July 2017

Vectorization Strategies for Ant Colony Optimization on Intel Architectures

Victoriano Montesinos [a,1] and José M. García [a]
[a] *Computer Engineering Department, University of Murcia, 30100 Murcia, Spain*

Abstract. This paper presents an efficient parallel and vectorized implementation of three different selection functions (Roulette Wheel, I-Roulette and DS-Roulette) for tour construction (the most time-consuming part of the Ant Colony Optimization bio-inspired metaheuristic) targeting two Intel multi-core processors and the Knights Corner Intel Xeon Phi coprocessor. The results show that our best implementation (with I-Roulette as selection function) on Xeon Phi 7120P runs up to 78.98x faster compared to its sequential counterpart on a Xeon v2 CPU.

Keywords. Ant Colony Optimization, Parallel Metaheuristics, Xeon Phi, Vectorization.

1. Introduction

Ant Colony Optimization (ACO) [1] is a well-known population based metaheuristic which has been applied successfully to a wide range of *NP*-hard combinatorial optimization problems, including the Traveling Salesman Problem (TSP). ACO is a bio-inspired swarm intelligence method based on ants' foraging process and first proposed by Marco Dorigo in 1992. The technique generates solutions in a constructive way, starting with an empty solution and iteratively adding new elements. When using ACO for solving the TSP, each solution is a tuple of cities. The metaheuristic consists of two main stages: *tour construction* and *pheromone update*. In the first stage, each ant builds a path by iteratively selecting a next city among the unvisited ones. After this, pheromone update is performed, comprising two phases: *pheromone evaporation* (in order to gradually forget bad tours) and *pheromone deposit* (for reinforcing good quality solutions).

This paper presents an efficient implementation of three different selection functions for the tour construction stage (which is common to all ACO variants). Parallelization of the pheromone update stage is left as future work, as tour construction takes over 99.82 % of the time in the sequential version and the pheromone update stage is different for each ACO variant. Thus, our conclusions in this paper concern all ACO algorithms.

Firstly, we develop a parallel implementation for the default selection function (Roulette Wheel Selection) for tour construction, observing important limitations due to a poor vectorization of the code. Then, we implement on Intel architectures two other alternative algorithms that were introduced for GPUs: I-Roulette and DS-Roulette. Finally, we propose a partially vectorized implementation of Roulette Wheel Selection (named

[1] Corresponding author.
Email addresses: victoriano.montesinos@um.es (V. Montesinos), jmgarcia@um.es (J.M. García).