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
Departamento de Arquitectura y Tecnología de Computadores
University of Granada
Granada, Spain
jmerelo@geneura.ugr.es

Antonio Miguel Mora
Departamento de Arquitectura y Tecnología de Computadores
University of Granada
Granada, Spain
amorag@geneura.ugr.es

1. INTRODUCTION

Evolutionary Computation (EC) seems to be an effective method for improving, and therefore optimizing, behaviours that lead to a better use of available resources. For doing so the scientific community has produced several libraries of diverse quality and applied it in 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 is not normally considered in the community.

Genetic Algorithms (GA) [3] 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 (and should) be implemented following the best architecture and engineering practices.

On the object oriented world there are reported various analysis to model GA behaviour, nevertheless that is not the case on 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.

Other concurrent oriented languages such as Ada [5] has been reported as a feasible technology for implement this kind of algorithms.

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 obtained will be presented in Section **??**, and we will finish the paper with the conclusions that derive from them (commented in Section 3).

2. IMPLEMENTING AN EVOLUTIONARY ALGORITHM IN A CONCURRENT FUNCTIONAL LANGUAGE

A varied arsenal of programming patterns, i.e., paradigms, exist for implement 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 characterization

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

Actors are concurrent execution units which use asynchronous message passing for communication. They are implemented as processes in the virtual machine (VM) of the language's runtime and not like O.S. threads. Thus they are very lightweight in birth, live and death. The use of immutable messages eliminate many of the typical problems of concurrent development, and support the emulation of the Object Oriented (OO) paradigm with its modeling facilities.

Functional programming has been defined by the role of functions in program composition and by the use of the very efficient list data structure with many high order functions that operate upon them. Erlang honored the functional linage and include an ultra fast persistent technology called Dets and Mnesia. Besides, Erlang offers a macro system, and the concept of records for encapsulate coding patterns and group data by an entity.

2.2 Genetic Algorithm mapping to Erlang

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.

Genetics algorithms, as many computational models tend to be described in literature as 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[4] as use case in order to show our mean.

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

An Erlang actor is implemented by a sequence of pairs pattern/expression that define 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.

2.2.1 Generality and configuration

The two previous components have the general 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: the chromosomes, fitness function, mutation operator, selection criteria and replacement policy. All these particularizations must be implemented in some 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. 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/`.

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 AsociaciÃşn Universitaria Iberoamericana de Postgrado -AUIP-, the University of Granada, Spain, and the University of Informatics Sciences, Cuba.

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] A. Bienz, K. Fokle, Z. Keller, E. Zulkoski, and S. Thede. A generalized parallel genetic algorithm in erlang. Midstates Conference For Undergraduate Research in Computer Science and Mathematics, 2011.

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

[4] J. J. M. Guervós, A. M. Mora, C. M. Fernandes, A. I. Esparcia-Alcázar, and J. L. J. Laredo. Pool vs. island based evolutionary algorithms: An initial exploration. In Xhafa et al. [7], pages 19–24.

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

[6] 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.

[7] 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.