Parallel and Distributed Implementation Models for Bio-inspired Optimization Algorithms

Hongjian Wang(✉) and Jean-Charles Créput

IRTES-SeT, Université de Technologie de Belfort-Montbéliard,
90010 Belfort, France
hongjian.wang@utbm.fr

Abstract. Bio-inspired optimization algorithms have natural parallelism but practical implementations in parallel and distributed computational systems are nontrivial. Gains from different parallelism philosophies and implementation strategies may vary widely. In this paper, we contribute with a new taxonomy for various parallel and distributed implementation models of metaheuristic optimization. This taxonomy is based on three factors that every parallel and distributed metaheuristic implementation needs to consider: *control*, *data*, and *memory*. According to our taxonomy, we categorize different parallel and distributed bio-inspired models as well as local search metaheuristic models. We also introduce a new designed GPU parallel model for the Kohonen's self-organizing map, as a representative example which belongs to a significant category in our taxonomy.

Keywords: Parallel and distributed computing · Metaheuristic · Genetic algorithm · Ant colony optimization · Self-organizing map

1 Introduction

In the combinatorial optimization community, there exist a number of different bio-inspired optimization metaheuristics, such as genetic algorithms (GA), ant colony optimization (ACO), and artificial neural networks (ANN). Inspired by natural systems and designed to mimic certain phenomena or behaviors of biology, these algorithms aim at finding, as optimally as possible, approximate solutions to real-life difficult problems which are usually not able to be solved by exact approaches in reasonable computing time. Biologic systems are usually made up of populations of simple individuals, ants, birds or neurons, interacting locally with one another and with their environment. This trait should imply some potential for parallel and distributed implementations of the derived bio-inspired optimization algorithms. However, the implantation, from nature to practical parallel and distributed computational systems, is not as smooth as it looks like, owing to various restrictions of the latter, coming from 1) resource sharing and competition, 2) communication and synchronization among computing nodes, 3) system robustness requirement. As a result, gains from different parallelism philosophies and implementation strategies may vary widely,

P. Siarry et al. (Eds.): ICSIBO 2014, LNCS 8472, pp. 68–79, 2014.
DOI: 10.1007/978-3-319-12970-9_8

and it is very tricky to come out with a consummate model. Trying to cast some interesting insights on this issue, this paper firstly contributes with a new taxonomy for various parallel and distributed implementation models of metaheuristic optimization, and then categorizes different bio-inspired models as well as local search metaheuristic models according to our taxonomy, including the introduction of a new designed GPU parallel model for the Kohonen's self-organizing map (SOM) [1], as a representative of "control distributed, data decomposition, shared memory" category.

The rest of this paper is organized as follows. Section 2 describes the proposed taxonomy with three factors. According to this new taxonomy, Section 3 categorizes some parallel and distributed implementation models of metaheuristic optimization algorithms, including GA, ACO, SOMANN, and local search. A new designed GPU parallel model for SOMANN is also introduced in Section 3. The partly distributed model and the fully distributed model are discussed in Section 4 before some conclusions of this work are drawn in Section 5.

2 Taxonomy for Parallel and Distributed Strategies

Generally, parallel computing speeds up computation by dividing the work load among a certain amount of processors. In the parallel computing community, two main sources of parallelism which are well accepted are *data parallelism* and *control parallelism* [2,3]. Data parallelism refers to the execution of the same operation or instruction on multiple large data subsets at the same time [2]. This is in contrast to control parallelism (or task parallelism, or function parallelism, or operation parallelism), which refers to the concurrent execution of different tasks allocated to different processors, possibly working on the "same" data and exchanging information [3]. Parallel computation based on these two parallelisms is particularly efficient when algorithms manipulate data structures that are strongly regular, such as matrices in matrix multiplications. Algorithms operating on irregular data structures or on data with strong dependencies among the different operations remain difficult to parallelize efficiently and to characterize comprehensively, using only data or control parallelism. Metaheuristics generally belong to this category, and parallelizing them offers opportunities to find new ways to use parallel and distributed computational systems and to design parallel algorithms [4]. In our opinion, the traditional dual classification for general parallel computing looks inadequate when dealing with the various parallel and distributed optimization metaheuristics. One important point that should be emphasized concerns the allocation of processors and memory according to the instance size of the problem. We think this point, specific to optimization, should be alighted in the taxonomies of parallel and distributed metaheuristic implementations, since it determines the maximum size of the input that could be solved in systems on hand *and* how the performance should grow according to the amount of physical cores and memory.

We propose a new taxonomy as shown in Fig. 1. It is based on the three factors that every parallel and distributed implementation model of metaheuristic

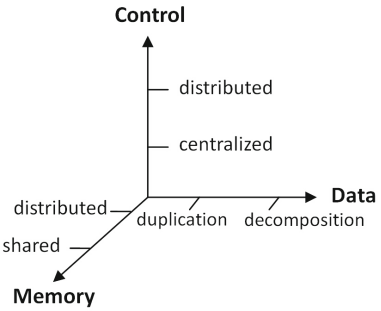

Fig. 1. Taxonomy based on *control*, *data*, *memory*

optimization needs to consider: *control*, *data*, and *memory*. Note that though the two terms of our taxonomy are literally similar to the traditional parallelism classification, they stand for very different considerations.

— *Control.* This term is about algorithmic organization and its corresponding execution pattern on parallel processors. Some parallel and distributed implementation models are based on centralized control on different levels. The most common case is the so called "master-slave" model, as shown in Fig. 2(a), in which a *master* process manages the population and hands out individuals to evaluate to a number of *slave* processes. After the evaluation, the master process iteratively collects the results and applies some global operations, such as selection, to produce the next generations. Ergo in this case, the master process plays a central role while the slave processes act as co-processors to accelerate computation. In out taxonomy, we call this kind of implementation model "control centralized". The opposite implementation model should be under a completely distributed control pattern, without depending on any central control that would break the entire computing network if it was suppressed from the computation implementation, as the cellular model shown in Fig. 2(b). Thus the robustness can be guaranteed because the computation can continue even when some computing units fall down. We call this kind of implementation model "control distributed".

— *Data.* This term denotes the input problem data, of size N, and the representation of the solution. The size of the solution could generally be $O(N)$ since it is in relation to the input. However, the size might depend on optimising operations and the implementation choices of designers. Some algorithms perform metaheuristic exploration and exploitation within a set of solutions (population), handling each solution in parallel, and then select the best-so-far solution iteratively. Implementation models of this kind are built upon "data duplication" and the required memory is with $O(NM)$ where M is the population size. Alternatively, other algorithms generate every part of the whole solution separately in parallel. The final solution can be then obtained by combining together partial results from all the processors. Hence implementation models of this kind are founded on "data decomposition" and their memory employment could remain

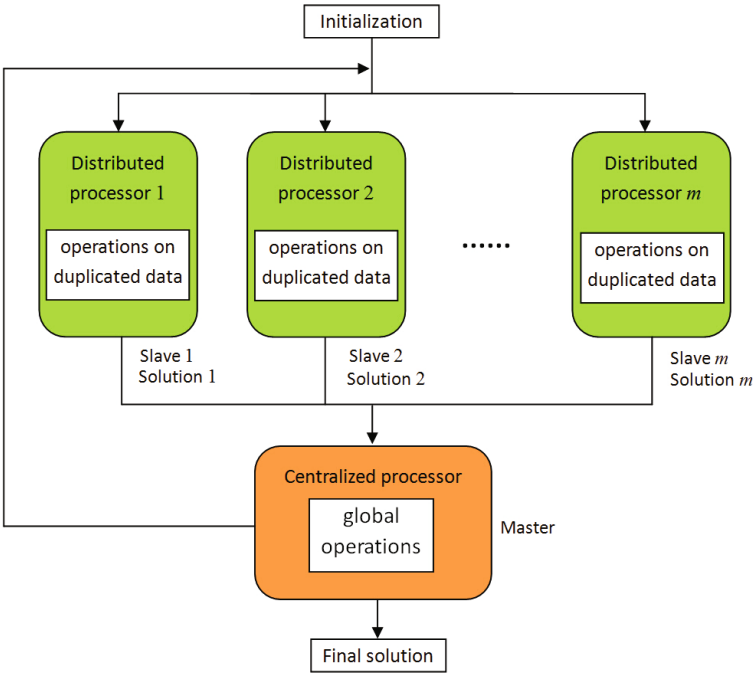

Fig. 2. Comparison between (a) "master-slave" model and (b) cellular model. The parallel "master-slave" model is under "control centralized, data duplication" pattern while the cellular model is under "control distributed, data decomposition" pattern.

in $O(N)$. This linear relationship to the problem size makes these models able to handle larger scale problems with limited physical memory, than the models under "data duplication" pattern.

— *Memory.* This term concerns concrete implementations on different parallel and distributed computing platforms. Two commonly used categories are "shared memory" and "distributed memory", and we adopt them in our taxonomy. Normally, if the considered algorithm is implemented in shared memory systems, then it usually suffers from memory access contention, especially if global memory access is through a single path such as a bus. Cache memory alleviates the problem but it does not solve it. On the other hand, if the considered algorithm is implemented in distributed memory machines, then it has better scaling behavior, which means that the performance is relatively unaffected if more processors (and memory) are added and larger problem instances are tackled. The information exchange among different processors is via message passing mechanism. As a result, the communication bottleneck of distributed memory computing systems usually becomes the main obstacle to high performance of the "distributed memory" implementation models.

With our taxonomy in hand, any parallel and distributed implementation model of metaheuristic optimization can be classified and analyzed based on the three factors. By doing so, the employment of processors and memory according to the problem size can be predicted *and* the possible performance bottlenecks could also be forecasted. For example, most of the parallel and distributed GA implementations under "master-slave" model are based on "control centralized, data duplication", as shown in Fig. 2(a). Then the amount of processors needed is with $O(M)$ where M is the population size *and* the required memory is with $O(NM)$ where N is the problem size. If an implementation is in shared memory computing systems, for example on the GPU CUDA platform, then a lot of attention should be paid on the *global memory* access efficiency and contention. Note that when the input size N grows, the solution occupies a larger part of the central memory limiting the use of processors. Consequently with a fixed memory size, the number of used processors should decrease as the input size increases. On the other hand, implementations under coarse-grained models, where the radio of computation to communication is high, are more adapted to distributed memory computing systems, such as clusters. This is the case of cellular GA implementation model [5] that is based on "distributed memory, data duplication".

3 Categorizing Different Implementation Models

In this section, we consider and categorize some bio-inspired metaheuristics, including GA, ACO, and SOMANN. Implementation models of other parallel metaheuristics, such as local search, are also classified and analyzed in our taxonomy.

3.1 Parallel Genetic Algorithms

GAs are search algorithms inspired by genetics and natural evolutionary principles. The most important operations in GAs are reproduction, mutation, fitness evaluation and selection (competition). There are several possible levels at which GAs can be parallelized: the fitness evaluation level, the individual level or the population level [5]. Parallelization at the fitness evaluation level is usually implemented under "master-slave" model, in which each individual fitness is evaluated simultaneously on a different processor. This architecture belongs to the "control centralized, data duplication" category according to our taxonomy, and it can be implemented on both shared memory multiprocessors as well as distributed memory machines.

Individual or population-based parallel approaches for GAs introduce additional terms that should be considered, such as *deme*, *migration* and *topology* [6]. These approaches are inspired by the observation that natural population tends to possess a spatial structure. The two important spatial structure based categories are the *island* and the *cellular* models. The island model [7] features geographically separated subpopulations of relatively large size. Subpopulations may exchange information from time to time by allowing some individuals to migrate from one subpopulation to another according to various patterns. In the cellular model [8], individuals are placed on a large toroidal one or two-dimensional grid, one individual per grid location. Fitness evaluation is done simultaneously for all individuals, and selection, reproduction and mating take place locally within a small neighborhood. From an implementation point of view, these two kinds of models are often adapted to distributed memory systems [9,10] and accordingly they are classified into the "control distributed, data duplication, distributed memory" category according to our taxonomy.

3.2 Parallel Ant Colony Optimization

As early as when Dorigo [11] initially proposed ACO, he suggested the application of parallel computing techniques to enhance both the ACO search and its computational efficiency. A comprehensive survey on parallel ACO can be found in [12]. Among various parallel ACO implementations, the "master-slave" model has been quite popular in the research community, mainly due to the fact that this model is conceptually simple and easy to implement. According to Pedemonte et al. [12], the "master-slave" model is further divided into three distinguished subcategories regarding the *granularity*. The standard implementation of *coarse-grain master-slave* ACO assigns one ant to a slave that is executed on an available processor. The master globally manages the global information (i.e. the pheromone matrix, the best-so-far solution, etc.), and each slave builds and evaluates a single solution. The communication between the master and slaves usually follows a synchronous model. This kind of implementation model is under "control centralized, data duplication" pattern. In the *medium-grain master-slave* model, a domain decomposition of the problem is applied. The slaves solve each subproblem independently, whereas the master manages the

overall problem information and constructs a complete solution from the partial solutions reported by the slaves. Furthermore, in the *fine-grain master-slave*, the slaves perform minimum granularity tasks, such as processing single components used to construct solutions, or parallel evaluation of solution elements. These two kinds of implementation models are under "control centralized, data decomposition" pattern and they can be implemented both in shared memory systems and in network of workstations or clusters, with each node having independent memory. Frequent communications between the master and slaves are usually required in these models, and this issue is more severe when they are implemented in distributed memory systems than shared memory systems.

There exist other parallel and distributed ACO implementation models that are under "control distributed" pattern. In the *cellular* model [12,13], a single colony is structured in small neighborhoods, each one with its own pheromone matrix. Each ant is placed in a cell in a toroidal grid, and the trail pheromone update in each matrix considers only the solutions constructed by the ants in its neighborhood. In the *multicolony* model [12,14], several colonies explore the search space using their own pheromone matrices. The cooperation is achieved by periodically exchanging information among the colonies. In the *parallel independent runs* model [12,15,16], several sequential ACOs, using identical or different parameters, are concurrently executed on a set of processors. The executions are completely independent, without communication among the ACOs, therefore the model does not consider cooperation between colonies. The latter two models have distributed controlling at colony level. These three models above are all under "data duplication" pattern and they can be implemented in both shared memory [16] and distributed memory [13] systems.

3.3 Parallel Self-Organizing Map Artificial Neural Networks

Partly motivated by how visual, auditory or other sensory information is handled in separate parts of the cerebral cortex in the human brain, the Kohonen's SOM [1] is a prominent unsupervised ANN model providing a topology-preserving mapping from a high-dimensional input space onto a two-dimensional map space. Some methods for computing SOM on GPU have been proposed [17,18]. These methods accelerate SOM process by parallelizing the inner steps at each basic iteration, firstly, to find out the winner neuron in parallel, secondly, to move the winner neuron and its neighbors in parallel. Consequently these kinds of implementation models fall into the "control centralized" category.

In our opinion, one interesting model for parallel SOM should be attributed to the "control distributed, data decomposition, shared memory" category, in that, firstly, distributed control guarantees the model's robustness, secondly, data decomposition eases the burden of massive memory usage when dealing with large-scale problems, and thirdly, shared memory reduces the communication costs among different processing units and allows easy implementation on Graphics Processing Unit (GPU) like systems. Given this ambition, we have designed a novel parallel SOM model and implemented it on GPU Compute Unified Device Architecture (CUDA) platform, trying to deal with large scale

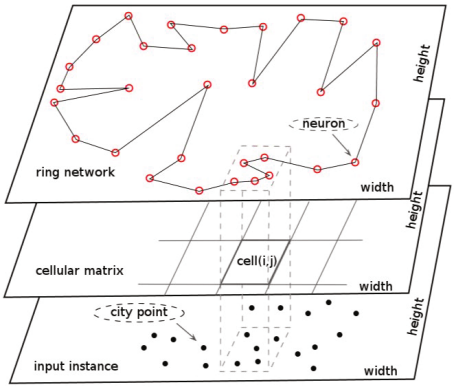

Fig. 3. Parallel cellular model: the input data density distribution, the cellular matrix and the neural network. To a given cell of the cellular matrix corresponds a constant part of the input data as well as a part of the neural network made up of SOM's topological grids/neurons.

travelling salesman problems (TSP) [19]. As illustrated in Fig. 3, three main data structures are used to implement the parallel model. Between the neural network and the input data, we add a uniform two-dimensional cellular matrix with linear relationship to the input size, as a level of decomposition of the plane and the input data. Its role is to memorize the neurons in a distributed fashion and authorize many parallel closest point searches in the plane by a spiral search algorithm [20,21], and then many parallel training procedures. Each uniformly sized cell in the cellular matrix is a basic training unit and will be handled by one parallel processor/GPU thread. Thus, the model proceeds from a cellular decomposition of the input data, in Euclidean space, such that each processor represents a constant and small part of data. Therefore, according to the increase of parallel processors in the future, this approach should be more and more competitive, while at the same time being able to deal with very large size inputs. This quintessential property holds because of the linear memory and processors needed according to the input size. More design details and experimental results of the parallel SOM model can be found in [19].

3.4 Parallel Local Search

Local search is a metaheuristic algorithm which could be viewed as "walks through neighborhoods". The walks are performed by iterative procedures that allow moving from one solution to another, through the solution domains of the problems at hand. Parallelism naturally arises when dealing with a neighborhood, since each of the solutions belonging to it is an independent unit. This kind of parallelization is called *iteration-level parallel model*, a low level "master-slave" model in which evaluation of the neighborhood is made in parallel [22,23]. At the beginning of each iteration, the master duplicates the cur-

rent solution among parallel nodes. Each of them manages a number of candidates, and the results are returned to the master. This implementation model is obviously under "control centralized, data duplication" pattern. In [23], Luong et al. have re-designed the above model on GPU platform. Considering a neighbor as a slight variation of the candidate solution which generates the neighborhood, they only copy the representation of this candidate solution from CPU to GPU. Then N^2 threads are employed to carry out the parallel *2-opt* moves and evaluations, where N is the TSP instance size. Each parallel evaluation only deals with the slight variation based on the candidate solution, with the help of a neighborhood mapping which locates each thread's corresponding variation position in the solution representation. Then the fitness results generated by parallel threads need to be gathered and selected for a best one, which will become the new starting solution, called "pivot", at the next local search iteration. The solution representation and the fitness structure are stored in the global memory of GPU. From the above, it can be concluded that this strategy is under "control centralized, data decomposition, shared memory" pattern.

Other two major parallel models for local search can be distinguished as *solution-level* and *algorithmic-level*. In the solution-level parallel model, the focus is on the parallel evaluation of a single solution *and* the function can be viewed as an aggregation of partial functions. Implementations based on this model are under "control centralized, data decomposition" pattern. In the algorithmic-level parallel model, several local search metaheuristics are simultaneously launched for computing robust solutions. The well-known multistart local search, in which different local search algorithms are launched using diverse initial solutions, is an instantiation of this model [22]. Implementations based on this model are under "control centralized, data duplication" pattern. In our opinion, centralized selection procedures among parallel processors are inevitable, as long as each processor deals with a whole solution. Differently, an interesting model should be fully distributed, where each processor carries out its own local search based on part of the input data, generating one part of the whole solution. Operations on different processors are completely independent with each other *and* no centralized selection procedure is needed. Eventually, a final solution can be obtained by combining all the partial results from different processors. Ergo this implementation model of local search is under "control distributed, data decomposition" pattern, as shown in Fig. 2(b), and it is supposed to be able to solve very large challenging problems, such as the World TSP Challenge, in distributed computing systems such as clusters.

4 Partly Distributed Model vs. Fully Distributed Model

In literature, many implementation models are labeled as "distributed model". Actually, some of them belong to the "control centralized, distributed memory" category according to our taxonomy while others belong to the "control distributed, shared memory" category. In our opinion, these two kinds of implementation models are only *partly distributed*, or distributed in a weak sense.

For example, even if the "master-slave" model is implemented in distributed memory systems with computing nodes communicating by message transfers, the master process necessarily deals with specific data structures different from the slave data structures. We think only the implementation model based on "control distributed, distributed memory" is *fully distributed*, or distributed in a strong sense. No component has special role in this kind of implementation and it could be carried out on networks of stations, or processors, communicating by message transfers, and with all processors executing the same code.

From our point of view, a very significant conceptual implementation model should be under "control distributed, data decomposition, distributed memory" pattern, because it is fully distributed and makes possible to solve very large problems in distributed computing networks. In literature, we found one example which belongs to this category and it was proposed by Nguyen et al. in [24]. They applied an effective implementation of hybrid GA incorporating Lin-Kernighan heuristic, to the 1,904,711-city World TSP Challenge. They divided the world instance into a number of smaller subinstances and then applied PHGA to these subinstances. Finally, they reconnected all the best segments of each subinstance to form a new best tour for the world instance. This example, however, has a high level of granularity since each processor deals with a significant part of the input data using a hybrid GA incorporating Lin-Kernighan heuristic. Based on the same requirement of data decomposition, we have also designed a cellular SOM model to the TSPs, as introduced in this paper. However, in our current work, we implement this model on GPU CUDA platform with global memory, which makes the implementation partly distributed and belong to the "control distributed, data decomposition, shared memory" category. This model however has very low level of granularity with few input data assigned to each processor. Executing low-level granularity models based on data decomposition in distributed memory systems means an important challenge.

5 Conclusion

Parallel and distributed metaheuristics offer the possibility to address large scale problems which are often intractable to traditional sequential algorithms. A good way of formulating, analyzing and classifying different parallel and distributed implementations will be very helpful in designing efficient, scalable, and robust algorithms in return. One important point that should be emphasized concerns the allocation of processors and memory according to the problem size. We think this point, specific to optimization, should be alighted in the taxonomies of parallel and distributed metaheuristic implementation models, since it determines the maximum size of the problem that could be solved in systems on hand *and* how the performance should grow according to the amount of physical cores and memory. With this in mind, we have proposed a new taxonomy and categorized different bio-inspired metaheuristic implementation models according to this taxonomy. Also we have contributed with a new designed GPU parallel model for SOMANN. Furthermore, we have discussed partly distributed models

and fully distributed models, in weak and strong senses. We hope the efforts made in this paper will help others, particularly designers and engineers who want to use bio-inspired optimization algorithms for large scale complex problems, choose the right parallelization model for their applications.

References

1. Kohonen, T.: Self-organizing maps, vol. 30. Springer (2001)
2. Freitas, A.A., Lavington, S.H.: Data parallelism, control parallelism, and related issues. In: Mining Very Large Databases with Parallel Processing, pp. 71–78. Springer (2000)
3. Crainic, T.G., Toulouse, M.: Parallel meta-heuristics. In: Handbook of Metaheuristics, pp. 497–541. Springer (2010)
4. Crainic, T.G., Toulouse, M.: Parallel strategies for meta-heuristics. Springer (2003)
5. Tomassini, M.: Parallel and distributed evolutionary algorithms: A review (1999)
6. Konfrst, Z.: Parallel genetic algorithms: Advances, computing trends, applications and perspectives. In: Proceedings. 18th International Parallel and Distributed Processing Symposium, p. 162. IEEE (2004)
7. Cohoon, J.P., Hegde, S.U., Martin, W.N., Richards, D.: Punctuated equilibria: a parallel genetic algorithm. In: Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, July 28-31. Massachusetts Institute of Technology, L. Erlhaum Associates, Cambridge, Hillsdale (1987)
8. Manderick, B., Spiessens, P.: Fine-grained parallel genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 428–433. Morgan Kaufmann Publishers Inc. (1989)
9. Andre, D., Koza, J.R.: Parallel genetic programming: A scalable implementation using the transputer network architecture. In: Advances in Genetic Programming, pp. 317–337. MIT Press (1996)
10. Folino, G., Pizzuti, C., Spezzano, G.: A scalable cellular implementation of parallel genetic programming. IEEE Transactions on Evolutionary Computation **7**, 37–53 (2003)
11. Dorigo, M.: Optimization, Learning and Natural Algorithms. PhD thesis, Politecnico di Milano (1992)
12. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Applied Soft Computing **11**, 5181–5197 (2011)
13. Pedemonte, M., Cancela, H.: A cellular ant colony optimisation for the generalised steiner problem. International Journal of Innovative Computing and Applications **2**, 188–201 (2010)
14. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. Journal of Parallel and Distributed Computing **62**, 1421–1432 (2002)
15. Stützle, T.: Parallelization Strategies for Ant Colony Optimization. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 722–731. Springer, Heidelberg (1998)
16. Bai, H., OuYang, D., Li, X., He, L., Yu, H.: Max-min ant system on gpu with cuda. In: 2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC), pp. 801–804. IEEE (2009)
17. McConnell, S., Sturgeon, R., Henry, G., Mayne, A., Hurley, R.: Scalability of self-organizing maps on a gpu cluster using opencl and cuda. Journal of Physics: Conference Series **341**, 012018 (2012)

18. Yoshimi, M., Kuhara, T., Nishimoto, K., Miki, M., Hiroyasu, T.: Visualization of pareto solutions by spherical self-organizing map and its acceleration on a gpu. Journal of Software Engineering and Applications 5 (2012)
19. Wang, H., Zhang, N., Créput, J.-C.: A Massive Parallel Cellular GPU Implementation of Neural Network to Large Scale Euclidean TSP. In: Castro, F., Gelbukh, A., González, M. (eds.) MICAI 2013, Part II. LNCS, vol. 8266, pp. 118–129. Springer, Heidelberg (2013)
20. Bentley, J.L., Weide, B.W., Yao, A.C.: Optimal expected-time algorithms for closest point problems. ACM Transactions on Mathematical Software (TOMS) **6**, 563–580 (1980)
21. Créput, J.C., Koukam, A.: A memetic neural network for the euclidean traveling salesman problem. Neurocomputing **72**, 1250–1264 (2009)
22. Talbi, E.G.: Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons (2009)
23. Van Luong, T., Melab, N., Talbi, E.G.: Gpu computing for parallel local search metaheuristic algorithms. IEEE Transactions on Computers **62**, 173–185 (2013)
24. Nguyen, H.D., Yoshihara, I., Yamamori, K., Yasunaga, M.: Implementation of an effective hybrid ga for large-scale traveling salesman problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **37**, 92–99 (2007)