A Comparative Study of Synchronization of Parallel ACO on Multi-core Processor

Shigeyoshi Tsutsui
Hannan University
5-4-33, Amami-higashi, Matsubara
Osaka 580-8502, Japan
tsutsui@hannan-u.ac.jp

Noriyuki Fujimoto
Osaka Prefecture University
1-1 Gakuen-cho, Nakaku, Sakai
Osaka 599-8531, Japan
fujimoto@mi.s.osakafu-u.ac.jp

ABSTRACT

This paper proposes parallelization methods of ACO algorithms on a multi-core processor aiming at fast execution to find acceptable solutions. As an ACO algorithm, we use the *c*AS (cunning Ant System) and test on several sizes of Quadratic Assignment Problem (QAP) instances. As the parallelization method, we use agent level parallelization in one colony. According to the synchronization and exclusive control modes among threads, we propose three types of parallel ACO algorithms. Among them, that which we call the rough asynchronous parallel model shows the most promising results.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search Heuristic Methods

General Terms

Algorithms

Keywords

Evolutionary Computation; Parallel Computation; Multicore processor; Ant Colony Optimization; Quadratic Assignment Problems

1. INTRODUCTION

Recently, microprocessor vendors supply processors which have multiple cores of 2, 4, or more, and PCs which use such processors are available at a reasonable cost. Since the main memory is shared among cores, parallel processing can be performed efficiently in multi-core processor.

In this paper, we describe the parallelization of *c*AS [1], a variant of ACO algorithm, on a multi-core processor and discuss the experimental results of the parallelized *c*AS when we apply the algorithms to solving QAP, a typical *NP*-hard problem in permutation domains.

Many parallel ACO algorithms have been studied. A Brief summaries can be found in [2]. In many of the previous studies, attention is mainly focused on the parallelization using multiple colonies. In contrast to these studies, in this study parallelization is performed at the agent (or individual) level in one colony aiming at speedup of the ACO algorithm on a computing platform with a multi-core processor.

GECCO'15 Companion, July 11-15, 2015, Madrid, Spain
ACM 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764895

2. PARALLELIZATION OF *c*AS ON A MULTI-CORE PROCESSOR

In this study, parallelization is performed at the agent level in one colony. Operations for each agent are performed in parallel in one colony. In our study, a set of operations for an agent is assigned to a thread in OpenMP. Usually, the number of agents m may be larger than or equal to the core number of the platform, we generate n_{core} threads, where n_{core} is number of cores to be used. These threads execute operations for m agents of the *c*AS.

Fig. 1 shows the configuration of the parallel *c*AS. The Agant_Pool maintains agents of *c*AS. The Thread_Assignor assigns agents to Cas_Thread_1, Cas_Thread_2, …, and Cas_Thread_n_{core}. We implemented three types of parallel models as described in the following.

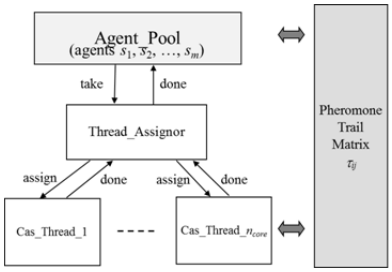

Figure 1. Parallel *c*AS on a multi-core machine.

2.1 SP-*c*AS (The synchronous parallel *c*AS)

In this model, all agents are performed in each thread, independently as shown in Fig. 2. However, in the SP-*c*AS, pheromone density updating is performed after all members finish their process in each iteration, the same way as is done in the sequential *c*AS. Thus, we call this parallel model *synchronous*. Here, pheromone density (τ_{ij}) updating is performed as follows:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t)$$
$$\Delta\tau_{ij}^{k}(t) = 1/C_{k,t} \text{ if pair } (i,j) \in \text{agent}_k,\ 0 : \text{otherwise}, \quad (1)$$

where the parameter ρ ($0 \le \rho < 1$) is the trail persistence, $\Delta\tau^{k}_{ij}(t)$ is the amount of pheromone agent$_k$ emits at iteration t, and $C_{k,t}$ is the functional value agent$_k$.

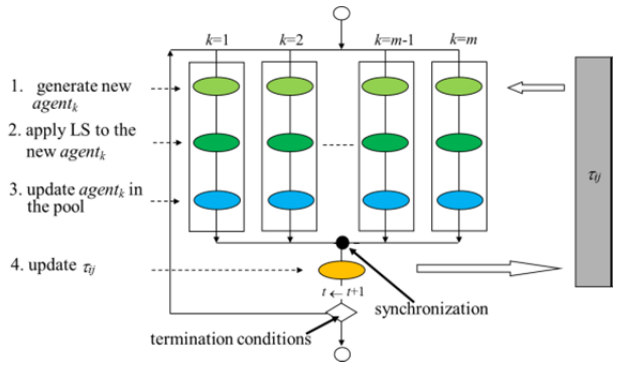

Figure 2. SP-*c*AS (The synchronous parallel *c*AS).

AP-*c*AS (The asynchronous parallel *c*AS): In SP-*c*AS, the pheromone density updating is performed after the processing of Step 3 of Fig. 2 is completed, as in usual ACO algorithms. It may cause some waiting time in the iteration in Fig 1 when there is no agent to be processed. AP-*c*AS is intended to remove this waiting time. To attain this feature, all steps of *c*AS is performed asynchronously as shown in Fig. 3. Note that the update of the iteration counter t in applies to each agent (each agent has its own iteration counter).

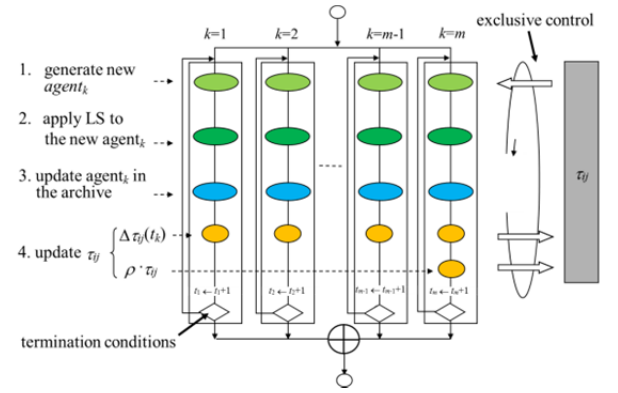

Figure 3. AP-*c*AS (The asynchronous parallel *c*AS).

Here, one problem arises in the updating pheromone density of ACO. Strictly saying, pheromone update procedure requires all agent members to be fixed. But this will undermine an asynchronous execution. To perform the pheromone update asynchronously, we modified the pheromone density updating as follows. We allow each agent to emit pheromone and increase τ^k_{ij} by Δ^k_{ij}. As for whole updating of pheromone density by multiplying the τ_{ij} by ρ, only one agent among m agents is allowed to perform. Although the above mentioned pheromone density updating is not strictly equivalent to Eq. (1), we can say that it emulates the pheromone updating process of Eq. (1).

RAP-*c*AS (The rough asynchronous parallel *c*AS): RAP-*c*AS is basically the same as AP-*c*AS. The difference is only in the pheromone density updating methods. In AP-*c*AS, the pheromone density updating is treated as a critical section. However, this causes some waiting time in accessing τ_{ij}. In RAP-*c*AS, we do not treat the pheromone density updating procedure as a critical section. We allow the process run without any exclusive control accepting access conflict to τ_{ij}. Please note here, allowing access conflict to τ_{ij} never causes any fatal troubles like in a banking system.

3. EXPERIMENTAL RESULTS

We used Xeon E5-2670v2 (Ivy Bridge), 2.5GHz, 10-core with 80GB main memory. The code was written in GNU GCC with OpenMP. We measured the performance by the average time to find acceptable solutions in successful runs in seconds (T_{avg}). 25 runs were performed in each experiment. We used the following 5 QAP instances in QAPLIB [3]; tai50b, tai60b, tai80b, tai100b, and tai150b. The numbers in the instance names show their problem size. Except for tai150b, we set their acceptable solution to known optimal solutions. We set tai150b to be within 0.2% of the known optimal solution. As for the local search, we used Tabu Search (TS). The values of the control parameter for *c*AS and TS are the same as were used in [4]. Table 1 summarizes the results. The values indicates the ratios of parallel execution time and single-core execution time.

Table 1. Results of three parallel models.

Core	Model	tai50b	tai60b	tai80b	tai100b	tai150b
2	SP	2.4	2.4	2.1	**1.9**	1.9
	AP	2.1	2.8	2.0	1.7	**2.3**
	RAP	**2.6**	**3.2**	2.1	**1.9**	**2.3**
4	SP	3.9	4.5	4.3	4.0	3.8
	AP	4.1	4.7	4.6	3.6	3.9
	RAP	**4.4**	**5.2**	**4.9**	**4.6**	**4.5**
8	SP	7.7	7.8	7.7	7.4	7.7
	AP	5.7	9.4	7.8	5.9	8.7
	RAP	**8.6**	**11.5**	**8.6**	**8.0**	**9.5**

As seen in this table, there is no significant differences between SP-*c*AS and AP-*c*AS. For example, in parallel runs with 8 cores, SP-*c*AS wins AP-*c*AS on tai50b and tai100b and AP-*c*AS wins SP-*c*AS on tai60b, tai80b, and tai150b. In parallel runs with 4-core, SP-*c*AS wins AP-*c*AS on tai50b, tai100b. On the other hand, RAP-*c*AS wins both SP-*c*AS and AP-*c*AS on all instances, though with small core number (n_{core}=2), the difference is not so prominent.

4. CONCLUSION

In this paper, we described three types of parallelization for *c*AS on a multi-core processor and discussed the experimental results of the parallelized *c*AS when we applied the algorithms to solving QAP. The results showed that the rough asynchronous parallel *c*AS (RAP-*c*AS), which uses an asynchronous parallel model without applying any critical section in access the pheromone density τ_{ij}, showed the most promising results. The results coincides with previous research in [5].

5. REFERENCES

[1] S. Tsutsui. *c*AS: Ant Colony Optimization with Cunning Ants, PPSN IX, 2006.

[2] M. Manfrin, M. Birattari, T. Stűtzle, and M. Dorigo. Parallel ant colony optimization for the traveling salesman problems, ANTS-2006, pp. 224-234, 2006.

[3] R.E. Burkard, E. Cela, S. Karisch, and F. Rendl. QAPLIB ww.seas.upenn.edu/qaplib, 2009.

[4] S. Tsutsui. ACO on Multiple GPUs with CUDA for Faster Solution of QAPs, PPSN 2012, Part II, pp. 174-184, 2012.

[5] S. Tsutsui and N. Fujimoto. Parallel Ant Colony Optimization Algorithm on a Multi-core Processor, ANTS-2010 pp. 488-495, 2010.