Adapting evolutionary algorithms to the concurrent functional language Erlang

J. Albert Cruz
Centro de Arquitecturas Empresariales
Universidad de Ciencias Informáticas
La Habana, Cuba
jalbert@uci.cu

Juan-J. Merelo, Antonio M. Mora and Paloma de las Cuevas
Departamento de Arquitectura y Tecnología de Computadores
Universidad de Granada
Granada, Spain
{jmerelo,amorag,paloma}@geneura.ugr.es

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent Programming*; D.2.8 [**Software Engineering**]: Performance measures; G.1.6 [**Mathematics of Computing**]: Numerical Analysis—*Optimization*

General Terms

Algorithms, Languages, Performance, Measurement

Keywords

Evolutionary Algorithms, Functional Languages, Concurrent Languages, Erlang, Algorithm Implementation

1. INTRODUCTION AND STATE OF THE ART

Evolutionary Computation (EC) seems to be an effective method for improving, and therefore optimizing, behaviours that lead to a better use of available resources. This scientific field is very active producing libraries of diverse quality and applying it to many domains. Nevertheless it is concentrated in the use of widespread implementation technologies such as C/C++, Fortran and Java. Getting out of that mainstream it is not normally seen a land for improvement science. One no mainstream language with very much potential is Erlang, it is an implementation technology that support functional and concurrent paradigms and that's starting to been used in the scientific community [4].

Genetic Algorithms (GA) [2] are general function optimizers that encode a potential solution to a specific problem on a simple data structure (a chromosome). There are only two components of them that are problem dependent: the solution encoding and the function that evaluate the quality of a solution, i.e., the fitness function. The rest of the algorithm does not depend on the problem and could be implemented following the best architecture and engineering practices.

GECCO'13, July 6-10, 2013, Amsterdam, The Netherlands.

Inside the object oriented world there are reported various analysis for modeling GA behaviour, nevertheless that is not the case of the functional side. This work tries to show some possible areas of improvement on that sense.

This article is focused on GAs as a domain of application, and describes their principal concepts and characteristics, showing how can they be modeled by means of Erlang constructs.

The rest of the paper is organized as follows: next section presents considerations around the functional paradigm and its conceptual relation with evolutionary computation; we will explain then the general characteristics of a pool based concurrent evolutionary evolutionary used as use case in this work. Results and conclusions obtained will be presented in Section 3.

2. EVOLUTIONARY ALGORITHM IN A CONCURRENT FUNCTIONAL LANGUAGE

A variety of programming patterns, i.e., paradigms, exist for implementing the algorithms models. GAs are characterized by an intensive use of strings (lists of some kind) for encoding genes, the existence of populations that evolve by theirselves, and the variation in selection criterias through time. A programming language whose characteristics fit these needs would be highly appreciated.

2.1 Erlang description

There is a claim in modern software development for programming languages that help with concurrent programming and simplify coding practice. The functional programming language Erlang would an answer that provides the actor pattern concept for concurrency and the functional paradigm for general modeling, design and coding of solutions.

Actors are concurrent execution units which use asynchronous message passing for communication. They are implemented as processes in the Erlang's virtual machine and not like operating system (OS) threads, therefor they are very lightweight in creation and execution. The use of messages eliminate the sharing of state and eliminate many of the typical problems of concurrent development, thats support the emulation of the Object Oriented (OO) paradigm with its modeling facilities.

Functional programming is defined by the use of functions in program composition and by using lists. Erlang honored the functional linage and include an ultra fast persistent

Message	Description
`{configPool, NIM}->`	Initialization, the parameter NIM is the initial configuration.
`{requestWork, Pid, Capacity} ->`	Client requests for a population to evolve.
`{generationEnd, NewIndividuals, OldIndexes, Pid}->`	One client report its successfully end of calculation.

Table 1: The messages that a pool accepts.

Message	Description
`initEvolution ->`	Marks the beginning of the processing.
`{evolve, P, NIndexes} ->`	When the pool could assign a subpopulation to process.

Table 2: The messages that a client is able to respond to.

technologies called Dets and Mnesia. Besides, Erlang offers the concept of records for group data.

2.2 Genetic Algorithm mapping to Erlang

Genetic algorithms, as many computational models, tend to be described in literature on its operational and imperative way. Their implementation in a functional language must follow a different path, structuring the algorithm model in terms less imperative and more declarative. We are going to use a parallel pool based evolutionary strategy [**?**] as use case in order to show our mean.

The pool will be an execution entity (an actor acting like a server) that will own the population and also keep a track of the advance in the solution search. The clients, which are concurrent, will do the calculations and will join and leave the system at any time without consequences. Chromosomes will be encoded as lists and the different parts of the GAs algorithms will be implemented as Erlang functions.

An Erlang actor is implemented by a sequence of pairs pattern/expression defining each message that it could handle. It is close to the OO parlance and a way to organize the code. In this case we use one message per service that pool must provide, table 1 presents this.

Clients are modeled by actors. They are the units of evolution, with the main computation responsibilities; the table 2 shows its interface.

2.3 Generality and configuration

The two previous components have the architecture of the algorithm, they are general and could be used for many problems. In order to solve a particular situation, they must be *injected* by several functions and data structures which define: chromosomes, fitness function, mutation operator, selection criteria and replacement policy. All these particularizations must be implemented in an Erlang source file and configured in the *configBuilder* module.

The design made promotes a clear separation between architecture (the general, constant and paradigmatic foundation part) and problem encoding (the representation and criteria of solution finding).

3. EXPERIMENT AND CONCLUSIONS

In this ongoing project we are testing the efficiency and simplicity of implementations of GAs by functional programming. The parallel models of GA are mapped to actors in the Erlang languages obtaining easily to understand architectures. All the code has been released as open source code at `https://github.com/jalbertcruz/erlEA/`.

The library was tested with *MaxOnes* problem. The chromosomes was 128 elements long, with an initial population of 256 individuals. 50 clients was used and they worked with 20 individuals each time a pool assign a generation to them. The solution, an optimum, was reached after 3229 assignments.

With this concept test we are showing how simple is to structure a parallel GA, now we could proceed with more complex GA models, experiments and problems in order to explore the potential of the technology.

Acknowledgements

This work is supported by projects TIN2011-28627-C04-02 and TIN2011-28627-C04-01 and -02 (ANYSELF), awarded by the Spanish Ministry of Science and Innovation and P08-TIC-03903 and TIC-6083 (DNEMESIS) awarded by the Andalusian Regional Government. It is supported too by the PhD Programm of Intelligent Systems and Softcomputing hold by the Asociación Universitaria Iberoamericana de Postgrado -AUIP-, the University of Granada, Spain, and the University of Informatics Sciences, Cuba. It has also been supported by project 83, Campus CEI BioTIC http://ceibiotic.ugr.es

4. REFERENCES

[1] H. Adeli and N.-T. Cheng. Concurrent genetic algorithms for optimization of large structures. *Journal of Aerospace Engineering*, 7(3):276–296, 1994.

[2] D. E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[3] L. Santos. Evolutionary computation in ada95, a genetic algorithm approach. *Ada User Journal*, 23(4), 2002.

[4] G. I. Sher. *Handbook of Neuroevolution Through Erlang*. Springer, 2013.

[5] K. Tagawa. Concurrent differential evolution based on generational model for multi-core cpus. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7673 LNCS:12–21, 2012. cited By (since 1996) 0.

[6] F. Xhafa, L. Barolli, and K. F. Li, editors. *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2012, Victoria, BC, Canada, November 12-14, 2012*. IEEE, 2012.