SCIENCE CHINA
Information Sciences

• HIGHLIGHT •

June 2017, Vol. 60 068102:1–068102:3
doi: 10.1007/s11432-015-0594-2

Dynamic strategy based parallel ant colony optimization on GPUs for TSPs

Yi ZHOU[1,2], Fazhi HE[1*] & Yimin QIU[3]

[1]*State Key Laboratory of Software Engineering, School of Computer Science, Wuhan University, Wuhan* 430072*, China;*
[2]*School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan* 430081*, China;*
[3]*School of Information Science and Engineering, Wuhan University of Science and Technology, Wuhan* 430081*, China*

Received August 13, 2016; accepted September 12, 2016; published online February 27, 2017

Citation Zhou Y, He F Z, Qiu Y M. Dynamic strategy based parallel ant colony optimization on GPUs for TSPs. Sci China Inf Sci, 2017, 60(6): 068102, doi: 10.1007/s11432-015-0594-2

Metaheuristics are a type of approximate optimization algorithms for solving hard and complex problems in science and engineering [1]. They can be defined as algorithm templates that can be easily adapted to solve specific optimization problems. Motivated by search behavior, most researchers divide metaheuristics into two classes: trajectory-based and population-based [1]. Ant colony optimization (ACO) is a population-based metaheuristic inspired by the social behavior of ants [2]. In ACO, one of the most significant features is positive feedback, which benefits from the pheromone left by ants that can guide the exploration process. ACO has been used to solve NP-hard problems, such as the traveling salesman problem (TSP) and the quadratic assignment problem (QAP). ACO application areas include network routing, molecular modeling, data mining, etc.

In recent years, new hardware that delivers massive amounts of parallel-processing power, e.g., the Cell Broadband Engine, field-programmable gate arrays, and graphics processing units (GPUs), has become available. Using classic parallel models on a GPU platform is a topic of considerable interest [3]. Using GPUs, the computational efficiency of ACO algorithms is significantly improved [4–6]. However, due to GPU resource limitations, the largest number of TSP cities in previous GPU benchmarks has been 2392 [4–6]. Parallel strategies suitable for TSPs larger than 2392 cities remain unknown.

Our contribution. This article presents ideas to optimize existing GPU-based ACO algorithms for the TSP. We extend previous GPU-based ACO algorithms in two aspects: problem scale and computational efficiency. To solve larger problems, we present and evaluate two kernel strategies. To fully exploit GPU computing power, we propose a new algorithm in the tour-construction stage. Our major contributions are as follows:

(1) We propose suitable kernel strategies for solving problems on different scales. Two kernel strategies in the tour-construction stage, KE-ALL and KE-ONE, are presented and evaluated.

(2) We present a new parallel implementation of roulette-wheel selection, named Tiling Roulette (TR), on a GPU.

(3) We design a new algorithm in the tour-construction stage named single-instruction, multiple-data (SIMD)-oriented tour construction (STC) and implement this algorithm on a GPU with a novel dynamic work-group strategy (ST-DYNAMIC).

* Corresponding author (email: fzhe@whu.edu.cn)

The authors declare that they have no conflict of interest.

© Science China Press and Springer-Verlag Berlin Heidelberg 2017

(4) We evaluate our algorithm with the standard TSP library (TSPLIB) for a large range of problems with as many as 4461 cities; a range of this size has not been reported in previous GPU-based ACO public references [4–6]. We can obtain a speedup factor of 44x compared to the CPU counterpart [2] with the same solution quality.

(5) We also compare our algorithm with the existing data-parallel implementation of ACO on a GPU by Cecilia et al. [5] within 2392 cities.

Methodology. We refer the interested reader to the supplementary file for the detail of ACO algorithm and our GPU-based approaches. We introduce two novel optimization methods as follows.

(1) Optimization of Roulette-Wheel selection. The roulette-wheel selection can be parallelized using the parallel prefix-sum and parallel search methods. We optimize this process with a method called Tiling Roulette.

We present the pseudo-code of our approach in Algorithm 1. First, we divide the probability array into t ($t = n/$tile_size) equal-sized tiles, and the tile size is equal to the work-group size. Second, we perform a parallel scan operation sequentially from tile 0 to tile $t-1$. If the sum of a tile is larger than the random value rand_value generated previously, this tile is selected. Third, the work-items search in parallel for the first element in the selected tile that is larger than rand_value; they later write the result to selected_id.

Algorithm 1 Pseudo-code of the TR selection process

Input: prob: probability array; prob_sum: sum of the probability array;
Output: selected_id: result city index of the selection;
1: __local float tile[tile_size];
2: __local int selected_id;
3: __private float tile_item, tile_item_scanned, tile_sum = 0;
4: int $i = 0$, tid = get_local_id(0);
5: float rand_value = rand01() * prob_sum;
{For each item in a tile, do parallel scan. The tiles are processed serially.}
6: **for** i=tid; $i < n$; i+=tile_size **do**
7: tile[tid] = tile_item = prob[i] + tile_sum;
8: tile_item_scanned = parallel_scan(tile);
9: **barrier**();
10: tile_sum = tile[tile_size − 1];
{If the tile is selected, break the loop.}
11: **if** tile_sum ⩾ rand_value **then**
12: break;
13: **end if**
14: **end for**
{Parallel search the city id in the selected tile.}
15: **if** (tile_item_scanned − tile_item) < rand_value **and** tile_item_scanned ⩾ rand_value **then**
16: selected_id = i;
17: **end if**

The advantages of the TR approach are as follows:

(i) Fewer random numbers generated.

(ii) Improved memory locality.

(iii) Reduced scan computation.

(2) Tour-construction algorithm. We present the proposed tour-construction algorithm, which is tailored for a GPU. The algorithm is called SIMD-oriented tour construction that is based on the following definitions.

Definition 1. L^k is a tour list of n cities belonging to ant k and initializes with an ascending order of city numbers from 1 to n. $L^k_{i,j}$ is a subset of L^k with city indices ranging from i to j.

Definition 2. C^k is a cursor (integer value) of ant k that initially points to the first element of L^k.

Definition 3. $S_{i,j}$ is a node-swap operator on the elements of $L^k_{i,j}$, which swaps nodes L^k_i and L^k_j.

Because each ant travels to cities separately in the same manner, we consider one ant for the algorithm. The tour construction for ant k contains the following steps:

(i) Randomly select a city from $L^k_{C^k,n}$ called L^k_j.

(ii) Perform S_{C^k,L^k_j}, and then, increase C^k by 1.

(iii) Repeat steps 1 and 2 until C^k equals n. $L^k_{1,n}$ is the travel path of ant k.

Based on the above approach, we present a new GPU strategy called ST-DYNAMIC, which dynamically changes the work-group size for each movement of ants (Algorithm B5).

Experimental results. Our experiments were carried out on an Intel Core i7-4770 (@3.4 GHz) machine with an Nvidia Kepler GPU (GTX780). The experiments used a standard set of benchmark instances from the TSPLIB library. The detail of the experimental configuration was shown in Appendix C.1.

The basic GPU data-parallel algorithm is named ST-GROUP. As shown in Figure 1, ST-GROUP-STC starts by accelerating ST-GROUP from pcb442 and improves with increases in the problem size. Then, we added the TR method to ST-GROUP-STC (ST-GROUP-STC-TR), which results in nearly the same contribution to the acceleration with the same trend. Our ST-DYNAMIC is the fastest method. It is more suitable for large datasets, and can solve problem sizes as large as 4461. The speedups are 41.8x and 44.7x in pr2392 and fnl4461 respectively with ST-DYNAMIC.

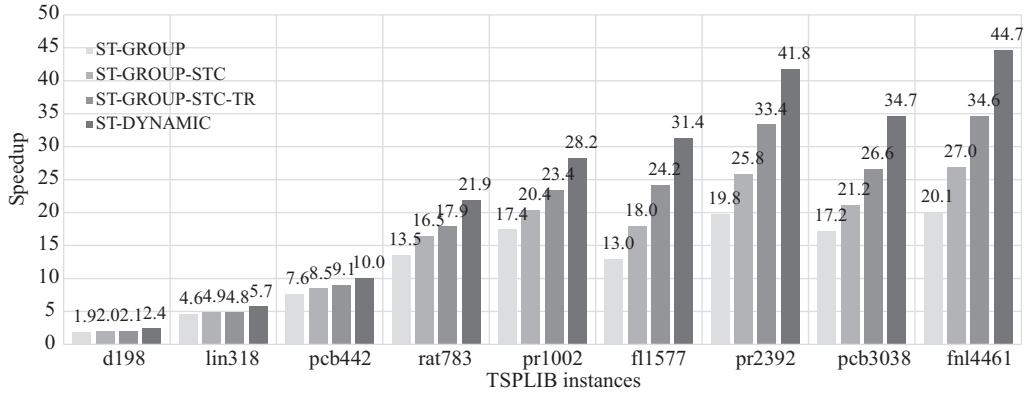

Figure 1 Speedup comparison of Kernel ST-GROUP and our three strategies.

To ensure the efficiency of our algorithm, we compared it against the first data-parallel GPU implementation of AS (DP-GPU-AS) provided by Cecilia et al. [5]. The results show our algorithm is up to 3.6x faster than DP-GPU-AS (Table C4). We also compared our algorithm with two improved GPU ACO algorithms [4, 6] on a GTX580 GPU with 512 CUDA cores. The results demonstrate that the performance of our proposed GPU ACO algorithm is the best one (Table C5).

The solution quality comparison results indicate that the solution quality of our GPU algorithm is similar to the solution quality of the sequential CPU algorithm. The percentage deviations of the GPU average results from the CPU average results are within 0.4%.

Conclusion. We proposed a new parallel ACO algorithm tailored for GPUs to achieve higher performance, and implemented this algorithm using a dynamic work-group strategy. We used a large range of TSP instances, varying from 198 to 4461 cities, to evaluate our algorithm. As a result, we obtained an outstanding speedup of up to 44x in the TSP problem compared to the CPU counterpart.

We also compared our algorithm with the existing GPU data-parallel ACOs. The results indicated that our algorithm was the best in terms of computational performance.

In future work, we will optimize our algorithm using quantitative performance analysis and algorithmic optimization approaches [7] to be suitable for middle-scale TSP problems and other GPU architectures. We will also try to extend the idea of GPU acceleration to other population-based metaheuristic methods. We may also apply the GPU acceleration method to the fields of CAD/graphics/images/video [8–10].

Acknowledgements This work was supported by National Science Foundation of China (Grant Nos. 61472289, 61502353) and Hubei Province Science Foundation (Grant No. 2015CFB254). The authors thank Dr. Cecilia for providing the CUDA source code in [5], which is a great benchmark for comparison.

Supporting information Appendixes A–C, including Algorithm B5, Tables C4 and C5. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv, 2003, 35: 268–308
2 Dorigo M, Stützle T. Ant Colony Optimization. Cambridge: MIT Press, 2004. 65–90
3 Alba E, Luque G, Nesmachnow S. Parallel metaheuristics: recent advances and new trends. Int Trans Oper Res, 2013, 20: 1–48
4 Uchida A, Ito Y, Nakano K. An efficient GPU implementation of ant colony optimization for the traveling salesman problem. In: Proceedings of the 2012 3rd International Conference on Networking and Computing (ICNC), Okinawa, 2012. 94–102
5 Cecilia J M, García J M, Nisbet A, et al. Enhancing data parallelism for ant colony optimization on GPUs. J Parallel Distr Com, 2013, 73: 42–51
6 Dawson L, Stewart I. Improving ant colony optimization performance on the GPU using CUDA. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC), Cancun, 2013. 1901–1908
7 Zhou Y, He F Z, Qiu Y M. Optimization of parallel iterated local search algorithms on graphics processing unit. J Supercomput, 2016, 72: 2394–2416
8 Wu Y Q, He F Z, Zhang D J, et al. Service-oriented feature-based data exchange for cloud-based design and manufacturing. IEEE Trans Serv Comput, 2016, doi: 10.1109/TSC.2015.2501981
9 Li K, He F Z, Chen X. Real time object tracking via compressive feature selection. Front Comput Sci-Chi, 2016, 10: 689–701
10 Cheng Y, He F Z, Wu Y Q, et al. Meta-operation conflict resolution for human-human interaction in collaborative feature-based CAD systems. Cluster Comput, 2016, 19: 237–253