A Strategy for Evaluating a Haskell Template in Parallel

Natela Archvadze

Parallelism in Haskell

The parallelism in Haskell represents the natural an reliable usage of calculating cores with following properties:

- The parallel programing id determined. This means that it's possible to repair parallel program in parallel without execution.
- The parallel program is a multilevel and declared, the have no direct connection with such a mechanisms as synchronization which is message.

As more abstract the program is as it's a simple to execute it on the parallel software. However it should be taken in account the quality of specification and dependence on data.

The model of parallel programing and the strategy of calculations

Lazy evaluations is a mechanism, which is being used for calculating expressions. The idea is that the calculations are being performed when there is a necessity. more precisely the calculation of arguments is being performed only in way and in time when it will be a strong necessity to reach a results. For example: after choosing first element of list the other part of the list is not necessary and it gives an opportunity to avoid in HEAD (1 : ones) impression the next calculation of ones endless list. Generally we have the following property: in case of using lazy calculations the expressions are being evaluated according to that context in which they are being used.

How to represent the "map" function, using the "Idea" which has following definition

```
map: :( a->b)->[a]->[b]
map f []    =[]
map f (x:xs)=f x : map f xs
```

The lazy data structure which is been created for "map" function and in which is evident two "ideas" can be written:

```
map:: ( a->b)->[a]->[b]
map f []       =[]
map f (x:xs)=let
               x′  = f x
               xs′ = map  f  xs
             in
               x′: xs′
```

The templates for the defined tail recursion code Haskell functions

The templates for the defined tail recursion code can be represented as[13-16]:

```
f [ ] = g1 [ ]
```

```
f ( x : xs ) = g2 ( g3 x ) ( g4 ( f ( g5 xs ) ) )
```
`g1, g2, g3, g4, g5` functions are depended on the program's conditions:

`g1` – is the function, to process empty list

`g2` – is the function, which combines the tail and the top of the list.

`g3` – the function, which processes the top of the list.

`g4` – is the function, which processes the recursion call for not empty list's

`g5` – is the function, which is processing the tail of not empty list for recursion call

It's possible to represent for example function "last" which returns the last element from list with following example:

```
last  :: [a] -> a
last [x]          =  x
last (_:xs)       =  last xs
g1 _ = error
g2 a b = b
g3 x = x
g4 x = x
g5 x = x
```

The template of the list is being represented using the "idea"

```
ListTemplate  [ ] = g1 [ ]
ListTemplate( x : xs ) = g2 ( g3 x ) ( g4 (ListTemplate( g5 xs )
) )
ListTemplate :: [a]->b
ListTemplate []       =   []
ListTemplate (x:xs)= let
                             x'  = g3  x
                             x''=g5 xs
                             x'''= ListTemplate ( x'')
                             xs' =g4 (x''')
                             in
                             g2( x': xs')
```

Conclusion

Nowadays the most important is the issue of creation such as program code which can be processed in parallel on several cores of one processor or on several computers. In the times when the computers with several cores is available for everyone, creation such a applied program which will effectively use several streams is still a big issue.

Functional programming gives an opportunity to noticeably simplify parallel programing. It's happening because in functional program are memory areas which are being used by several streams at the same time. Each function works with data which has been received from it's input. In spite of said before, the issue of effective dividing of calculations in different streams still exist.

References

1. Simon Marlow. Parallel and Concurent Programming in Haskell. O’REILLY. 2013.

2. Автоматическое построение «основной рекурсивной» части программы по описанию структур данных. Proceedings of the System Analysis and Information Technologies 14-th International Conference SAIT 2012 .