Parallel Computing 46 (2015) 98–119

Contents lists available at ScienceDirect

Parallel Computing

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

A scalable parallel genetic algorithm for the Generalized Assignment Problem

Yan Y. Liu [a,b,c,d,*], Shaowen Wang [a,b,c,d,e,f]

[a] CyberGIS Center for Advanced Digital and Spatial Studies, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[b] CyberInfrastructure and Geospatial Information Laboratory (CIGI), University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[c] Department of Geography and Geographic Information Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[d] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[e] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States
[f] Department of Urban and Regional Planning, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States

ARTICLE INFO

Article history:
Received 13 February 2013
Received in revised form 4 December 2013
Accepted 23 April 2014
Available online 9 May 2014

Keywords:
Generalized Assignment Problem
Genetic algorithm
Heuristics
Parallel and distributed computing
Scalability

ABSTRACT

Known as an effective heuristic for finding optimal or near-optimal solutions to difficult optimization problems, a genetic algorithm (GA) is inherently parallel for exploiting high performance and parallel computing resources for randomized iterative evolutionary computation. It remains to be a significant challenge, however, to devise parallel genetic algorithms (PGAs) that can scale to massively parallel computer architecture (also known as the mainstream supercomputer architecture) primarily because: (1) a common PGA design adopts synchronized migration, which becomes increasingly costly as more processor cores are involved in global synchronization; and (2) asynchronous PGA design and associated performance evaluation are intricate due to the fact that PGA is a type of stochastic algorithm and the amount of computation work needed to solve a problem is not simply dependent on the problem size. To address the challenge, this paper describes a scalable coarse-grained PGA–PGAP, for a well-known *NP*-hard optimization problem: Generalized Assignment Problem (GAP). Specifically, an asynchronous migration strategy is developed to enable efficient deme interactions and significantly improve the overlapping of computation and communication. Buffer overflow and its relationship with migration parameters were investigated to resolve the issues of observed message buffer overflow and the loss of good solutions obtained from migration. Two algorithmic conditions were then established to detect these issues caused by communication delays and improper configuration of migration parameters and, thus, guide the dynamic tuning of PGA parameters to detect and avoid these issues. A set of computational experiments is designed to evaluate the scalability and numerical performance of PGAP. These experiments were conducted for large GAP instances on multiple supercomputers as part of the National Science Foundation Extreme Science and Engineering Discovery Environment (XSEDE). Results showed that, PGAP exhibited desirable scalability by achieving low communication cost when using up to 16,384 processor cores. Near-linear and super-linear speedups on large GAP instances were obtained in strong scaling tests. Desirable scalability to both population size and the number of processor cores were observed in weak scaling tests. The design strategies applied in PGAP are applicable to general asynchronous PGA development.

* Corresponding author at: Department of Geography and Geographic Information Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States. Tel.: +1 2172449315.
E-mail addresses: yanliu@illinois.edu (Y.Y. Liu), shaowen@illinois.edu (S. Wang).

http://dx.doi.org/10.1016/j.parco.2014.04.008
0167-8191/

1. Introduction

Inspired by natural selection, Genetic Algorithm (GA) represents a generic heuristic method for finding near-optimal or optimal solutions to difficult search and optimization problems [1]. GA mimics iterative evolutionary processes with a set of solutions encoded into a population at the initialization stage. Through GA operators (e.g., selection, crossover, mutation, and replacement) that are often stochastic, the population evolves based on the rule of "survival of the fittest" [2,3]. Such an evolutionary process stops when the population converges to solutions of specified quality. The computational challenges of GA are attributed to both problem-specific characteristics (e.g., problem 'difficulty' (e.g., *NP*-hard), problem size, the complexity of fitness function, and distribution characteristics of solution space specific to problem instances), and runtime efficiency of stochastic search [4].

High performance and parallel computing has been extensively studied to tackle the aforementioned computational challenges in GA as GA has inherent parallelism embedded in the evolutionary process [5]. For example, a population can be naturally divided into a set of sub-populations (also called demes) that evolve and converge with a significant level of independence. Various types of parallel genetic algorithms (PGAs) have been developed and broadly applied in a rich set of application domains [5–8]. More interestingly, previous work by Alba and Troya [9] showed that PGA computation not only improves computational efficiency over sequential GAs, but also facilitates parallel exploration of solution space for obtaining more and better solutions. In fact, Hart et al. [10] showed that running PGA even on a single processor core outperformed its sequential counterpart. Therefore, PGA is often considered and evaluated as a different algorithm rather than just the parallelization of its corresponding sequential GA.

This paper describes a scalable PGA (PGAP) to exploit massively parallel high-end computing resources for solving large problem instances of a classic combinatorial optimization problem – the Generalized Assignment Problem (GAP). GAP belongs to the class of *NP*-hard 0–1 Knapsack problems [11–13]. Numerous capacity-constrained problems in a wide variety of domains can be abstracted as GAP instances [14] such as the job-scheduling problem in computer science [15] and land use optimization in geographic information science and regional planning [16]. Various exact and heuristic algorithms have been developed to solve GAP instances of modest sizes [17]. However, in practice, problem instances often have larger sizes while the problem solving requires quick solution time and the capability for finding a set of feasible solutions of specified quality, which compounds the computational challenges.

Our PGA approach focuses on the scalability to massively parallel processor cores (referred to as cores hereafter) available from high-end computing resources such as those provided by the National Science Foundation XSEDE [18] cyberinfrastructure. PGAP is a coarse-grained steady-state [19] PGA that searches solution space in parallel based on independent deme evolution and periodical migrations among connected demes. Scalability is a key to efficiently exploiting a large number of cores in parallel. Previous PGA implementations mostly rely on synchronization to coordinate parallelized operations (e.g., the migration operation in coarse-grained PGAs and the selection operation in fine-grained PGAs), primarily because the computation of PGA is an iterative process and it is straightforward to implement iteration-based synchronization. Synchronization is often needed at two places: (1) waiting for all PGA processes to rendezvous before migration operations; and (2) using synchronous communication to exchange data. While success on scaling PGA to many cores has been achieved based on hardware instruction-level synchronization supported by SIMD architectures [20], we argue that the computational performance of PGA is under-achieved through synchronizing iterations across massively parallel computing resources with MIMD architecture [21].

Therefore, an asynchronous migration strategy is designed to achieve scalable PGA computation through a suite of non-blocking migration operators (i.e., export and import) and buffer-based communications among a large number of demes connected through regular grid topology. The asynchrony of migration is effective to not only remove the costly global synchronization on deme interactions, but also allows for the overlapping of GA computation and migration communication. Addressing buffer overflow issues caused by inter-processor communications and understanding their relationship to the configuration of asynchronous PGA parameters are crucial to design scalable PGA on high-end parallel computing systems. Through algorithmic analysis on PGAP, we identified two buffer overflow problems in exporting and importing migrated solutions, respectively. In export operations, the overflow of the outgoing message buffer used by the underlying message-passing library may cause runtime failure and abort the PGA computation. In import operations, the overflow of the import pool maintained for receiving solutions from neighboring demes may cause the loss of good solutions. Through algorithmic analysis, we derive two conditions to guide the setting of PGAP parameters, including migration parameters, topology, and buffer sizes based on the underlying message passing communication library in order to detect and/or avoid aforementioned buffer overflows. To the best of our knowledge, our work is the first to explicitly consider the relationship between the configuration of asynchronous PGA parameters and underlying system characteristics so as to improve the reliability of asynchronous PGAs.

Experiment results showed that, with the asynchronous migration strategy, our scalable PGA is able to efficiently utilize 16,384 cores with significantly reduced communication cost. Specific strong and weak scaling tests were designed to evaluate the scalability and numerical performance of PGAP because conventional weak and strong scaling methods are not directly applicable to PGA. For example, the problem size used in conventional weak scaling is not a good indicator of the amount of computation needed to achieve a certain solution quality in PGA. PGA and sequential GA also have different