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

ABSTRACT

In this paper we describe how the usual sequential and procedural Evolutionary Algorithm is mapped to a concurrent and functional framework using the Erlang language. The design decisions, as well as some early results, are shown.

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

The Evolutionary Computation (EC) field is focused on the use widespread implementation technologies such as C/C++, Fortran and Java. Getting out of that mainstream it is not normally seen as a land for scientific improvements. 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 [6].

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 be implemented following the best architecture and engineering practices.

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 how their principal traits can the be modeled by means of Erlang constructs. This will be done in the next section.

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.

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, therefore 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 ultra fast technologies for persist data called Dets and Mnesia. Besides, Erlang offers the concept of records for group data which helps to modularize the code.

Genetic algorithms, as many other computational models, tend to be described in literature in an operational and imperative way. Their implementation in a functional lan-

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 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 a client is able to respond.

guage must follow a different path, structuring the algorithm model in less imperative and more declarative terms. We are going to use a parallel pool based evolutionary strategy [4] as use case.

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.

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 proposed design promotes a clear separation between architecture (the general, constant and paradigmatic foundation part) and problem encoding (the representation and criteria of solution finding) which is good for apply the library to solve another problem.

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 were used and they worked with 20 individuals each time a pool assigned a generation to them. The solution, an optimum, was reached after 3159 assignments in 3.144590 seconds. For 100 clients the solution was reached in 6.121570 seconds and 6446 assignments; and for 25 clients it was 1835 assignments in 1.687000 seconds. This results shows that in this case there it is not convenient to use very much clients in order to obtain the solution, when the number of clients increase there is a lot of bad solutions thats cames and goes to the clients.

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 project TIN2011-28627-C04-02 (ANYSELF), awarded by the Spanish Mineco and P08-TIC-03903 awarded by the Andalusian Regional Government. It is also supported too by the PhD Program of the AUIP and by project 83, Campus CEI BioTIC.

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. In *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. Merelo-Guervós, A. M. Mora, C. M. Fernandes, A. I. Esparcia-Alcázar, and J. L. Jiménez-Laredo. Pool vs. island based evolutionary algorithms: An initial exploration. In Xhafa et al. [8], pages 19–24.

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

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

[7] 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.

[8] 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.