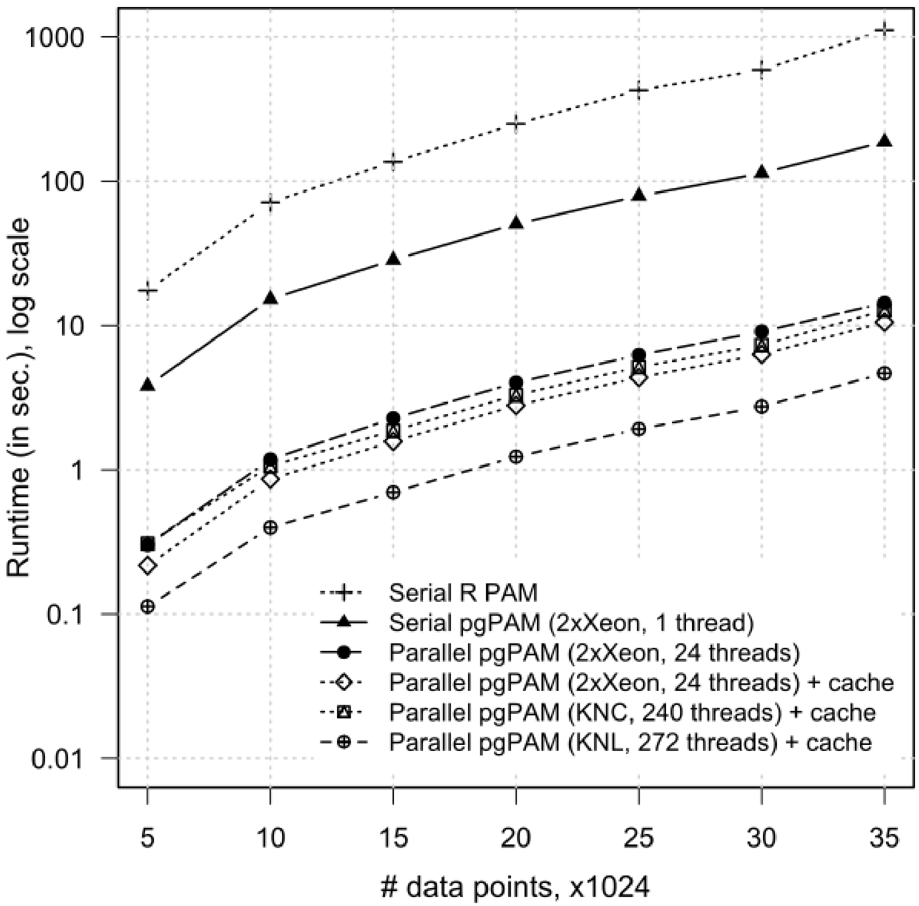
Experimental evaluation

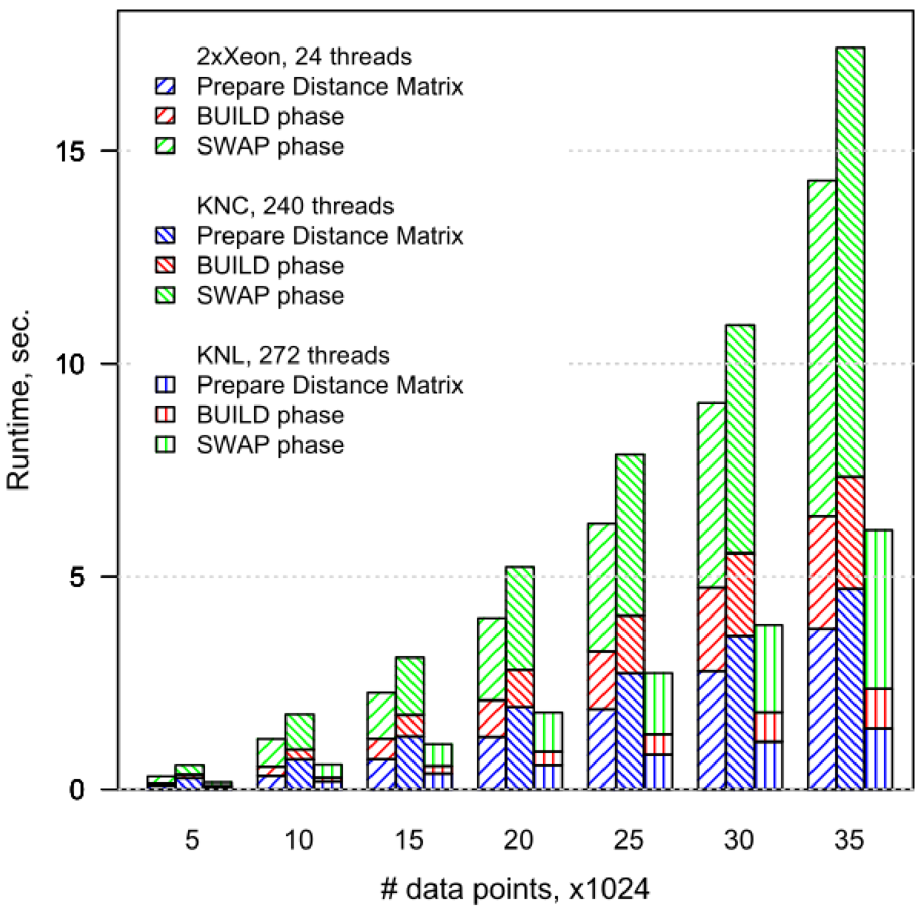
- Clustering algorithm: PAM (Partition Around Medoids)
  - Represents cluster centers as points of input data set (*medoids*)
  - *CALCULATION* of distance matrix
  - *BUILD phase*: initial clustering by the successive selection of medoids
  - *SWAP phase*: improving clustering according to an objective function
- Datasets

Name	dim	# of clusters	# of points, $\times 2^{10}$	Semantic
Census	67	10	35	Population surveys by the US Census Bureau
MixSim	5	10	35	Generator of synthetic datasets for evaluation of clustering algorithms
Power	3	10	35	Individual household electricity consumption
FCS Human	423	10	18	Aggregated human gene information

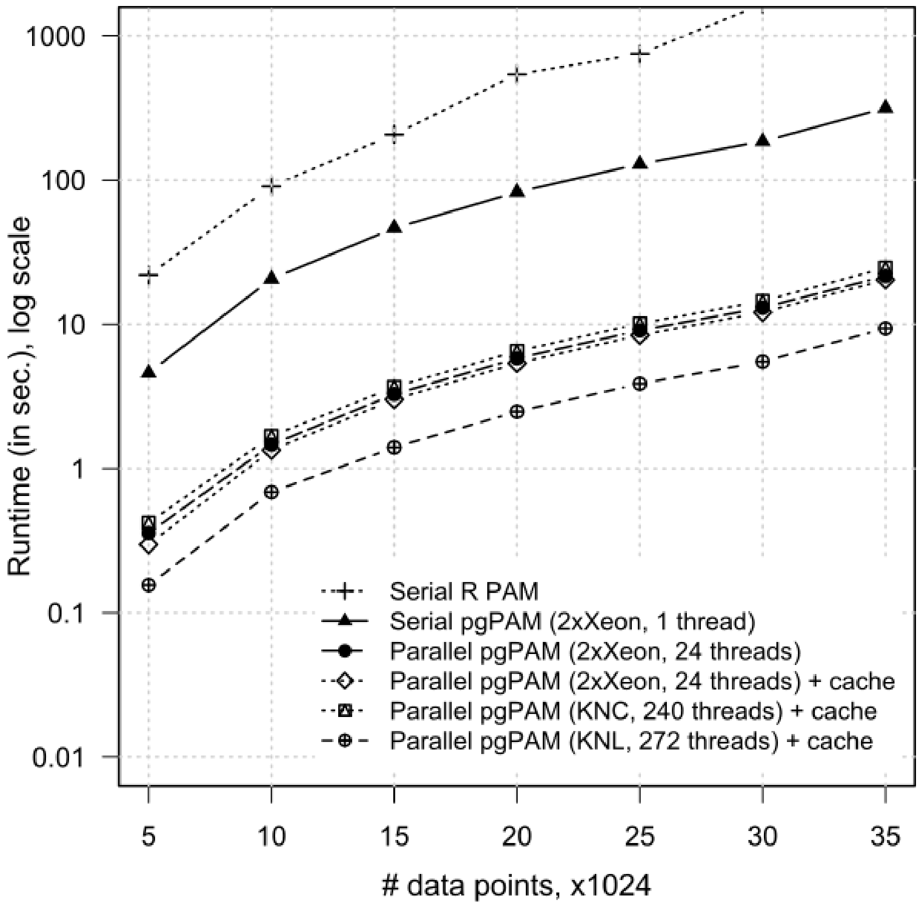
- Competitor: PAM algorithm of *R* data mining package

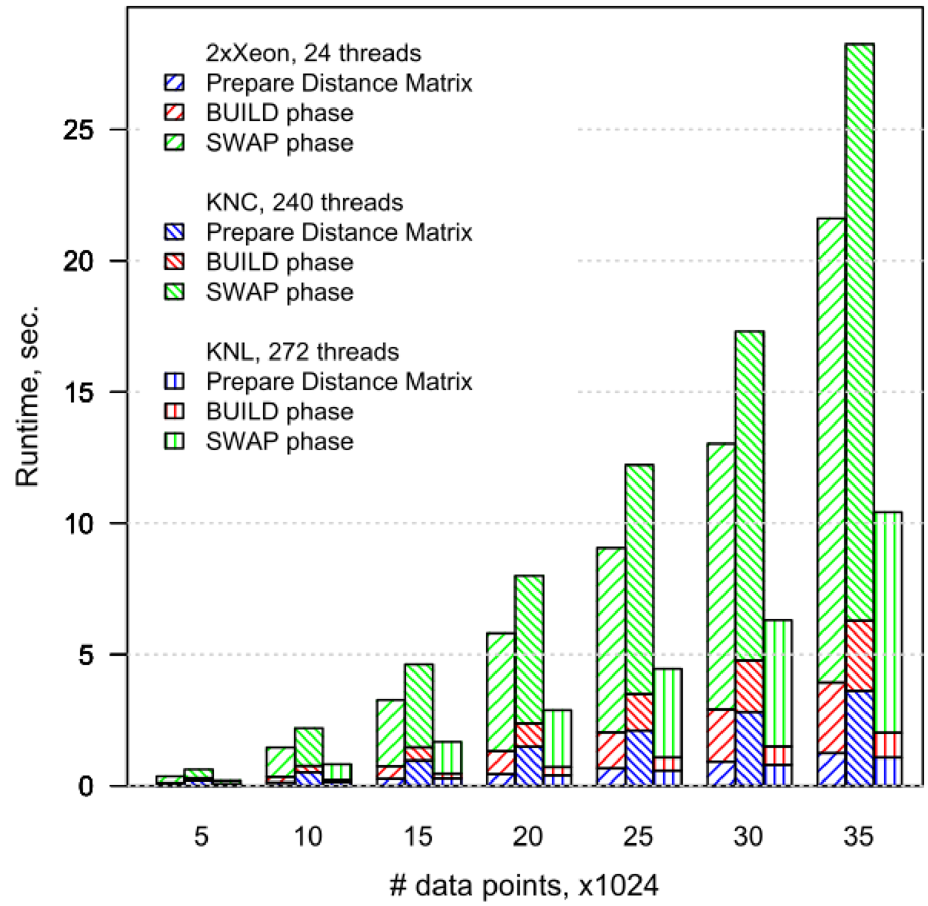
Performance: Census dataset



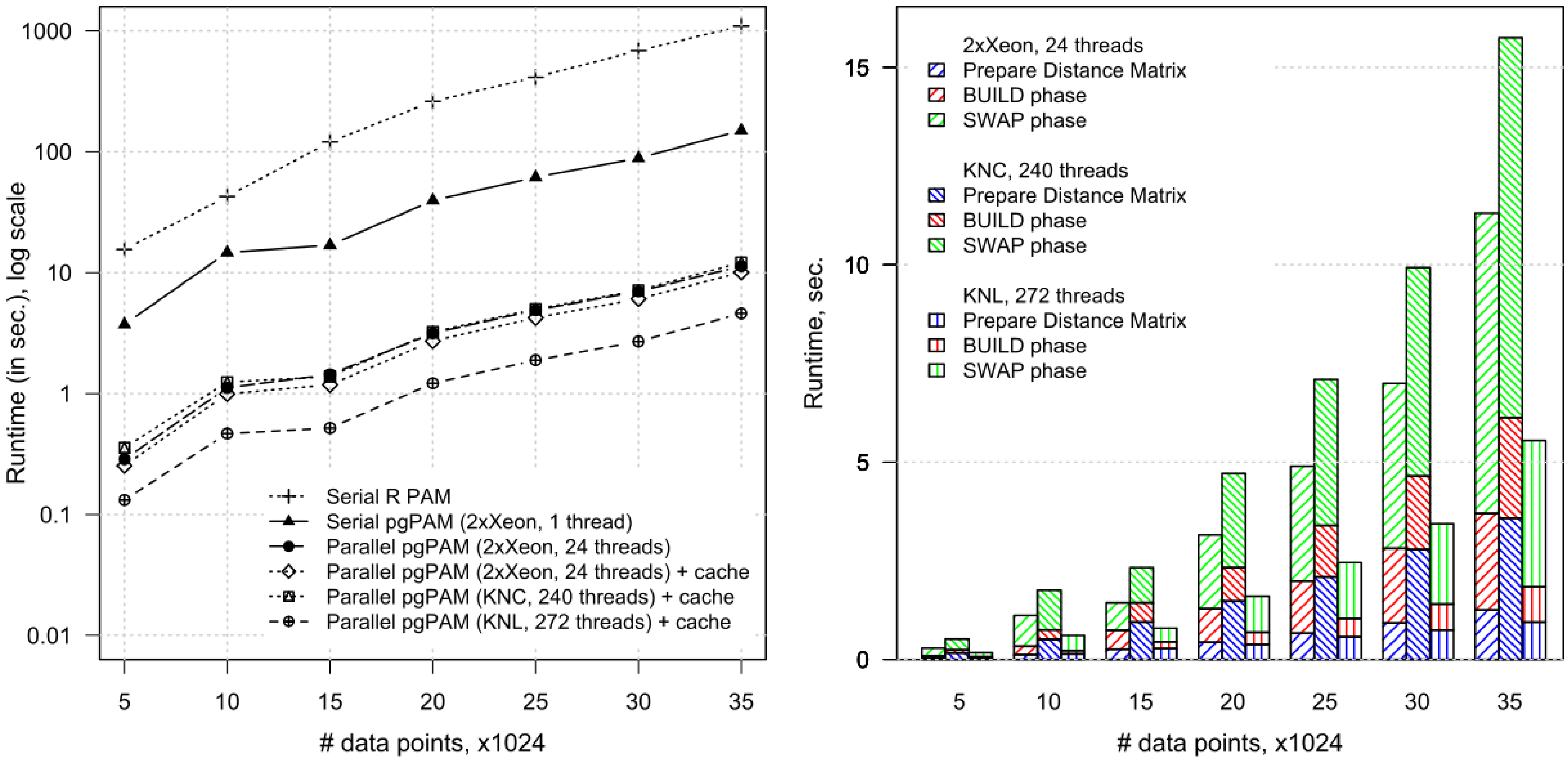


Performance: MixSim dataset

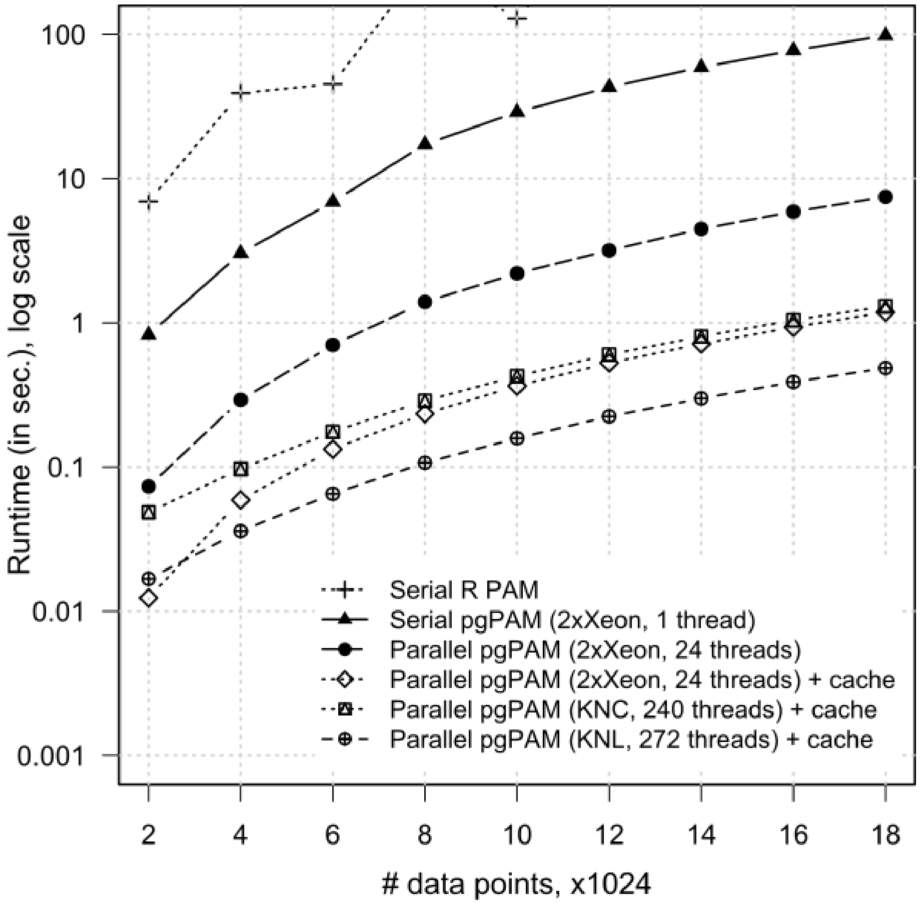


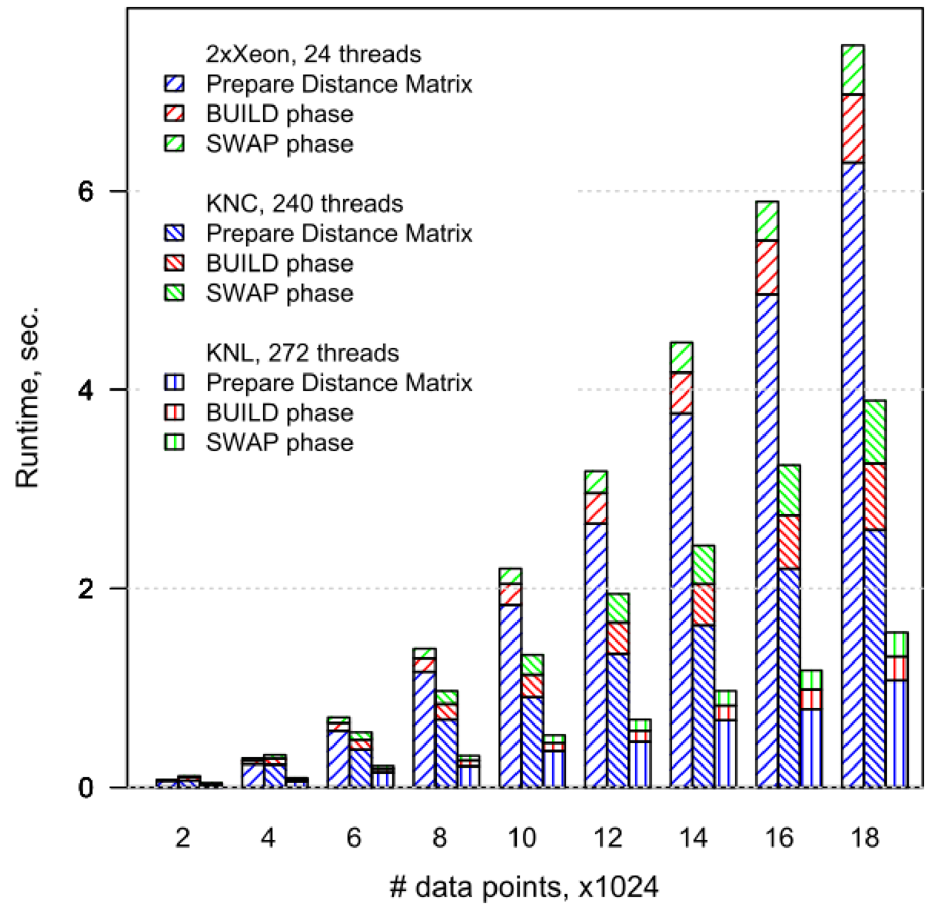


Performance: Power dataset

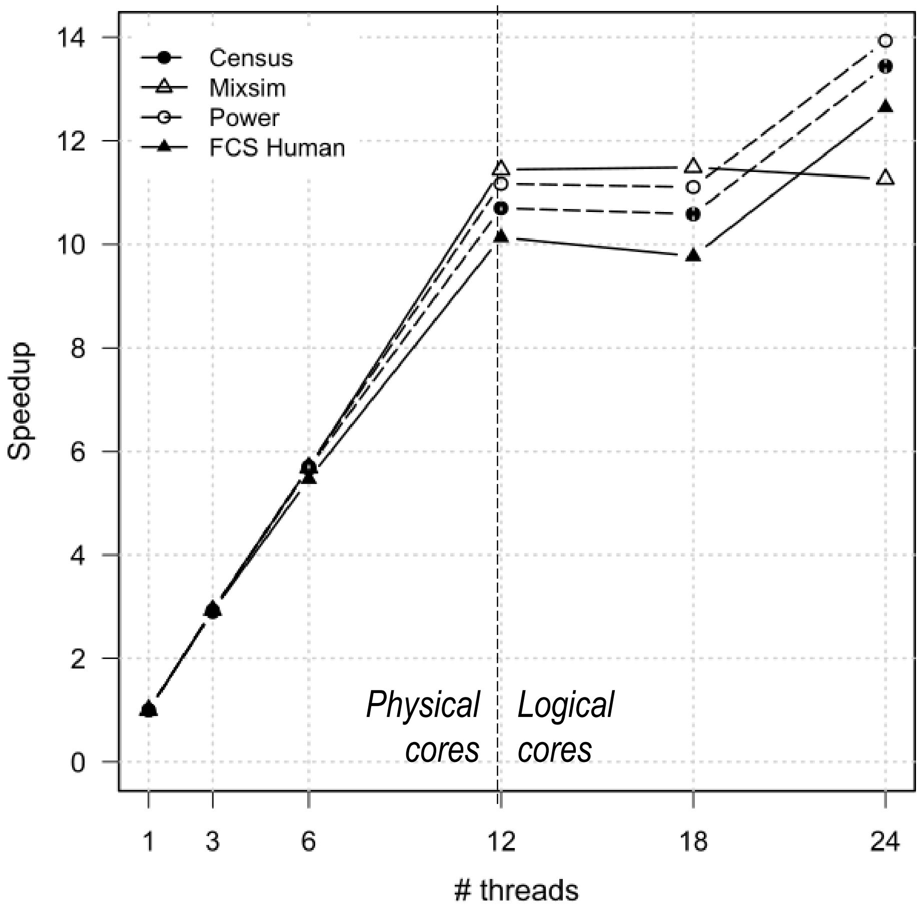


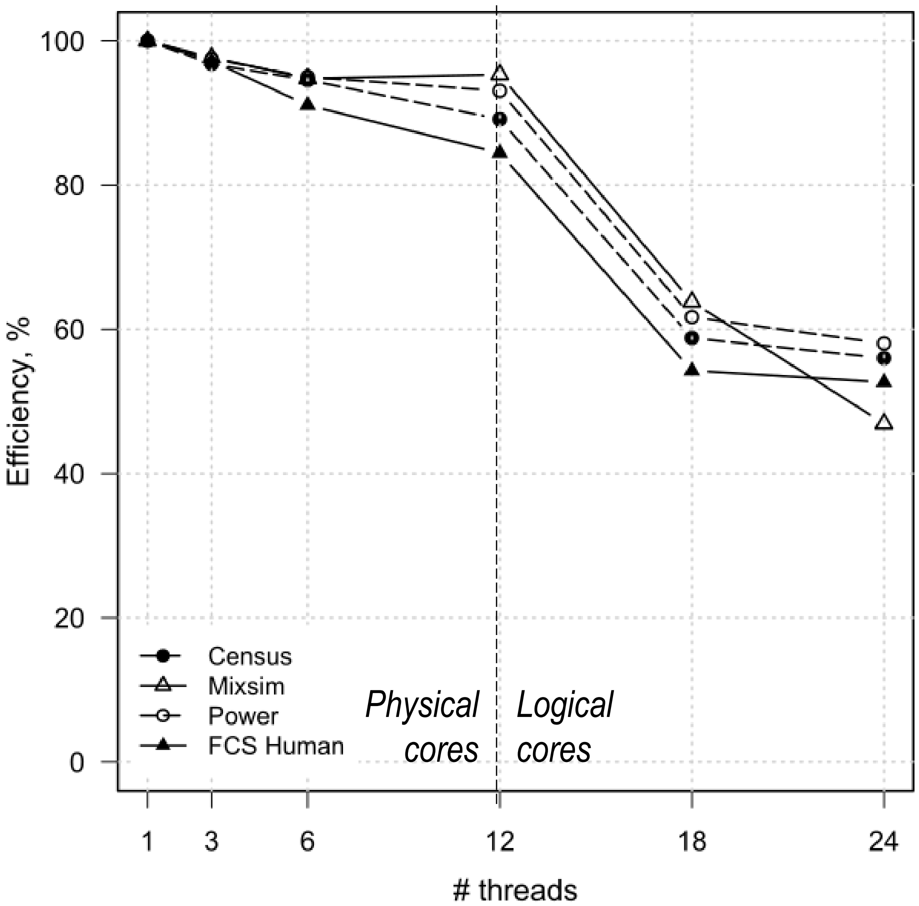
Performance: FCS Human dataset



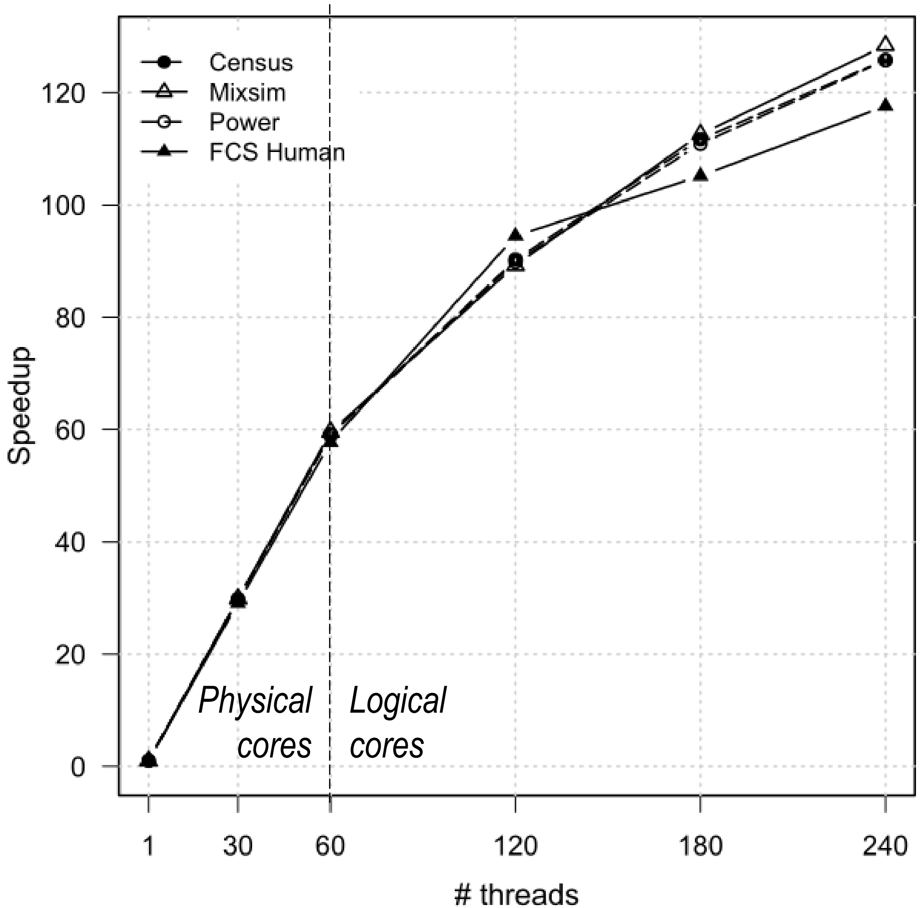


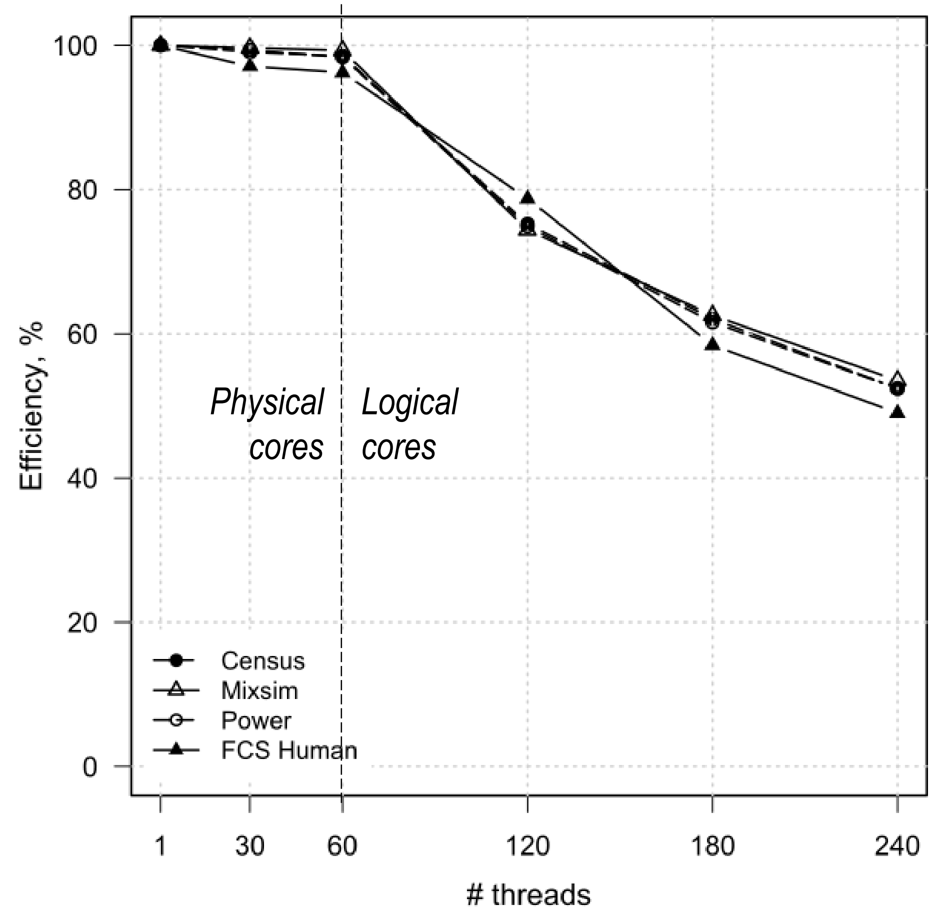
Speedup and Efficiency: 2×Xeon, 24 threads



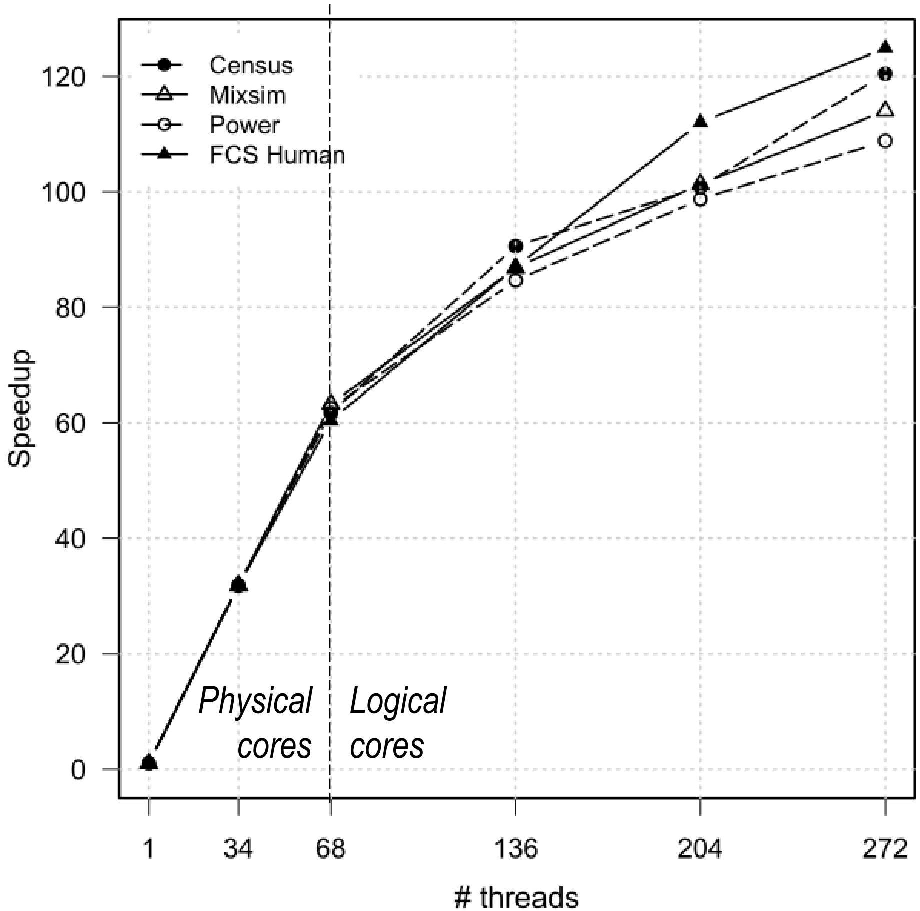


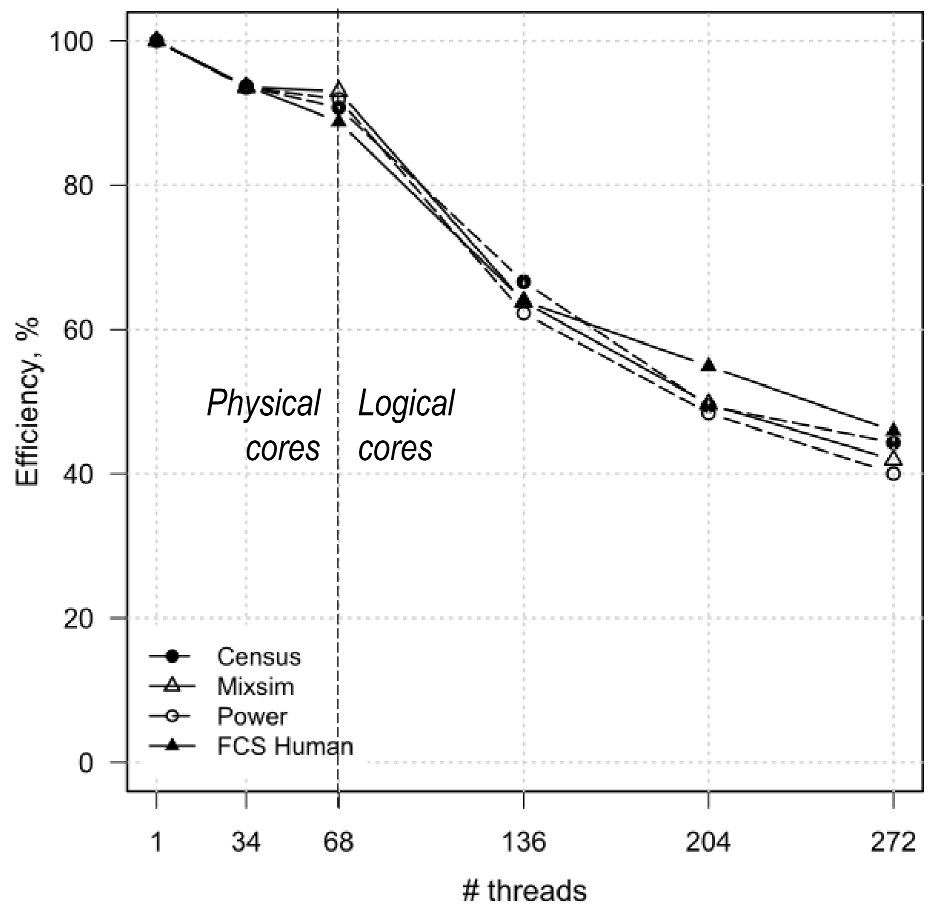
Speedup and Efficiency: KNC, 240 threads





Speedup and Efficiency: KNL, 272 threads





Conclusion

- We have proposed an approach to data mining inside DBMS based on parallel implementation of UDFs for many-core platforms
- We have implemented the approach for PostgreSQL, Intel Xeon Phi (KNC and KNL), and PAM clustering algorithm
- We have conducted experiments on synthetic and real data sets that show good scalability of our approach and overtaking analogue (*R* data mining package).

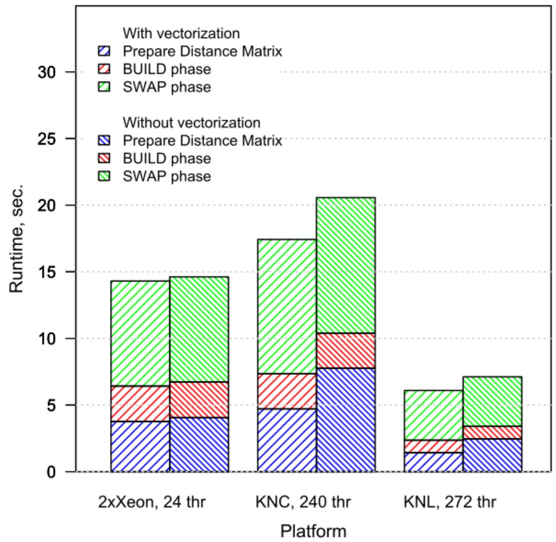
Thank you for paying attention! Questions?

Mikhail Zymbler

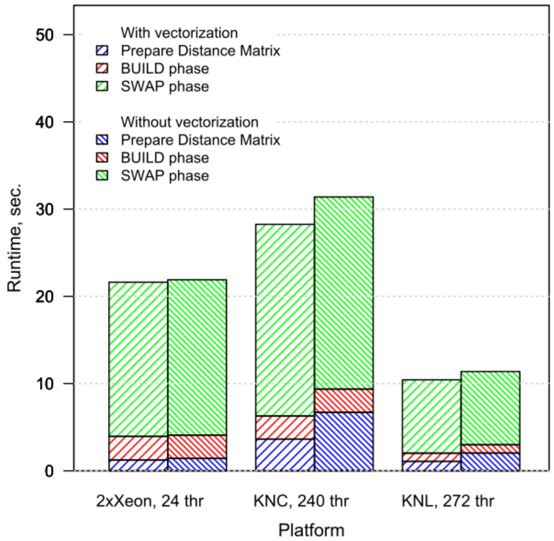
mzym@susu.ru

Impact of vectorization

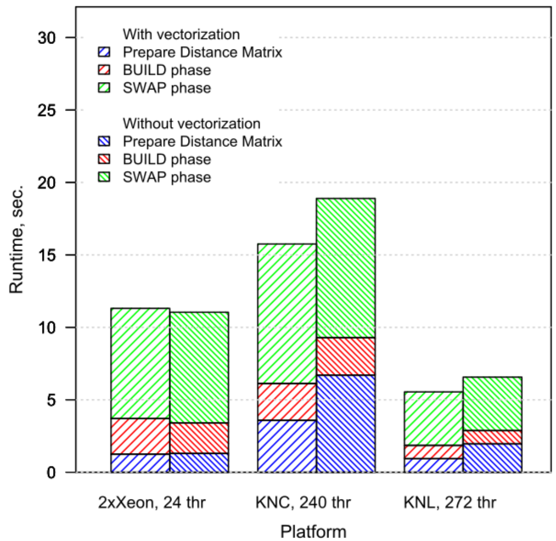
Census



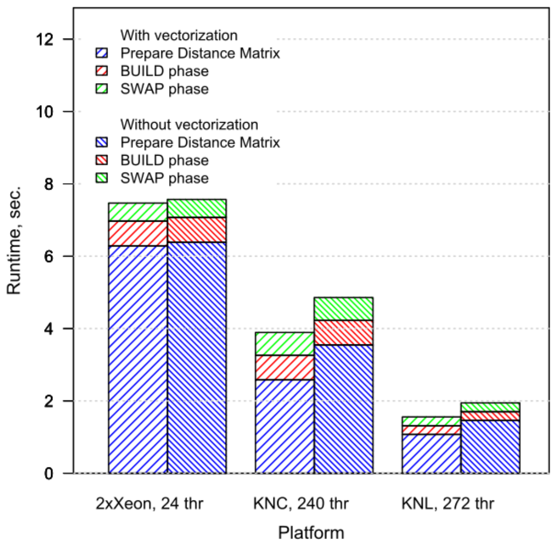
MixSim



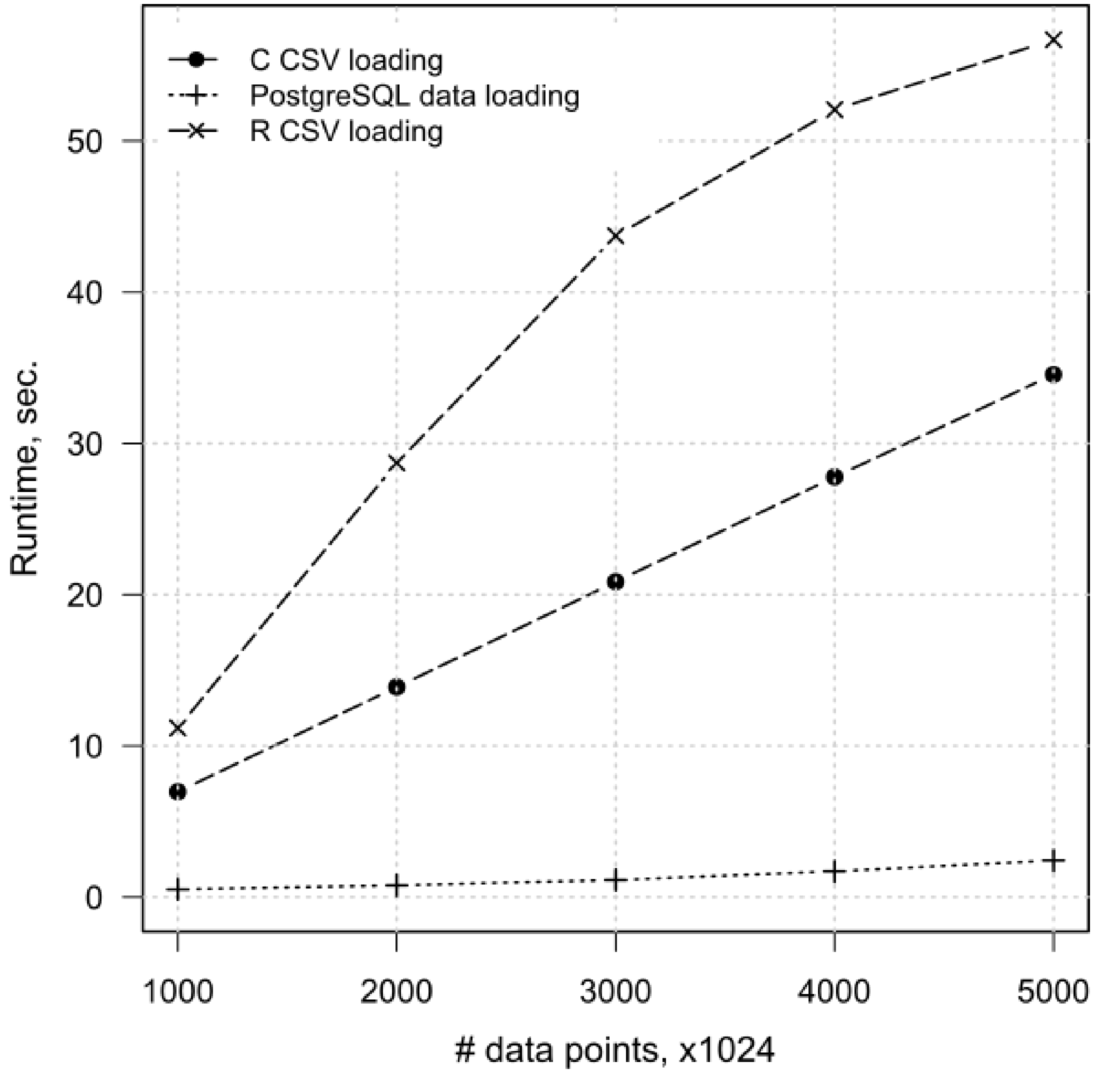
Power



FCS Human



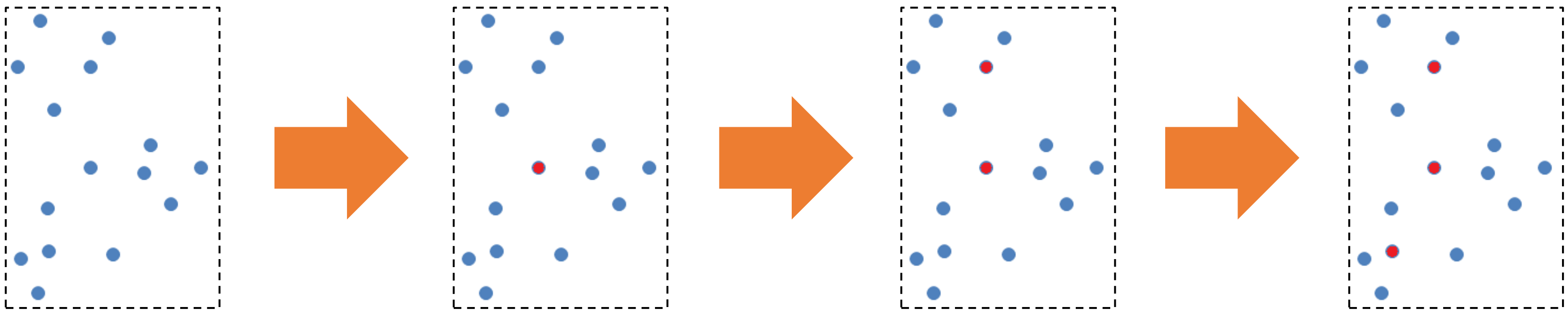
Performance of load data into RAM



PAM: Partition Around Medoids

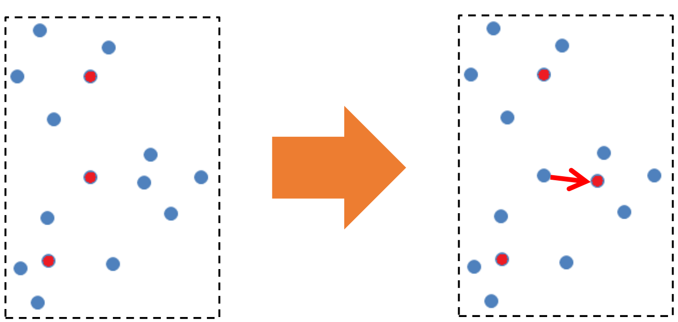
- Objective function $E = \sum_{j=1}^{n} \min_{1 \le i \le k} dist(c_i, o_j)$

  where c_i is the medoid, o_j is the clustered object, *dist* is the distance metric
- BUILD phase: complexity is $O(kn^2)$



$E=\infty$ $E= 2.250$ $E= 1.578$ $E= 1.014$

- SWAP phase: complexity is $O(k(n - k)^2)$ per iteration



$E= 1.014$ $E= 0.865$

PAM: pseudocode

```
Input: set of objects O, # of clusters k
Output: set of clusters C

// Calculation of distance matrix
distMatr ← calcDistMatr (O)
// BUILD phase
C ← BuildMedoids (distMatr)
repeat // SWAP phase
      T_min ← findBestSwap (M, C)
      Swap (c_min, o_min) for T_min
until T_min < 0
```

Rewriting loops: an example

- Vector processor unit is of 512 bits
  - 16 float elements or 8 double elements
- Rewriting loops provides vectorization of computations (SIMD, Single Instruction Multiple Data)

Scalar loop

```
for(i = 0; i < n; ++i)
    a[i] = b[i] + c[i];
```

SIMD loop

```
#ifdef DOUBLE_PRECISION
  #define LEN 8
#else
  #define LEN 16
#endif
for(i = 0; i < n; i += LEN)
    a[i:LEN] = b[i:LEN] + c[i:LEN];
```

Loop tiling: an example

Without tiling

x_1	y_1	z_1

x_1	y_1	z_1	x_2	y_2	z_2	…	x_{32}	y_{32}	z_{32}
x_{33}	y_{33}	z_{33}	x_{34}	y_{34}	z_{34}	…	x_{64}	y_{64}	z_{64}

With tiling

x_1	y_1	z_1

x_1	x_2	…	x_{32}	y_1	y_2	…	y_{32}	z_1	z_2	…	z_{32}
x_{33}	x_{34}	…	x_{64}	y_{33}	y_{34}	…	y_{64}	z_{33}	z_{34}	…	z_{64}