Mathematical and Computer Modelling 53 (2011) 1167–1181

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

Distributing evolutionary computation in a networked multi-agent environment

Wei-Po Lee *, Hong-Yi Ke

Department of Information Management, National Sun Yat-sen University, Kaohsiung, Taiwan

ARTICLE INFO

Article history:
Received 8 December 2009
Received in revised form 3 September 2010
Accepted 29 November 2010

Keywords:
Mobile agent
Evolutionary algorithm
Distributed computing
Autonomic computing
Gene regulatory network
Evolutionary robotics

ABSTRACT

It has become increasingly popular to employ evolutionary algorithms to solve problems in different domains, and parallel models have been widely used for performance enhancement. Instead of using parallel computing facilities or public computing systems to speed up the computation, we propose to implement parallel evolutionary computation models on networked personal computers (PCs) that are locally available and manageable. To realize the parallelism, a multi-agent system is presented in which mobile agents play the major roles to carry the code and move from machine to machine to complete the computation dynamically. To evaluate the proposed approach, we use our multi-agent system to solve two types of time-consuming applications. Different kinds of experiments were conducted to assess the developed system, and the preliminary results show its promise and efficiency.

1. Introduction

Evolutionary algorithms (EAs) have been developed to construct more dynamic models of computational intelligence. They simulate the process of natural evolution to search for the fittest through selection and re-creation over generations. These algorithms are population-based optimization processes—based on the collective learning process within a population of individuals in which each individual represents a search point in the space of potential solutions for a given problem. Generally speaking, the dimension of the solution space is determined by the length of the chromosome. For instance, in the binary encoding scheme, a chromosome with length n indicates that the evolution techniques are expected to find the appropriate solution from a space with 2^n candidates. It can be observed that, when the length of the chromosome is reasonably increased to match the increase in task complexity, the solution space will grow exponentially. Consequently, the search will become increasingly difficult.

In order to use EAs to solve more difficult problems, two inherent features of EAs must be taken into consideration seriously. The first is *premature convergence*. As is well known, one of the attractive features of evolutionary algorithms is that they can quickly concentrate on searching promising areas of the solution space. But this feature also has a negative effect of losing population diversity before the goal is reached. In other words, the EA converges to local optima. To overcome this problem, parallel model EAs have been proposed to divide the original population into several subpopulations to maintain population diversity, and their efficiency has been confirmed [1–3]. The other problem that needs to be encountered is the *computation time*. As mentioned above, the EA is a population-based approach, and it has to evaluate all population members before selecting the fittest to survive. When using an EA to solve a problem, the population size must be reasonably large in order to allow the EA to search the space globally, and it typically increases with respect to the increasing problem difficulty. As a result, an inordinate amount of time is required to perform all the evaluations for a hard problem. To enhance the search

* Corresponding author.
E-mail address: wplee@mail.nsysu.edu.tw (W.-P. Lee).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.11.084

performance and speed up EAs at the same time, it is therefore critical to develop methods of running parallel model EAs on high-performance computing environments.

Conceptually, the central idea of the parallel model EA mentioned above is to divide a big population in a sequential EA into multiple smaller subpopulations and to distribute them on separate processors. One way to support this parallelism is to implement the parallel models on parallel computers. Depending on the number of subpopulations involved, different types of parallel computers have been used. For a small number of subpopulations, distributed memory MIMD (Multiple Instruction stream, Multiple Data stream) computers are often used. On the other hand, massively parallel SIMD (Single Instruction stream, Multiple Data stream) computers are used to implement parallel models with a large number of subpopulations. Though parallel computers can speed up EAs dramatically, they are not generally available. Also, it is expensive to upgrade their processing power and memory. The other way is to take public computing systems to speed up the computation [4,5]. The system of public computing is designed as a software platform utilizing heterogeneous computing resources from volunteer computers. It is an approach of distributing computation through the Internet, and has emerged as a useful computing architecture for tackling large-scale computation problems. However, computation with public computer architecture has some requirements that are not always available, such as little task interdependency, light network communication, and high network reliability. In particular, because the volunteer computational nodes are outside the system's control, extra work (e.g., redundant computing that conducts the same work on different machines [6]) is needed to ensure the correctness of the results.

To make the parallel model EAs more realistic, in this paper we propose a flexible and efficient alternative, agent-based methodology to work on networked computers that are locally available and manageable to support parallelism. As the autonomous components in the autonomic computing paradigm [7,8], agents can sense the environment situations and then respond to the environment by acting reactively and proactively. Moreover, different agents can manage their behaviors and cooperate with each other to achieve their goal. With these unique characteristics, our agent-based system can operate in an adaptive way, in which mobile agents play the major roles that dynamically allocate available machines for the evolutionary computation (EC) code. To verify the proposed approach, we developed a prototype application system on a middleware platform JADE (Java Agent Development Framework) in which the agents control their own thread of execution and can easily be programmed to initiate the execution of actions autonomously. We also applied our agent-based system to two types of time-consuming EA applications. The preliminary results show the promise and efficiency of our approach.

2. Background

It has become increasingly popular to employ EAs to solve problems in different domains, and parallel models have been used for performance enhancement. As mentioned above, to parallelize the above evolutionary procedure is to divide a big population in a sequential EA into multiple smaller subpopulations so that they can be evaluated on separate processors simultaneously. According to the subpopulation size, parallel EAs can be categorized into two types: coarse-grain and fine grain. A coarse-grain (or island model) EA divides the whole population into a small number of subpopulations in which each subpopulation is evaluated by an independent EA in a separate processor. In this model, each subpopulation is manipulated by a sequential EA, and the selection and genetic operations are limited to happen only in the local subpopulation. A communication (or migration) phase is introduced in the island model EA to periodically select some promising individuals from each subpopulation and send them to other subpopulations. A common strategy is to use the selected individuals to substitute the worst ones in the neighboring subpopulations. In this way, an EA has higher possibility to maintain population diversity and to protect good solutions found locally. The coarse-grain EA is usually implemented on distributed memory MIMD architecture that is composed of a collection of processors and their associated memories, with the processors interconnected to provide a means for communicating. In this architecture, each processor has the capability to execute an independent instruction stream.

On the other hand, the fine-grain model EA divides the original population into a large number of subpopulations in which each of them includes only a small number of individuals. This model is designed to take advantage of machines with a large number of processors (1000 or more), and, in the ideal case, each individual is evaluated on a different processor. In this model, the selection and mating are restricted to occur only between an individual and its localized neighborhoods. The network topology in a fine-grain-type EA determines how the local optimals spread to the entire population, and this affects the performance of EA profoundly. How to constrain the interactions between subpopulations is yet to be investigated. The fine-grain model is usually implemented on massively parallel SIMD computers that contain a control processor to drive several task processors: the control processor issues a single instruction stream to all task processors that execute the instructions simultaneously for their own individual data streams.

Implementing parallel model EAs involves expensive parallel computers that normally cannot be expected in a campus-based computing environment. A possible alternative is to use a public computing system, such as the BOINC (Berkeley Open Infrastructure for Network Computing [4]) instead. But it should be noted that public computing is different from grid computing. Grid computing shares secured, trusted and centrally managed resources within or between organizations, while in public computing the resources are contributed mostly from network volunteers, and the reliability of these machines is thus not guaranteed. Therefore, a promising method without expensive hardware facilities and with the control of the computing resources is to map parallel model EAs onto a set of locally networked computers. There are some possible