Data Mining Algorithms Parallelizing in Functional Programming Language for Execution in Cluster

Ivan Holod, PhD, Assoc. Prof.,
iiholod@mail.ru

Aleksey Malov, PhD,
alexeimal-2@yandex.ru

Sergey Rodionov, PhD,
sv-rodion@mail.ru

Saint Petersburg Electrotechnical University

Motivation

• Imperative programming languages (Java, C / C ++, Fortran and others.) are not suitable for parallel execution.

• Existing expansion of imperative languages for parallel execution (Ada, High Performance FORTRAN, High Performance C ++ and others.) allow to parallelize only individual structures such as cycles.

• Alternative imperative languages are functional languages (Lisp, Haskell et al.).

We propose to use functional programming language to reduce the efforts for parallelization of data mining algorithms.

Related works

- Existing algorithm implementations in functional programming languages do not suggest parallel execution.
- Algorithms were implemented by imperative programming languages:

Constructing parallel data mining algorithms

- Universal algorithm parallelizing
  - NIMBLE system
  - Multicore map-reduce framework
- Individual algorithm parallelizing
  - Decision tree: MWK, SUBTREE, SLIQ, SPRINT, DP, PDT
  - Association: CCPD, PCCD, APM, HPA, SPA, PE, PME, PC,PMC
  - Clustering: DIB, Collaborative, Pkmeans, MAFIA, P-CLUSTER

Conception

The algorithm can be built from separated blocks if these blocks have the following features:

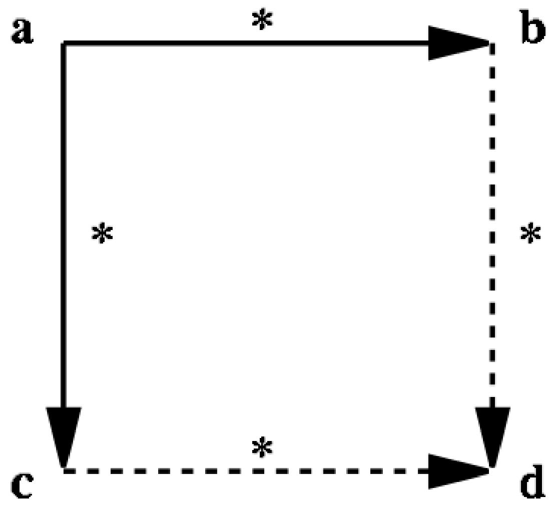
- **they are interchangeable** – **have unify interface**
- **they are executed in arbitrary order** – **have features of pure function**

The features of pure function:

- the function always produces the same result given the same argument(s).
- calculation of the result does not cause any semantically observable side effect.

Church–Rosser theorem

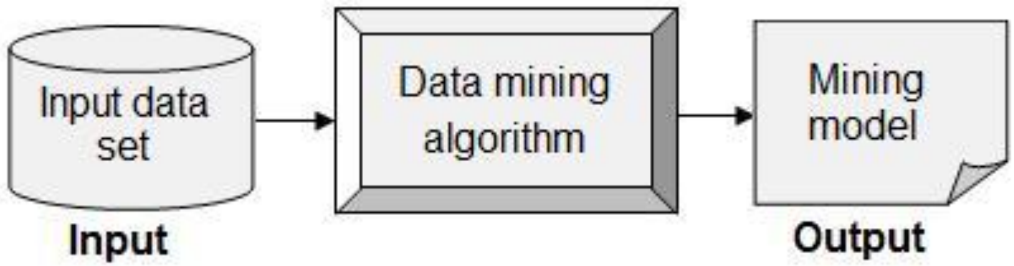
When applying reduction rules to terms in the λ-calculus, the ordering in which the reductions are chosen does not make a difference for the final result.



Pure functions can be executed in any order (include parallel order).

Functional block is part of algorithm that is the pure function with unified interface.

Data mining algorithm as functional expression



(defun <*algorithm_function_name*> (***d***)

$(\mathrm{fb}_n\ \boldsymbol{d}\ (\mathrm{fb}_{n-1}\ \boldsymbol{d}\ \dots\ (\mathrm{fb}_i\ \boldsymbol{d}\ \dots\ (\mathrm{fb}_1\ \boldsymbol{d}\ \mathrm{nil})\dots)\ \dots))$

(defun <*function_name*> (***d m***))

where:

- (defstruct ***d*** attr_list vectors_list)
- (defstruct ***m*** state rules_list)

Conditional function

(defun condition_function (*d m cf* fb_t fb_f)
(cond ((cf d m) (fb_t d m)) (T (fb_f d m)))

Example:

```
(defun is_curr_attr_target (d m)
   (condition_function
            d
            m
        (eql                                              ; cf function
            (nth m-state-curr_attr d-attr_list)
            m-state-targed_attr)
        m                                                 ; fb_t function
        for_all_vectors_cycle))                           ; fb_f function
```

Cycle function

```
(defun cycle_function (d m cf fb_init fb_pre fb_iter)
                        (cycle( d (fb_init d m) cf fb_pre fb_iter)))
(defun cycle (d m cf fb_pre fb_iter)
     (cond ((cf d m)
               (cycle d (fb_iter d (fb_pre d m)) cf fb_pre fb_iter))
      (T (fb_iter  d (fb_pre d m)))))))
```

Example:

```
(defun for_all_attributes_cycle (d m)
  (cycle_function d m
                    (eql nil d-attr_list)                ;cf function
                     m                                   ;fb_init function
                     cdr d-attr_list                     ;fb_pre function
                    inc_count_attribute d m))            ;fb_iter function
```

Function for parallelization of algorithm

```
(funcname mpi_parallel_function (d m merge split fb)
          (mpi:mpi-init) ;initialize MPI
          (merge d (split d m fb))
                    (mpi:mpi-finalize))
```

Example:

```
(defun mpi_vector_parall (d m)
    mpi_parallel_function (d m
      mpi_merge_vector_parall ;the merge function
      mpi_split_vector_parall ;the split function
      for_all_classes_cycle)) ;the fb function
```

Naive Bayes algorithm

```
for all attributes a
  if a is not target attribute
    for all vectors w
      increment count of vectors
    end for all vectors;
  end if
end for all attributes;

for all classes c
    for all vectors w
      increment count of vectors for
                 class of vector w;
    end for all vectors
end for all classes c
```

```
(defun NBAlgorithm (d, nil)
     for_all_classes_cycle ( d
        for_all_attributes_cycle (d, nil)))

(defun  for_all_classes_cycle  (d m)
   (cycle_function d m
       (eql nil d-cls_list)    m cdr d-cls_list
for_all_vectors_cycle (d m))

(defun for_all_attributes_cycle (d m)
   (cycle_function d m
       (eql nil d-attr_list)   m cdr d-attr_list
for_all_vectors_cycle (d m))

(defun for_all_vectors_cycle (d m)
   (cycle_function d m
      (eql nil d-vectors_list)      m cdr d-vectors_list
              inc_count_vec (d m))
```

Performing parallel processing

Parallel processing of **vectors**

```
(defun NBAlgorithmVectorsParallel (d)
  mpi_vector_parall (d
    for_all_classes_cycle ( d
      for_all_attributes_cycle (d))))
```

```
(defun mpi_vector_parall (d m)
    mpi_parallel_function (d m
      mpi_merge_vector_parall
      mpi_split_vector_parall
      for_all_classes_cycle))
```

Parallel processing of **attributes**

```
(defun NBAlgorithmAttrsParallel (d)
  for_all_classes_cycle ( d
    mpi_attr_parall (d
      for_all_attributes_cycle (d))))
```

```
(defun mpi_attr_parall (d m)
    mpi_parallel_function (d m
      mpi_merge_attr_parall
      mpi_split_attr_parall
      for_all_attributes_cycle))
```

Experimental results (sec)

Algorithm	W1	W5	W10	A1	A5	A10	C1	C5	C10
Number of vectors	**10 000**	**50 000**	**100 000**	100	100	100	1 000	1 000	1 000
Number of attributes	10	10	10	**100**	**500**	**1 000**	100	100	100
Avg. number of classes	5	5	5	100	100	100	**100**	**500**	**1000**

Algorithm	Cores	W1	W5	W10	A1	A5	A10	C1	C5	C10
NBAlgorithm	1	0.09	0.45	0.87	0.51	10.44	40.00	0.55	2.61	5.20
NBAlgorithm VectorsParallel	2	**0.05**	**0.24**	**0.45**	0.28	5.33	20.51	0.28	1.33	2.67
	4	**0.03**	**0.13**	**0.28**	0.19	3.39	12.30	0.17	0.74	1.50
NBAlgorithm AttrsParallel	2	0.05	0.25	0.48	**0.28**	**5.27**	**20.34**	0.27	1.32	2.65
	4	0.04	0.19	0.41	**0.16**	**2.96**	**11.47**	0.17	0.74	1.49

Conclusion

- The proposed approach uses the functional programming language for implementation of data mining algorithms as a functional expression from the functional blocks.
- It helps to create new algorithms, including parallel forms, from existing blocks with less efforts or modify the existing algorithms by replacing separate blocks.
- We implemented the Naïve Bayes algorithm and two its parallel forms: parallelizing by vectors and parallelizing by attributes .
- An experimental comparison of these versions showed their effectiveness for the data sets with different parameters.

Thank you for attention

Ivan Holod, iiholod@mail.ru
Aleksey Malov, alexeimal-2@yandex.ru
Sergey Rodionov, sv-rodion@mail.ru