ISSN 1512-1232

UDC 004.4

THE MODERN APPROACHES IN PARALLEL PROGRAMMING

Natela Archvadze[1], Merab Pkhovelishvili[2], Lia Shetsiruli[3], Otar Ioseliani[4]

[1] Department of Computer Sciences Faculty of Exact and Natural Sciences I. Javakhishvili Tbilisi State University 2, University st., 0143, Tbilisi, GEORGIA, Natela.Archvadze@tsu.ge

[2] Department of Programming N.Muskhelishvili Computing Mathematic Institute of Georgian Technical University, 7, Akuri st., 0193, Tbilisi, GEORGIA, merab5@list.ru

[3] Department of Mathematics and Computer Science Shota Rustaveli State University, 35, Ninoshvili st., 6010, Batumi, Georgia, lika77u@yahoo.com

[4] Georgian-American University, 8 Merab Aleksidze str., 0160 Tbilisi, Georgia, otari.ioseliani@gmail.com

***Abstract**:*

The technologies of traditional parallel programing ***MPI*** (Message Passing Interface) *and* ***OpenMP*** *will be discussed, in addition the role of these technologies and inabilities. The directions of modern parallel programming which appeared by using functional languages will be presented. The parallelizing of the programs using competition will be reviewed.*

Keywords: *Message Passing Interface, functional programming, parallel programing, F#.*

1. Introduction

Nowadays the most used technology of parallel programming for distributed memory computing systems is MPI (Message Passing Interface) [1]. The main interaction solution between processes in such a systems is a message passing.

MPI is being supported by the programming languages like FORTRAN and C/C++. The full version of MPI consist from more than 100 functions and procedures. MPI supports creation of MIMD (Multiple Instruction Multiple Data) style of parallel programming which means unification of processes with different code. Actually it is very difficult to create and debug programs in such a style for this reason in practice is being used SPMD (Single Program Multiple Data) model in which all of the processes are using the same code.

For data processing is being used parallel or conveyor type of processing.

There are three type of parallel computing systems:

- Multiprocessors with shared memory;
- Computing systems with distributed memory;
- GRID – technologies.

1.1 The executing script of MPI program mpirun

The executing script of MPI program mpirun has following face:

```
mpirun –np N – machinefile ipfile
<Program with arguments>
```

ISSN 1512-1232

Where N is the count of processes and ipfile is the name of the file which contains the IP addresses or names on which should be executed the program.

The main method of interaction between processes is sending messages. The message is the collection of some type of data. Each message has several attributes including the number of process sender, the number of process receiver, the identifier (tag) of process etc. Process receiver by using tag can choose two messages received from one process. Tag is the nonnegative integer

For processing message attributes in C/C++ is being used special structure and in FORTRAN is being used array, the fields of array contain the values of attributes.

1.2 OpenMP technology

OpenMP technology is being used for the computers with shared memory. The main idea is to convert usual sequential program on C++ (FORTRAN or C) into parallel version using the compiler directives, special functions and environment variables.

The program created with OpenMP consists from sequential and parallel districts. The program begins from sequential district and at the beginning only one processor (thread) is working. While accessing into parallel district is being created several threads which are executing one and the same program. This program is being executed on different processors (cores). The directives of OpenMP can be separated in three parts: the determinant of parallel part, the distributor of tasks and synchronizer.

The parallel district:

The parallel district is being determined by the directive:

```
#pragma omp parallel [opcial [[,] opcial]…]
```

In the openMP the parallel district has following appearance:

```
<Execute in parallel> (<Program district 1>, < Program district 2>… < Program district N>)
```

2. Parallel programing on functional programing languages

2.1 Parallel programming on LISP

For parallelism on LISP is being used following functionalities – `MAPCAR` and `MAPLIST:`

```
(MAPCAR F (a b … h)) => ((F a) (F b)… (F h))
(MAPLIST F (a b … h)) => ((F (a b…h)) (F (b…h))… (F (h)))
```

We can say that, map – functionalities are parallel by nature. It is necessary to create the programming language compiler for multiprocessor computers that it would be possible to execute computing function for each argument on different processor. Each computing should be performed independently on processor which returns the result as map – functional indicating the row which has been called [2].

In one of modern versions of LISP, Objective CAML is introduced the concept of the threads and it is a solution of recording parallel algorithms.

Objective CAML has a library for “Light” processes, threads which are organized by processes and not by operating system. Such a processes are using addresses area of their process creator and for that reason need less resources. The principal difference between thread and process

ISSN 1512-1232

is that if memory is being used for data at the same time for the same program child processes. Usage of threads is the solution of executing parallel algorithms within language.

2.2 Parallelism realization on extended LISP

Considering function with three arguments:

```
(func x y z )
```

Which requires the execution of following sequence:

```
(z in parallel ( x afer y )).
```

It can be written this way:

```
(defun func (&parallel (z &concurrent (x y))) body )
```

The parallel analog of (progn func1 … funcn) function can be determined as:

```
(parallelprogn func1 … funcn)
```

The realization of this function:

```
([f g h] .aglist) => (f.arglist) (g.arglist) (h.aglist)
```

The sequential composition operator seq can be introduced following:

`If e1` – is not empty construction, then

```
e1 `seq` e2 (result e2);
```

In opposite case "empty value".

By this operator is declared that the construction `e1` should be processed before will be returned the result of `e2.`

2.3 Lazy programming or transferred evaluations

Lazy programming or transferred evaluations means that the computations are being postponed until the results will not became necessary. It allows to reduce the amount of computations at the expense of those computations. The programmer can describe dependencies between functions and do not observe the exploitation of the "extra computations".

[5] Describes the asynchronous, parallel and competitive programming principles and features of the language of Haskell. Parallelism in Haskell is natural and safe usage of computational cores. It is characterized by following features:

Parallel programming is determined. This means that parallel program can be verified in parallel without execution.

a) Parallel program is a high level and a declarative and has no direct connections with mechanisms as it is synchronization or message exchange.

As more abstractive the program is as it is simpler to execute parallel software, however it should be taken in account the quality of specification and dependence on data.

Lazy programing has been implemented naturally into functional paradigm because the functional programming languages have useful tools for software prototyping, fast processing of mathematical software and projecting electro – computational machines [5].

2.4 Parallel programming on F#

Parallel programing on F# is determined by three directions [6]

1) By the async workflows which gives an opportunity to sequenced writing of code and are not determining callback methods obviously and for this reason are using async block;

2) MailboxPrepreocessor is the class of the F# standard library which makes realization of one of the patterns of parallel programming.

ISSN 1512-1232

3) Processing of events using .NET. F# allows manipulation of event threads and works with them as a sequence and uses functions filter, map, split and others.

Conclusion

At the beginning of resolving practical issues should be developed parallel algorithms and considered following issues:

- How will be changed the life cycle of resolving the issue using parallel algorithm;
- While determining the issue how should be estimated parallelism;
- How to evaluate the results of parallel algorithm;
- How will be changed the determination of the issue during transition in parallelism;
- What kind of solutions are provided by the parallel programing paradigm on the level of development of parallel algorithms;
- Which solutions are being used for distributing parallel algorithm for solving issue on the stage which is before development the program?

Usage of parallel computational system will be useful in case when used technology and programming paradigm corresponds the architecture of the system and the structure of algorithm corresponds the structure of computational system and for this reason it can be necessary conversion of algorithm.

References

[1] P. Tsereteli. Parallel Programing Using MPI and OpenMP technologies. Lectures, 2014, sangu.ge/images/MPI_OMP.docx
[2] Natela Archvadze, Merab Pkhovelishvili. Programming paradigm and the aspects of data processing in functional paradigm. Georgian Electronic Scientific Journal: Computer Science and Telecommunications 2009|No.2(19) .
[3] Graham Hutton. Programming in Haskell. 2007. ISBN:9780521871723 http://ebooks.cambridge.org/ebook.jsf?bid=CBO9780511813672
[4] Dushkin R. V. Functional programing in Haskell — M.: DMK-Press, 2007. — 608 стр. ISBN 5-94074-335-8
[5] Simon Marlow. Parallel and Concurrent Programming Haskell. 2012.
[6] Chris Smith. Programing F#. Publisher: Symbol-Plus, 2011.

Article received: 2016-07-18