Multi-level Parallelization for Hybrid ACO

Omar Abdelkafi, Julien Lepagnot(✉), and Lhassane Idoumghar

LMIA, Université de Haute-Alsace (UHA), E.A. 3993, 4 rue des frères lumière, 68093 Mulhouse, France
{omar.abdelkafi,julien.lepagnot,lhassane.idoumghar}@uha.fr

Abstract. The Graphics-Processing-Unit (GPU) became one of the main platforms to design massively parallel metaheuristics. This advance is due to the highly parallel architecture of GPU and especially thanks to the publication of languages like CUDA. In this paper, we deal with a multi-level parallel hybrid Ant System (AS) to solve the Travelling Salesman Problem (TSP). This multi-level is represented by two parallel platforms. The first one is the GPU, this platform is used for the parallelization of tasks, data, solution and neighborhood-structure. The second platform is the MPI which is dedicated to the parallelization of programs. Our contribution is to use these two platforms to design a hybrid AS with a Local Search and a new heuristic.

Keywords: Parallel hybrid metaheuristics · TSP · GPU · MPI

1 Introduction

Hybrid metaheuristics [1][2][3] are one of the most efficient classes of algorithms. The idea is to combine metaheuristics [4] and other techniques for optimization. With the combination of different techniques, these methods can require a longer computation time than others. This is one of the reasons that lead the community to propose parallel hybrid metaheuristics [5]. Another reason is the evolution of highly parallel architectures like the GPU. This evolution is due to the explosion of the industry of video games and his greedy demand for graphic power. Indeed, with the advent of CUDA, the use of GPU for non-graphic applications has become easier and hybrid metaheuristics have taken advantage of this evolution.

There are many levels of parallelization. For the Ant Colony Optimization (ACO) [6] applied to the TSP in the context of a single colony, the parallel execution of ants in the tour construction phase was initiated by Bullnheimer et al. [7]. Also in this same context, in 2013, Cecilia et al. [8] used the data parallelization in the update of pheromone to get the best performance from the GPU. In the context of multiple colonies, Stutzle [9] introduced the execution of multiple colonies in parallel with cooperation between colonies to improve the quality of solutions using the parallelization of programs. In CUDA programming, the execution on GPU is conducted by the kernel. It is a code called from the CPU

© Springer International Publishing Switzerland 2014
P. Siarry et al. (Eds.): ICSIBO 2014, LNCS 8472, pp. 60–67, 2014.
DOI: 10.1007/978-3-319-12970-9_7

(the host) and duplicated on GPU (the device) to run in a parallel way. The kernel is executed in a *grid*, which is a set of *blocks* where every block is a set of *threads*.

In this work, we propose a hybrid ACO through one of the first variant of this method named the Ant System (AS) [10]. Our first contribution is to propose a new design for AS multi-colonies using GPU and MPI and the second contribution is to hybridize this method with a parallel local search (PLS) providing the intensification of the search and a new heuristic to improve results.

The rest of the paper is organized as follows. In section 2, we introduce the background needed for ACO and TSP to help the understanding of this proposition. We describe in section 3 the design of our multi-level parallel hybrid AS before we discuss the results of our experimentation in section 4. Finally, in section 5, we conclude the paper and we propose some perspective.

2 Background

The TSP is an NP-hard problem and one of the most studied combinatorial problems. It consists in finding the least-cost Hamiltonian circuit between a set of cities starting and ending with the same city. In general, TSP can be represented by a complete undirected graph G = (V, E). The set V= {1, . . . , n} is the vertex set, E=$\{(i, j) : i, j \in V, i < j\}$ is an edge set. c_{ij} is defined on E as the Euclidean distance between two vertices i and j.

Intuitively in the natural behaviors, the ants search the food randomly in the first tour construction. They move from one point to another until they find food. Once it is done, ants get back to the starting point. This corresponds to the initialization. In the search process, ants depose pheromone along the path they take. The quantity of pheromone is implemented by equation (1):

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{k} \quad \forall (i, j) \in E \tag{1}$$

where $\Delta\tau_{ij}^{k}$ is the sum of pheromone which ant k deposits when it uses the edge between i and j. It depends on the length of the tour C^k constructed by the ant k; $\Delta\tau_{ij}^{k}$ is defined in equation (2):

$$\Delta\tau_{ij}^{k} = \frac{1}{C^k} \tag{2}$$

Another characteristic of the pheromone in the natural behavior is the evaporation: the pheromone evaporates over time. This characteristic is implemented with a parameter $0 < \rho \leq 1$ in equation (3):

$$\tau_{ij} = (1 - \rho)\tau_{ij}, \quad \forall (i, j) \in E \tag{3}$$

The second step is the tour construction. In the natural behaviors, the ants follow the pheromone to find the best tour. To implement this concept, a probability is defined in equation (4) where $n_{ij} = \frac{1}{c_{ij}}$, α and β are parameters and N_i^k is feasible neighborhood. A complete survey on ACO can be found in [6].

$$\Delta p_{ij}^k = \frac{[\tau_{ij}]^\alpha [n_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [n_{il}]^\beta} \quad (4)$$

3 Design of the Parallel Hybrid ACO

The most straightforward way to design parallel AS or ACO in general is the parallelization of ants. This kind of parallelization is called the task parallelization and this is our first parallel level. The idea is very simple and used in most of the parallel ACO algorithms. Every ant is represented by a thread and every thread performs the tour construction in parallel with other ants. Inside the kernel, the ant chooses the next city to visit among the cities not selected yet and according to the probability computed by equation (4). The *CURAND library* allows the generation of a different random tour for every ant. The classical roulette wheel is used to select the next city to visit.

For the pheromone update part(see equation 1), using task parallelization can lead to concurrent access problems, i.e. if several ants update the pheromone of the same arc at the same time. The only solution in this case is to use atomic instructions but it decreases dramatically the performance. Hence, we are rather using data parallelism proposed by [8].

The level of data parallelization is used for the kernel of Update pheromone (see algorithm 1), the Evaporation pheromone (see algorithm 2) and the Update probability (see algorithm 3).

Algorithm 1. The Update pheromone kernel:

1: *Input*: **Pants**: the population of ants; **fants**: the fitness of ants; **pheromone**: the matrix of pheromone; **cities**: the size of the instance; **ants**: the size of the population;
2: Get the index of the thread idx; /*each idx represent one couple of cities*/
3: **for** i:=1 to ants **do**
4: distance = fants[i];
5: **for** j:=1 to cities **do**
6: **if** the arc between i and j == idx **then**
7: pheromone[idx]=pheromone[idx]+($\frac{1}{distance}$);
8: **end if**
9: **end for**
10: **end for**

Algorithm 2. The Evaporate pheromone kernel:

1: *Input*: **pheromone**: the matrix of pheromone;
2: Get the index of the thread idx; /*each idx represent one couple of cities*/
3: pheromone[idx]=(1-ρ) $\times$ pheromone[idx];

Algorithm 3. The Update probability kernel:

1: *Input*: **pheromone**: the matrix of pheromone; **probabilities**: the matrix of probabilities; **cij**: the matrix of distances; **cities**: the size of the instance;
2: Get the index of the thread idx; /*each idx represent one couple of cities*/
3: /*control if the cities of the couple are the same*/
4: **if** cij[idx] $\neq 0$ **then**
5: arc = $(pheromone[idx])^{\alpha} \times (\frac{1}{cij[idx]})^{\beta}$
6: all = 0
7: position = $\lfloor \frac{idx}{cities} \rfloor$ /*Get the position of the couple in the matrix*/
8: /*when j=position, cij[(position × cities)+j]=0*/
9: **for** $j \in \{0, 1, \ldots, position - 1, position + 1, \ldots, cities\}$ **do**
10: all += $(pheromone[(position \times cities) + j])^{\alpha} \times (\frac{1}{cij[(position \times cities)+j]})^{\beta}$
11: **end for**
12: probability[idx] = $\frac{arc}{all}$
13: **end if**

Our idea to hybridize ACO is to use a Parallel Local Search and a new heuristics that we name smart ants. These algorithms are added to AS, but it can be used for all the variants of ACO. The PLS is applied to a group of ants after the Tour construction. It is a classical local search but the differences are the evaluation and generation of neighborhood executed in parallel with the GPU. It consists in representing every item of the solution by a thread, which leads to a parallel execution of neighbors generation. The thread generates and evaluates the neighbor of its item and searches the best possible switch. At the end of the parallel execution, the algorithm searches the best results of all the threads. This is the third level of parallelization.

The aim of the smart ant heuristic is to improve results. It executes as much iterations as the size of the instance without considering the start city which is static and unchangeable. The figure 1 shows a small example of the heuristic using 4 cities which mean 3 iterations and every vector represents an ant. In every iteration i we search the best ant inside the colony. For example in iteration 2 of the figure 1, the best ant is the third ant which have the index 2 because it starts from 0. All the ants follow the movement of ant_2 at the position 2 indicated by the arrow in the figure 1. The city in this position for ant_2 is city number 3. By consequence, ant_0 and ant_1 move their cities to get the city 3 in position 2. This is why we name it smart ants, because they have the intelligence to adjust their tour. All the ants perform this heuristic in parallel so we use the level of parallel tasks. As we can see with this heuristic, after a certain number of iterations, all the ants have the same tour. By consequence, this heuristic leads the search to stagnancy. To escape from this stagnancy, one improvement is added. The switch is not performed when the two cities to switch are adjacent (example in figure 1 the iteration 1 for ant_0).

The last step of our approach is to use MPI to execute our method on many GPU. This step introduces a new level of parallelization: the level of parallel programs. Actually, different colonies will be executed in parallel through many processes. For example, if we execute 3 processes, we will duplicate our algorithm

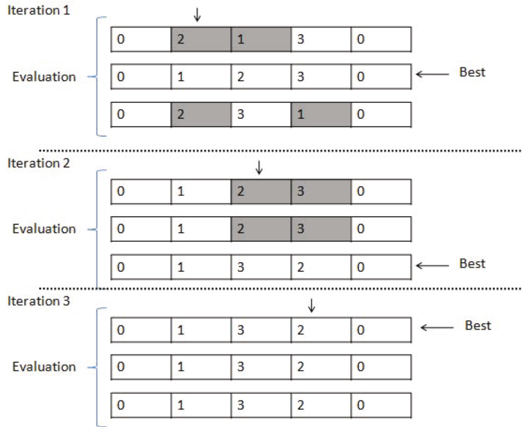

Fig. 1. Smart ants

3 times. By consequence, 3 colonies will be executed in parallel. MPI gives to our design another advantage: we can exchange information between processes in order to improve the results. To exchange information, the algorithm regularly chooses the best solution found in one process and updates the pheromone of the matrix located in the next process using a ring topology. The data parallelization is not suitable this time. Since inside the solution every city is visited only once, a new level of parallelization between cities is applied which is the solution level parallelization. For all the couple of cities used in the tour, the pheromones of these couples are updated in parallel. The atomic operation is not needed because each couple appears only once in a tour.

4 Experimental Results

4.1 Platform and Tests

In our experimentation, we use a cluster of 12 graphic cards NVIDIA Geforce GTX680. The benchmark used is a set of well known instances from the TSPLIB [11] with a size between 51 and 150 cities. All the results are expressed as a percentage deviation from the optimum. All the optimal solutions can be found in the online benchmark library TSPLIB.

4.2 The Smart Ants Heuristic

Table 1 shows the performance of the proposed algorithm with and without the smart ants (SA) heuristic for one colony. 25 tests are performed for every instance with 100 iterations. SA heuristic improves the average results of the 25 tests in the 5 instances.

Table 1. Evaluation of the smart ants heuristic

Instances	AVG with SA	AVG without SA
Eil51	**3.13%**	3.81%
Berlin52	**2.50%**	3.14%
Eil76	**5.64%**	6.35%
Pr76	**4.85%**	6.14%
KroA100	**4.67%**	5.26%

4.3 The Parallel Multiple Colonies Using MPI

We use the cluster with 12 GPU. 10 tests for each instance are performed for 10 instances from TSPLIB. Table 2 reports the best results (MIN), the worst results (MAX), the average results (AVG) and the average time required for the 10 tests. The parameters used are $\alpha = 1$; $\beta = 2$; $\rho = 0,5$. 300 iterations are executed for each colony and every one of them contains 256 ants. 12 processes are executed one per machine in the cluster. The number of colonies executed in parallel is 12. Every 10 iterations the processes exchange their best solutions using a ring topology. 60% of the average results are between 0 and 3%. From the 10 instances, 9 average results are inferior to 5%.

Table 2. The multi-level parallel hybrid AS

Instances	MIN (%)	MAX (%)	AVG (%)	Time (s)
Eil51	0.99	3.02	1.98	10.57
Berlin52	0.03	2.33	1.07	14
St70	1.53	3.2	2.61	25.96
Eil76	2.83	5.04	4.21	28.02
Pr76	1.99	3.55	2.76	31.44
Rat99	3.53	7.8	6.19	35.7
KroA100	2.41	3.61	3.17	55.92
Bier127	1.25	2.58	1.87	87.72
Ch130	1.86	3.11	2.51	72.2
Ch150	2.84	3.75	3.42	76.8

The next experiment has the aim to see the behavior of the cluster when the objective function is evaluated equally between one GPU and 8 GPU. In this experiment, the same number of ants is used in the two cases. Table 3 presents the average results of 10 tests for 4 instances. AVG 1 is the average for the first case, AVG 2 is the average for the second case and ACC is the acceleration of the cluster compared to one GPU. With these conditions, the parallel design with the cluster improves the results and gives accelerations between 1.22 and 1.84 times compared to one GPU.

The final experiment is to compare our approach to other methods from the literature. In table 4, works from the literature are used for the evaluation. 4 approaches are selected. [12] is an ACO algorithm for TSP and [13][14][15] are other approaches to solve TSP for 5 instances. [*] is our approach and the results are the percentage deviation from the optimum. The Friedman test [16], performed on these 5 problems with $\alpha = 5\%$, shows that we can reject the null hypothesis, i.e. there is at least one algorithm whose performance is different

Table 3. MPI accelerations

Instances	AVG 1(%)	AVG 2 (%)	ACC
Berlin52	2.72	**1.04**	×1.73
Pr76	5.4	**3.08**	×1.40
Bier127	2.53	**2.18**	×1.22
Ch150	4.43	**3.91**	×1.84

from at least one of the other algorithms. To know which algorithms are different, we perform paired comparisons. The critical value is C=3.67. The paired comparisons (see Table 5) show that the results obtained by [*] are different from those obtained by the four other approaches. From the above analysis, we can see that our hybrid algorithm is better and outperforms the other four metaheuristics.

Table 4. Literature comparison

Instances	[*]	[12]	[13]	[14]	[15]
Eil51	**1.98**	7.98	2.89	2.69	3.43
Berlin52	**1.07**	7.38	7.01	5.18	5.81
Eil76	4.21	12.08	4.35	**3.41**	5.46
Bier127	**1.87**	15.32	3	2.2	3.41
Ch130	**2.51**	24.15	2.82	2.82	2.82

Table 5. Paired comparisons

Instances	[12]	[13]	[14]	[15]
[*]	**19**	**10**	**4**	**12**
[12]	-	9	15	7
[13]	-	-	6	2
[14]	-	-	-	8

5 Conclusion and Perspectives

This work has two main objectives. The first one is to design a parallel ACO which can run in a cluster of GPU. The second objective is to improve the quality of solutions and to be as close as possible to the global optimum.

For the first objective, we use Five levels of parallelization. The first one is the parallelization of tasks performed by the GPU, which helps us to parallelize ants for the tour construction and the smart ant heuristic. The second level is the parallelization of data performed by GPU, which help us to update and evaporate the pheromones and to update the probabilities. The third level is the parallelization of the neighborhood structure performed also by GPU. This level is essentially used to parallelize the neighborhood inside the PLS. The fourth level is the solution level parallelization, performed by the GPU and used to update the pheromone when the best solution is exchanged between colonies. Finally, the last level is the parallelization of programs performed by MPI. It

allows us to parallelize different colonies and to diversify the search as much as possible. For the second objective we hybridize the AS: we use two techniques. The first one is to add the PLS for the intensification of the search. The second technique is to test a new heuristics named smart ant to improve results.

In our future works, we plan to apply the proposed algorithm to other combinatorial problems like the quadratic assignment problem. Another perspective is to reuse the same design for other swarm intelligence methods like the particle swarm optimization.

References

1. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing **11**(6), 4135–4151 (2011)
2. Lepagnot, J., Idoumghar, L., Fodorean, D.: Hybrid Imperialist Competitive Algorithm with Simplex approach: Application to Electric Motor Design, In: 2013 IEEE International Conference on Systems Man and Cybernetics (SMC) pp. 2454–2459, Manchester UK (October 2013)
3. Aouad, M.I., Idoumghar, L., Schott, R., Zendra, O.: Sequential and Distributed Hybrid GA-SA Algorithms for Energy Optimization in Embedded Systems, In: The IADIS International Conference Applied Computing 2010, pp. 167–174 (2010)
4. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys **35**, 268–308 (2003)
5. Cotta, C., Talbi, E.G. Alba, E.: Parallel Hybrid Metaheuristics, in Parallel Metaheuristics: A New Class of Algorithms. John Wiley and Sons (2005)
6. Dorigo, M., Stutzle, T.: Ant Colony Optimization. Bradford Company, USA (2004)
7. Bullnheimer, B., Kotsis, G., Strauss, C.: Parallelization strategies for the ant system. Applied Optimization **24**, 87–100 (1997)
8. Cecilia, J.M., Garcia, J.M., Nisbet, A., Amos, M., Ujaldon, M.: Enhancing data parallelism for Ant Colony Optimization on GPUs. J. Parallel Distrib. Comput. **73**, 42–51 (2013)
9. Stützle, Thomas: Parallelization Strategies for Ant Colony Optimization. In: Eiben, Agoston E., Bäck, Thomas, Schoenauer, Marc, Schwefel, Hans-Paul (eds.) PPSN 1998. LNCS, vol. 1498, p. 722. Springer, Heidelberg (1998)
10. Dorigo, M.: Optimization: learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Italy (1992)
11. Reinelt, G.: TSPLIB - A Traveling Salesman Problem Library. ORSA Journal on Computing **3**(4), 376–384 (1991)
12. Chirico, U.: A java framework for ant colony systems, Technical report, Siemens Informatica S.p.A (2004)
13. Cochrane, E.M., Beasley, J.E.: The co-adaptive neural network approach to the Euclidean traveling salesman problem. Neural Networks **16**(10), 1499–1525 (2003)
14. Masutti, T.A.S., Castro, L.N.D.: A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. Information Sciences **179**(10), 1454–1468 (2009)
15. Somhom, S., Modares, A., Enkawa, T.: A self-organizing model for the traveling salesman problem, Journal of the Operational Research Society, 919–928 (1997)
16. Idoumghar, L., Chérin, N., Siarry, P., Roche, R., Miraoui, A.: Hybrid ICA-PSO algorithm for continuous optimization. Applied Mathematics and Computation **219**, 11149–11170 (2013)