IMPLEMENTING PARALLEL EVOLUTIONARY ALGORITHMS IN A DISTRIBUTED WORKFLOW

Jonathan Burge
Whitworth University
jburge16@my.whitworth.edu

ABSTRACT

This paper contains research focused on the utilization of evolutionary algorithms (EAs), specifically the use of the DEAP library, in the environment of an implicitly parallel language. This research was motivated primarily by a desire to allow user to implement parallel EAs without the prerequisite mastery of MPI. The computation of EAs consist of many instances of the same computational operations. By evaluating EAs in an implicitly parallel work-space, we can obtain the benefits provided by parallelization without dealing with the full complexity of a distributed computer system. Swift/T is an implicitly parallel scripting language developed specifically for high performance computing. In this study, we investigate the benefits of using Swift/T to implicitly parallelize EAs by evaluating the performance and scalability from data collected from various tests using DEAP EAs and a simple agent based model simulation which is used for the fitness evaluation.

1. INTRODUCTION

The utilization of parallel computation is essential for any form of high processing computation. The parallel computation of tasks allow greater performance as the number of nodes is increased. These systems are able to compute solutions for problems which were previously unfeasible. Optimization of complex systems are one subset of the problems HPC systems can address. In an optimization problem, a solution is graded based on its evaluated fitness determined by the initial parameters of the solution. HPC systems allow for a parallel search of the search space, whereas a serial search would be impractical. Previous research has demonstrated that non-deterministic search methods are the most efficient means of navigating large search spaces for optimal solutions *. Of the several common non-deterministic heuristics, one of the most utilized is evolutionary algorithms (EAs).

In order to effectively implement evolutionary algorithms for HPC systems, the EAs must also be computed in a parallel manner. Fortunately, "EAs are naturally prone to parallelism, since most variation operations can be easily undertaken in parallel" [1]. Although EAs are naturally easy to compute in parallel, the implementation of such an algorithm requires significant understanding of some sort of message passing or communication between nodes, MPI being one of the most standard. Additionally, most users who would benefit from using a HPC system to solve their problem do not possess the technical capacity to implement their algorithm to run on a distributed system. This is a case in which utilizing an implicitly parallel language would allow these users to obtain the benefits of parallel computation without the complexity of dealing with a distributed system.

Swift/T is an implicitly parallel scripting language designed for HPC systems. It is naturally concurrent and allows the user to easily call functions from external languages including C, C++, Fortran, Python, R, Tcl or executable programs. Swift/T uses MPI and provides an abstraction for the user that hides this complexity [2]. In this report, we will demonstrate how Swift/T may be used to easily implement parallel evolutionary algorithms, without dealing with the complexity of message passing or other means of inter-node communication. Our motivation is to demonstrate that this practice allows users to obtain efficient and scalable results without dealing with the full complexity of a distributed system.

2. DEAP

In this study, instead of writing our own evolutionary algorithms, algorithms from the Distributed Evolutionary Algorithms in Python (DEAP) library were implemented. This was done to demonstrate how preexisting algorithms may be easily integrated into a Swift workflow. This ability to integrate preexisting code and algorithms into the workflow of a Swift script is one of the great strengths provided by the Swift/T scripting language.

The DEAP library's development was motivated by a desire to provide adaptable tools for scientists to implement distributed evolutionary algorithms specific to their problems, while abstracting away the complexity of message passing between nodes. This was a shift away from previously developed frameworks in which typically a variety of canned algorithms were provided to the user. The DEAP library is composed of modules, each of which has a specific function in the overall algorithm execution. The Core, Creator and Tools modules deal with the framework for the basic mechanics of evolutionary algorithm execution. Other important modules include the Distributed Task Manager and Algorithms modules. The first deals with managing the parallel execution of the EA. The latter contains several commonly used EAs which may either be used or adapted to meet the user's needs [4].

3. TESTING FRAMEWORK

In this study EAs were used to optimize a parameter set for a simple agent based model (ABM). This ABM was implemented using Repast Simphony, a Java based framework for executing agent based models and simulation. This specific model simulated a two dimensional environment populated by zombies and humans. Over the course of the simulation, zombies would chase and infect humans. The initial parameters of this simulation were the initial populations of the zombies and humans as well as the speed at which they could pursue or flee. An evolutionary algorithm was used to optimize these initial parameters in order to maximize the remaining human population at the conclusion of each simulation. The values of these initial parameters were contained by a lower and upper bounds.

The motivation for this research is based on the fact that often times, the most time and computational intensive aspect of an evolutionary algorithm is the evaluation of each solution's fitness. This computation could entail the execution of an external application which would then output fitness results. By reducing the number of solutions evaluated before one of a desired caliber is found, the cost of executing the algorithm decreases. In this study, I used this ABM as a simple example with which to test algorithm efficiency over various settings.

In this model, the value of the four parameters has an extremely intuitive impact on the solution's fitness. This allows us to easily infer the global optima of the search space defined by the given bounds. My aim was to study the efficiency and speed at which DEAP's evolutionary algorithm was able to approach this global optima at various settings.

When considering the performance of a given algorithm, two attributes must be considered. The first is the rate of improvement of both the population's fitness and that of the most fit individual. The second attribute is the magnitude of this improvement. For example, some settings for the evolutionary algorithm may quickly provide various solutions with a fitness of about half the actual global optima, but may struggle to further improve upon these solutions. Alternatively, an algorithm may take a long time to provide a useful solution, but actually be the fastest at obtaining solutions very close to the global optima. Since the global optima is known for these tests, it is easy to compute the respective speed at which each algorithm may provide a solution of a given quality.

4. FUTURE TESTS

I am currently in the preliminary stages of this testing. I plan on first testing algorithm performance and speed over a parameter spread of various solution population sizes. I will be measuring the number of simulation executions required to obtain a solution of various qualities. These qualities will be 50%, 75%, and 95% the global optima.

I believe that changes in the solution population size may have an interesting impact on the search for solution of various qualities for the following reasons. First of all, larger populations require the computation of more new solutions per generation then when smaller solution populations are used. This will generally increase the number simulations to execute per generations. However, increasing population size allows for a greater degree of solution diversity to be preserved. Since there are more individuals in the population, more distinct attributes of these solutions may be preserved between generations. This may allow for a more comprehensive search of the search space and consequently provide faster performance for the computation of more fit solutions. Other research regarding the effect of population size on algorithm performance has been conducted by Matej Črepinšek et al. [5].

6. ACKNOWLEDGEMENTS

I would like to thank the mentors who worked with me, Justin Wozniak and Mike Wilde, for their instruction and guidance. I would also like to acknowledge the help and support of my peers of the BigDataX REU program. Additionally I would like to acknowledge Jonathan Ozik and the other researchers at Argonne who both provided direction as well as had previously implemented the code used by the agent based model.

7. REFERENCES

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation,* 2002.

[2] T. Armstrong. *Implicitly Parallel Scripting as a Practical and Massively Scalable Programming Model for High-Performance Computing.* PhD thesis, 2015.

[3] E. Cantú-Paz. *A Summary of Research on Parallel Genetic Algorithms,* 1995.

[4] F. Rainville, F. Fortin, M. Gardner, M. Parizeau, and C Gagné. DEAP : A Python Framework for Evolutionary Algorithms. *Genetic and Evolutionary Computation Conference GECCO2012: EvoSoft Workshop.* 2012.

[5] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.* 45, 3, Article 35 (July 2013), 33 pages.