Some Aspects of Functional Programming Languages Application in the Parallelization Problems

Orest Geiko
Department of Computer Science
Vasyl Stefanyk Precarpathian
National University
Ivano-Frankivsk, Ukraine
ifgo69@gmail.com

Artur Martsinkovskyi
Department of Computer Science
Vasyl Stefanyk Precarpathian
National University
Ivano-Frankivsk, Ukraine
arthurmarz.learn@gmail.com

Деякі Аспекти Використання Функціональних Мов Програмування в Задачах Паралелізації

Орест Гейко
кафедра інформатики
Прикарпатський національний університет
імені Василя Стефаника
Івано-Франківськ, Україна
ifgo69@gmail.com

Артур Марцінковський
кафедра інформатики
Прикарпатський національний університет
імені Василя Стефаника
Івано-Франківськ, Україна
arthurmarz.learn@gmail.com

***Abstract*—This article is a consideration of some aspects of functional programming, that are used for parallel computation and creation of asynchronous applications on the basis of referential transparency, pure functions; persistent data structures and data immutability. This is a review of features that functional algoirthms like Map/Reduce posess in the usecases of the parallel data processing of huge datasets.**

***Анотація*—В статті розглянуто деякі аспекти функціонального програмування, які використовуються для паралелізації обчислень та формування асинхронних додатків на основі прозорості посилань, чистих функцій; персистентних структури даних та іммутабельності даних. Розглянуто особливості функціональних алгоритмів для паралельної обробки гігантських наборів даних, таких як Map/Reduce.**

Keywords—distributed programming, functional programming, execution in parallel.

Ключові слова—дистрибутоване програмування, функціональне програмування, паралелізм.

I. INTRODUCTION

From the 80-s years of XX-th century the orders of growth of speed of the hardware like processors and RAM[1] of information systems have allowed us to use substandard software that lacked optimisation. The flaws of software were neglected due to the fast growth of the computational power. The scale of data processed was not very huge. the nature of the tasks that computers had to perform was far from complex either permissible in the means of time of execution. Recent development and comprehensive nature of computer networks caused rapid change in the scale of data that needed to be transmitted, stored and processed. At the same time, orders of growth for capabilities of computers are not as high as the need for them, that requires multithreading and clasterization efforts to be able to process data streams coming in.[2] This makes parallel execution a requirement for any modern system. Parallel execution can be regarded as a powerful tool for making data processing more efficient, but it also puts some additional requirements and brings up issues that have never been a problem in sequential programming. These problems mostly consist of modification and access issues that can come to life in any system that processes and stores data in parallel. Functional programming and algorithms are the efficient way to handle such issues.

II. RESEARCH RELEVANCE

Usual imperative paradigm, as well as object-oriented paradigm is not very suitable for the distributed data processing. Programs in these paradigms are sequential and have side effects, that can affect on the process of execution unexpectedly. On the other hand, functional programs when composed in the right fashion, can make paralLelization much easier. There is a multitude of features that are unique to the functional paradigm: immutability of data, pure functions, referential transparency, recursive algorithms, functors, convolutions and higher order functions, persistent data

structures. All of these elements of functional programming let us to avoid some classes of errors that are usual for parallel execution, like deadlocks, data corruption on simultaneous write, race conditions. Also, functional algorithms are better at scaling and distribution amongst multiple computers. The main elements of functional programming that provide us with advanced parallelization techniques are[2]:

- Pure functions. Pure functions are functions that do not have IO side effects and memory(the are stateless, which means that they don't have any state and return same values for different calls with same arguments). They allow reinterpretations and optimizations on the compile(interpreter processing) stage. The result of pure function call can be cached in a hash table, allowing substantial speed-up for recursive algorithms. Also, such functions can be considered thread-safe.
- Referential transparency. Referential transparency is one of the fundamental principles of the functional programming; only referentially transparent functions can be memoized(transformed into the equivalent functions that use cached results). Some languages provide the programmer with tools to guarantee this structure. Some other on the other hand require referential transparency in all functions. Due to the fact that this features requires same output for same input in any time, the referentially transparent expression is determined by the definition. This allows the programmer to avoid some extra calculations that frequently appear in the flow of parallel programming.[6]
- Persistent data structures. Persistent data structures are kind of data structures that retain access to the previous versions. If we have the sequence of p elements and we have to change the element n in it, we create the new version of the structure that is different from the previous only be the value of the n-th element. As a result, we will have two versions of the sequence with access to each one of them. This technique lets the programmer keep the data structure fast an immutable at the same time sparing memory by reusing it.
- Functors. Any class or data type, that stores values and implements method map is called a functor. Also, functor should return the collection with the same type to the collection passed to it.[7] For example, array is a functor, because an array is data structure that stores values and implements method map that allows us to apply the function against the values it stores. A benefit from such approach to the parallelization lies in the fact that transformation of the elements is independent which means that the sequence can be divided into ranges each of which would be transformed in parallel.
- Monads. The main application of monads in the functional programming is the isolation of the IO and stateful behaviour. The principle of monad application is the fact that function that don't create the side effect itself, it can create and can be used later. But IO and statefullness are not the only usecases for monads. They are useful when the programmer wants to describe the calculations in purely functional manner, at the same time executing other calculations in parallel.[4] Thus, two main usecases of monads are sequential calculations and handling of the inherently side-effect generated data.

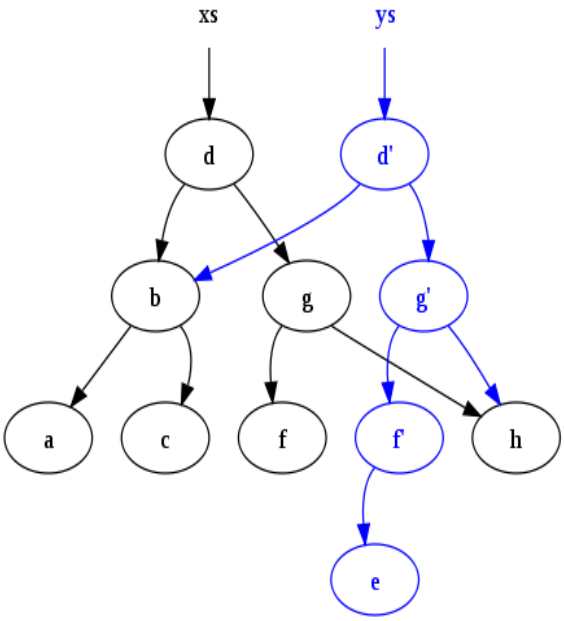

Fig 1. Representation of the work of persistent data structures (hash tables, vectors, lists, etc.)

III. PRACTICAL IMPLEMENTATION OF FUNCTIONAL PROGRAMMING IN PARALLELIZATION

The most famous use of the ideas of functional programming in parallel for relatively fast analysis and processing of huge amounts of data is the Google MapReduce algorithm.

This algorithm consists of two successive steps: the clustering of functors to an array of data and subsequent convolution, map steps and reduces accordingly (there can be more than one functor, for example, on Figure 2. there are three of them).

In the map step, a conversion is applied to each element of an array of input data that changes its contents, but retains the number of elements. In the case of distributed architecture, for the map step, the host computer receives the input data array, marks it and distributes it between its nodes by specifying the transformation. This process can take place several times before the convolution.

In the reduce step, a convolution of pre-processed data occurs. The main node receives responses from the working units and, based on them, forms the result - a solution to the problem that was initially formed.

The advantage of this algorithm is the possibility of its almost unlimited scaling, the absence of a global state and

the possibility of an iterative data flow with multiple transformations and reductions, of which even the hierarchical system can be built [4].

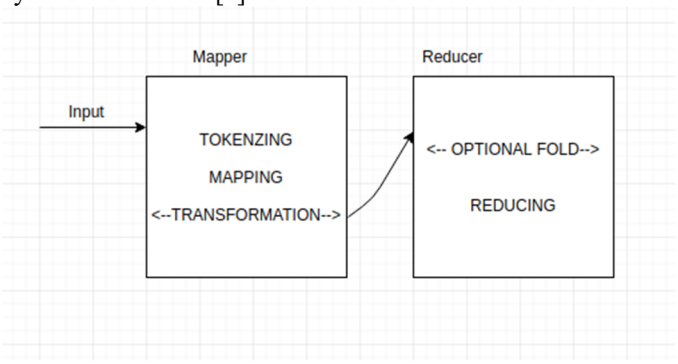

Fig 2. Listing an application of the Map Reduce algorithm for counting words in incoming texts with the possibility of parellisation on Clojure

```
(require '[clojure.string :as str])

;; Function for processing nodes on the Map step

;; для першочергової фільтрації вхідних даних.

(defn word-filter[words]

  (filter #(re-matches #"\w*" %) words))



;; Function that is used by processing nodes on
the Map step

;; for data transformation into indexed original
form

(defn sanitize-map[words]

  (map #(.toLowerCase %) words))



;; Function for Reduce step that is used for
data array convolution in the occurence hashmap
form.

(defn count-reduce[words]

  (reduce #(assoc %1 %2 (inc (%1 %2 0)))

        {}

        words))

;; Reduce step function for the final merge and
accumulation of values of the first-step
convolutions

(defn merge-count-reduce[occurence-maps]

  (reduce #(merge-with + %1 %2)

          (hash-map)

          occurence-maps))
```

```
(defn map-reduce[documents]

  (merge-count-reduce

    (map

      (comp

        count-reduce

        sanitize-map

        word-filter) documents)))


(def docs  [["a" "." "c"] ["mine" "token" "is"
"here" "a"]])


(map-reduce docs)
```

Conclusion

Functional programming is an effective paradigm for software development, which is becoming more and more important with the gradual obsoletion of the Moore's law and a significant increase in the number of processed data. Pipelining, which can be achieved with state-of-the-art functionality, the ability to cache and memorize the results of pure functions, the advantage of persistent data structures that occupy significantly less memory and the unchangeability of data with the isolation of the data in the individual elements of the program allows you to develop faster and less prone to program mistakes that allow you to avoid problems that are traditionally encountered during parallelization.[3] The functional approach allows you to create abstractions based on data and perception of functions in the form of data, which contributes to abstraction and allows you to use the mathematical apparatus organically when solving practical problems.

However, on the other hand, functional programming is now faced with a number of problems that make it difficult to use in the industry from a commercial point of view. First of all, this is an inadequate qualification of most programmers who traditionally use the imperative paradigm of programming and its derivatives in the form of object-oriented languages. Also, functional programming requires a lot of training in connection with many fundamentally new concepts and theoretical constructions that take time to assimilate. Compilers and interceptors of such languages are considerably more complex because they require the implementation of different calculation procedures simultaneously, the implementation of high-speed garbage collector (GC) and the distribution of clean and "contaminated" data zones.[8]

Another problem with functional programming is the complexity of simulating simulations and fast-changing systems that make lazy computing and immutability a burden rather than an advantage in connection with the constant

change of contexts and the complex indivisible interconnections of such systems [5,6].

In general, the prospect of using functional programming widely in the industry and academic environment is likely to be the gradual incorporation of the mature and appropriate elements of functional programming in more common programming development languages, as well as the formation of the appropriate ecosystems and communities of purely functional programmers with respect to tasks that are most in line with the rules. and points of benefit from the use of this type of language.

REFERENCES

[1] C. Okasaki "Purely Functional Data Structures" Cambridge University Press 1998

[2] G. Cousineau "The Functional Approach to Programming" Cambridge University Press 1998

[3] T. VanDrunen "Discrete Mathematics and Functional Programming" Paperback – October 16, 2012

[4] G. Michaelson "An Introduction to Functional Programming Through Lambda Calculus (Dover Books on Mathematics)" Copyright, 1989

[5] B. Jay "Pattern Calculus: Computing with Functions and Structures" Brodway NSW 2007

[6] G. E. Revesz "Lambda-calculus, Combinators and Functional Programming (Cambridge Tracts in Theoretical Computer Science)" Cambridge University Press 1998

[7] Ben Vandgrift "Clojure Applied. From practice to practioneer" Pragmatic Programmer Bookshelf 2015

[8] Michael Linn Functional programming patterns in Scala and Clojure Pragmatic Programmer Bookshelf 2013