Computing and Informatics, Vol. 37, 2018, 229–243, doi: 10.4149/cai_2018_1_229

PARALLELIZATION OF ANT SYSTEM FOR GPU UNDER THE PRAM MODEL

Andrej Brodnik

Department of Information Science and Technology
University of Primorska
Glagoljaška 8
6000 Koper, Slovenia
&
Faculty of Computer and Information Science
University of Ljubljana
Tržaška cesta 25
1000 Ljubljana, Slovenia
e-mail: andrej.brodnik@upr.si

Marko Grgurovič

Department of Information Science and Technology
University of Primorska
Glagoljaška 8
6000 Koper, Slovenia
e-mail: marko.grgurovic@student.upr.si

Abstract. We study the parallelized ant system algorithm solving the traveling salesman problem on n cities. First, following the series of recent results for the graphics processing unit, we show that they translate to the PRAM (parallel random access machine) model. In addition, we develop a novel pheromone matrix update method under the PRAM CREW (concurrent-read exclusive-write) model and translate it to the graphics processing unit without atomic instructions. As a consequence, we give new asymptotic bounds for the parallel ant system, resulting in step complexities $O(n \lg \lg n)$ on CRCW (concurrent-read concurrent-write) and $O(n \lg n)$ on CREW variants of PRAM using n^2 processors in both cases. Finally, we present an experimental comparison with the currently known pheromone matrix update methods on the graphics processing unit and obtain encouraging results.

Keywords: Parallel random access machine, graphics processing unit, ant system, metaheuristics, traveling salesman problem, combinatorial optimization

Mathematics Subject Classification 2010: 68-W10

1 INTRODUCTION

In this paper, we study the parallel variants of the Ant System (AS) algorithm, which is part of the ant colony optimization (ACO) family of metaheuristics. The ACO family of algorithms simulate the behavior of real ants which find paths using pheromone trails. A number of variations on the basic idea exist, such as the Ant System [7], Ant Colony System [6], the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [16] and many others. In this paper we focus on the canonical Ant System algorithm, which can be adapted to solve a variety of combinatorial optimization problems such as vehicle routing [2], quadratic assignment [13], subset problems [11] and others. In this paper, we will limit ourselves to the traveling salesman problem (TSP).

The adoption of the graphics processing unit (GPU) as a computing platform in recent years has triggered a wave of papers discussing the parallelization of known algorithms. Recent papers [4, 12, 17] have focused on providing a parallel version of Ant System for the GPU. In this paper, we show that these algorithms are more general and can be studied in absence of GPU specifics. In line with this observation, we suggest a move towards more well-understood theoretical models such as the parallel random access machine (PRAM). This greatly facilitates asymptotic analysis and subsequently allows one to see where the algorithms could be improved.

The main goal of this paper is to investigate efficient AS algorithms for variants of the PRAM model of computation and to identify how these might be useful in practice. We break down the AS algorithm into two separate phases: Tour Construction and Pheromone Update. Then we show that the existing GPU algorithms for AS can be translated to the PRAM model, which permits to perform asymptotic analysis. While Tour Construction remains efficient even on PRAM, we identify bottlenecks in the Pheromone Update phase, which are caused by reliance on atomic instructions that are not readily available on most variants of PRAM (or older GPUs). We overcome this with a novel Pheromone Update algorithm that does not require such instructions. Finally, we show that these results are relevant in practice when atomic instructions are not available. We do this by implementing the resulting Pheromone Update algorithm on the GPU and we obtain significantly better results in case of no atomic instructions.

The paper is structured as follows. In Section 2 we briefly introduce the PRAM model, the traveling salesman problem, and the Ant System algorithm in its sequential form. In Section 3 we provide PRAM implementations of the Ant System algorithm and show how to improve them, and finally we provide results of empirical tests on the GPU. In Section 4 we provide conclusions and suggestions for future work.

2 BACKGROUND

2.1 Parallel Random Access Machine

The PRAM model is a variant of the random access machine (RAM) adapted to parallel execution. We denote the number of processors by p. In this paper, we deal with two types of synchronous PRAM: concurrent-read exclusive-write (CREW) and concurrent-read concurrent-write (CRCW). The CREW variant assumes that each memory location is tied to a specific processor, and only that processor can write to it. However, any processor can read from any memory location. In contrast, the CRCW variant has no such restriction. Since under CRCW all processors can write to the same location, it is typical to parametrize the CRCW variant by how the competing writes are handled. In this paper we consider two standard ways of doing that:

- COMMON: All processors must write the same value.
- COMBINING: All values being concurrently written are combined using some operator (e.g. addition, maximum, etc.).

In this paper we will focus on CREW, CRCW and COMBINING CRCW algorithms, where by CRCW we mean algorithms that run under the COMMON variant. An important parallel operation which we will make extensive use of is finding the largest element in an array of n elements. Throughout the paper we will use $S(n)$ to denote the step complexity of a parallel algorithm, i.e. the number of steps executed. The work complexity of an algorithm, denoted by $W(n)$, corresponds to the total number of operations executed (over all processors). It is important to note that finding the maximum among n numbers can be performed in $S(n) = O(\lg \lg n)$ time under CRCW [15] with $p = n$. However, it is only possible in $S(n) = O(\lg n)$ time under CREW with the same number of processors. The work complexity is $W(n) = O(n)$ in both cases. Under COMBINING CRCW, finding the maximum can be performed in $S(n) = O(1)$ and $W(n) = O(n)$ by making use of the combining mechanism in a trivial way (i.e. setting it to be the maximum operation).

2.2 The Traveling Salesman Problem

In the traveling salesman problem (TSP), we are given a complete, directed graph $G = (V, E)$, with V and E being the sets of vertices and edges, respectively. We are also given a function $\ell : E \to \mathbb{R}^+$ which maps each edge to its length. To simplify notation, we define $n = |V|$. The task, then, is to produce a permutation Π of V

with the least cost. Let Π_k denote the vertex at position k in the permutation Π. The cost of a permutation Π of V is then defined as:

$$\ell(\Pi_n, \Pi_1) + \sum_{2 \le k \le n} \ell(\Pi_{k-1}, \Pi_k).$$

Observe, that even though our formulation requires a complete graph, sparse graphs can be handled by inserting the missing edges with length ∞. Undirected graphs can also be handled simply by creating two directed edges for each undirected edge, with equal lengths assigned to them.

2.3 Ant System for the TSP

As in the description of the TSP problem, we assume we are given a directed, complete graph $G = (V, E)$. We then define the heuristic matrix η and the pheromone matrix τ, both of dimensions $n \times n$. The heuristic matrix is constant throughout the algorithm and represents the quality of an edge (u, v). Formally, we choose $\eta_{u,v} = 1/\ell(u, v)$. The pheromone matrix changes throughout the execution of the algorithm. Two parameters α and β regulate the importance of pheromone and heuristic information, respectively.

Algorithm 1 Sequential Ant System

```
1: procedure AntSystem($\alpha, \beta, \rho, totalIterations$)
2:     Allocate matrices of size $n \times n$: $\eta$, $\tau$, $chance$, $\pi$, $tabu$
3:     Allocate vector of size $n$: $score$
4:     for $iter := 1$ to $totalIterations$ do
5:         Initialize($\alpha, \beta, \tau, \eta, score, chance, \pi, tabu$)
6:         TourConstr($\eta, score, chance, \pi, tabu$)
7:         PheromoneUpdate($\tau, \rho, score$)
8:     end for
9: end procedure
```

Ants then build solutions according to:

$$p(v|p, S) = \frac{\tau_{p,v}^{\alpha} \cdot \eta_{p,v}^{\beta}}{\sum_{w \in N(p,S)} \tau_{p,w}^{\alpha} \cdot \eta_{p,w}^{\beta}} \tag{1}$$

where $p(v|p, S)$ is the probability of choosing vertex v when at position p and according to the current partial solution S. The feasible neighborhood of the current incomplete solution is defined by $N(p, S)$. Since the TSP does not permit returns to previously included vertices (except for the last vertex), those vertices have probability zero of being included. This is typically accomplished by having each ant keeping the track of a tabu list.

Algorithm 2 Sequential Initialize

```
procedure INITIALIZE($\alpha, \beta, \tau, \eta, score, chance, \pi, tabu$)
    Allocate vector of size $n$: $sum$
    for $i := 1$ to $n$ do
        $sum[i] := 0$
    end for
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $sum[i] := sum[i] + \tau[i,j]^\alpha \cdot \eta[i,j]^\beta$
        end for
    end for
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $chance[i,j] := \tau[i,j]^\alpha \cdot \eta[i,j]^\beta / sum[i]$
            $tabu[i,j] := 1$
        end for
    end for
    for $i := 1$ to $n$ do
        $\pi[i,1] := i$
        $score[i] := 0$
        $tabu[i,i] := 0$
    end for
end procedure
```

Once solutions are constructed, they are evaluated to obtain their respective qualities, which in most cases is simply the inverse of the cycle length. Once evaluated, the qualities are used to update the pheromone matrix. First, each cell of the pheromone matrix is decreased by a constant factor (evaporation) and then increased according to the solution score (pheromone deposit). Let $f(S)$ denote the score of solution S and let Z be the set of all solutions produced by the ants, where each ant contributes a single solution. Then, the pheromone update stage is defined by:

$$\tau_{v,w} \leftarrow (1-\rho) \cdot \tau_{v,w} + \sum_{S \in Z | (v,w) \in S} f(S). \tag{2}$$

When considering AS for TSP, the recommended number of ants equals the number of vertices [8]. Thus hereof we always assume we have n ants, each starting its solution in a different vertex. The Ant System algorithm, as we have described it, can be formalized as Algorithm 1. An initialization stage (Algorithm 2) was added where certain bookkeeping tasks can be performed. The tour construction (Algorithm 3) and pheromone update (Algorithm 4) stages correspond to what we have described. In line 7 of Algorithm 3 we call the function *rand*(), which is supposed to return a random uniformly distributed real number in the range $(0,1)$. This is the source of randomness in the algorithm, and allows it to implement the

Algorithm 3 Sequential Tour Construction

```
procedure TourConstr($\eta$, $score$, $chance$, $\pi$, $tabu$)
    for $i := 1$ to $n$ do
        for $k := 2$ to $n$ do
            $v := 0$
            $c := -\infty$
            for $j := 1$ to $n$ do
                $t := chance[\pi[i, k-1], j] \cdot rand() \cdot tabu[i, j]$
                if $t \geq c$ then
                    $c := t$
                    $v := j$
                end if
            end for
            $\pi[i, k] := v$
            $tabu[i, \pi[i, k]] := 0$
            $score[i] := score[i] + \eta[\pi[i, k-1], \pi[i, k]]$
        end for
    end for
    for $i := 1$ to $n$ do
        $score[i] := score[i] + \eta[\pi[i, n], \pi[i, 1]]$
    end for
end procedure
```

probabilistic selection according to Equation (1). The algorithm also uses a number of matrices, which play the following roles: *chance* stores the visit probability values (cf. (1)), π holds the solutions, *tabu* is used to prevent infeasible solutions. The vector *score* holds the computed score for each solution.

3 PARALLEL ANT SYSTEM

It is conceptually simpler to consider Ant System as a combination of two algorithms: tour construction and pheromone update (lines 6 and 7 in Algorithm 1, respectively). Attempts at parallel AS, e.g. [5, 3], are usually not very attractive for the PRAM model, since they either employ coarse parallelization or neglect certain parts of parallel AS, typically pheromone update. However, it turns out that parallel AS algorithms for the GPU model [12, 17, 4] translate almost without any effort to the PRAM model. Thus, we focus exclusively on the translation and improvement of the GPU algorithms. It is important to note that the unit of parallelism in the GPU is a thread while on a PRAM the unit of parallelism is a processor. However since the PRAM is a theoretical model, the actual meaning of processor in this context is abstract.

Algorithm 4 Sequential Pheromone Update

```
1:  procedure PheromoneUpdate($\tau, \rho, score$)
2:      for $i := 1$ to $n$ do
3:          for $j := 1$ to $n$ do
4:              $\tau[i,j] := (1-\rho)\cdot\tau[i,j]$
5:          end for
6:      end for
7:      for $i := 1$ to $n$ do
8:          for $k := 2$ to $n$ do
9:              $\tau[\pi[i,k-1],\pi[i,k]] := \tau[\pi[i,k-1],\pi[i,k]] + score[i]$
10:         end for
11:         $\tau[\pi[i,n],\pi[i,1]] := \tau[\pi[i,n],\pi[i,1]] + score[i]$
12:     end for
13: end procedure
```

Due to the decomposition of AS into two algorithms (construction and update), the complexity of AS becomes the worst of the two. We will now explore strategies for each algorithm.

3.1 Tour Construction

The simplest method (cf. [12, 17]) delegates each ant to a unique processor. Now, since each ant stochastically considers each vertex n times (cf. (1)) and has $p = n$ processors, this amounts to step complexity $S(n) = O(n^2)$ and work complexity $W(n) = O(n^3)$.

A remarkable contribution of [4] is their strategy for parallel tour construction. Their tour construction method uses $p = n^2$ processors and associates each ant with n processors. When each ant can make use of n processors, it can effectively generate multiple random numbers in parallel. Then, the maximum operation is used to choose one among n neighbouring vertices, again in parallel. In total, n maximum operations are performed per ant. When translating this result to the PRAM model, the step complexity of the algorithm depends on the model of computation. In the case of CREW, the maximum can be found with a step complexity $S(n) = O(\lg n)$ and work complexity $W(n) = O(n)$. Since there are n maximum operations per ant this brings the step complexity to $S(n) = O(n \lg n)$. There are n ants in total, each performing n maximum operations, meaning the work complexity remains $W(n) = O(n^3)$. However, under CRCW, maximum can be performed in $S(n) = O(\lg \lg n)$ step complexity (see e.g. [15]), thus the step complexity of the algorithm becomes $S(n) = O(n \lg \lg n)$, with the work complexity remaining the same as in the CREW case. Under COMBINING CRCW, this is further reduced to $S(n) = O(n)$ by simply taking the combining operation to be maximum.

It is possible to further reduce the step complexity of the CRCW algorithm to $S(n) = O(n)$ using $p = n^3$ processors and a different method to find the maximum which takes $S(n) = O(1)$: simply compare all pairs of elements in the array in parallel. However, we will restrict ourselves to $p = n^2$, since the large amount of additional processors required hardly justifies the $\lg \lg n$ gain.

3.2 Pheromone Update

Once tour is constructed, the pheromone update must be performed. In [12, 17] the latter is accomplished sequentially rather than in parallel, i.e., one processor performs the update in $S(n) = O(n^2)$ and $W(n) = O(n^2)$ while others are waiting. This method is appropriate if we use the first construction method, which also has a step complexity of $O(n^2)$, but it becomes a bottleneck if we choose the more parallel construction method of [4].

Two pheromone update methods can be found in [4]. The first is straightforward and is based on atomic instructions for addition (cf. the summation in Equation (2)). This method corresponds to the use of COMBINING CRCW with the combining operation set to addition. Thus, we already have one parallel method for pheromone update with $p = n$ and running with a step complexity of $S(n) = O(n)$ and a work complexity of $W(n) = O(n^2)$. If we allow $p = n^2$, then the update can be performed in $S(n) = O(1)$.

The second method of [4] which they refer to as "scatter to gather" is more computationally intensive, but does not use atomic instructions. In this case each cell of the pheromone matrix is represented by a distinct processor, so we require $p = n^2$. Each processor loops through all solutions, summing only the relevant qualities. Solutions are of size $O(n)$ and there are n solutions, meaning each processor performs $S(n) = O(n^2)$ operations. Since there are n^2 processors, this yields a $W(n) = O(n^4)$ work complexity. This method works under both CREW and CRCW models, but in terms of computational complexity, it is uninteresting. Better bounds are accomplished by performing pheromone update sequentially, i.e. by a single processor while others wait. Nonetheless, we mention this method because we will show how to improve its complexity.

3.3 Improvements

In this subsection, we propose a novel method for pheromone update, which improves the currently known bounds under the CREW and CRCW models. Tour construction in our algorithm is performed as in [4], which translates effortlessly to the PRAM. However, instead of using their "scatter to gather" pheromone update, we develop a new technique.

Theorem 1. Pheromone update using $p = n^2$ processors can be performed in $S(n) = O(n)$ and $W(n) = O(n^3)$ under a CREW PRAM.

Proof. Each ant already stores a list of n entries, which correspond to vertices in the order it visited them. In addition to this list, we require that each ant also stores an array *edge* of length n, implicitly storing which edge was used to reach a particular vertex. For example, if the edge (u, v) was used to visit vertex v, then we would set $edge[v] := u$. During pheromone update, we can now check whether a given solution S contains the desired edge in constant time. Without this array, we would have to inspect every element of the solution, which would take $O(n)$ time. There are n solutions, so the step complexity of pheromone update becomes $S(n) = O(n)$ and the work complexity becomes $W(n) = O(n^3)$. $\square$

The pseudocode for the parallel algorithm is shown in Algorithms 5, 6, 7 and 8. PRAM algorithms use a scalar processor identifier. To improve readability we use a two-dimensional processor identifier $(x, y) \in [n] \times [n]$, where $[n] = \{1, 2, \ldots, n\}$. Remember that each ant is using n processors, so the x component of the identifier denotes an ant and the y component denotes an ant's processor. The algorithm consists of an initialization phase, where we compute the probability (*chance* matrix) of choosing certain edges and reset structures after each iteration. We explicitly denote variables that are local to each processor by prefixing them with a *local* identifier in their initialization. All matrices in the algorithm are of size $n \times n$. The matrices η, τ, *chance*, π, *tabu* and vector *score* were already described in Section 2. Additional matrices exist for the parallel algorithm which have the following roles: R holds the results from parallel random number generation and *edge* is used as described in the proof of Theorem 1.

Theorem 2. Algorithm 5 executes on a CREW PRAM.

Proof. It is easy to see that writes to R (line 3 in Algorithm 7) and τ (lines 2 and 5 in Algorithm 8) preserve write exclusivity since only processor (x, y) writes to $R[x, y]$ and $\tau[x, y]$. We lump together the proof of write exclusivity for *chance* (line 8 in Algorithm 6), *tabu* (lines 9 and 13 in Algorithm 6 and line 7 in Algorithm 7), *score* (line 12 in Algorithm 6 and lines 8 and 13 in Algorithm 7) and *edge* (lines 6 and 12 in Algorithm 7). Observe that in each case the processor's y index is set to one. For *score*, which only has one dimension, this avoids conflicts. The rest are matrices and all writes from processor $(x, 1)$ are to cells (x, k) where $k \in [n]$, which does not lead to any conflicts. Note that the proof for the write exclusivity of π (line 11 in Algorithm 6 and line 4 in Algorithm 7) is the same. Naturally, we require that the parallel implementation of $\arg\max$ observes the write exclusivity of π (which leads to different implementations on CREW and CRCW). $\square$

Corollary 1. Algorithm 5 executes under a CRCW PRAM.

It is easy to see that the suggested pheromone update method can be sped up if more processors are provided. For example, given $p = n^3$ processors, each cell in the pheromone matrix can be represented by n processors, allowing pheromone summation to be performed using reduction. However, there seems little incentive to do so, since the complexity of parallel Ant System algorithm becomes dominated by the tour construction step.

Algorithm 5 Parallel Ant System

```
1: procedure PAntSystem($\alpha, \beta, \rho, totalIterations$)
2:     Allocate matrices of size $n \times n$: $R, \eta, \tau, chance, \pi, tabu, edge$
3:     Allocate vector of size $n$: $score$
4:     for $i := 1$ to $totalIterations$ do
5:         for $(x, y) \in [n] \times [n]$ in parallel do
6:             PInitialize($x, y, \alpha, \beta, \tau, \eta, score, chance, \pi, tabu$)
7:             PTourConstr($x, y, R, \eta, score, chance, \pi, tabu, edge$)
8:             PPheromoneUpdate($x, y, \tau, \rho, edge, score$)
9:         end for
10:    end for
11: end procedure
```

Algorithm 6 Parallel Initialize

```
1: procedure PInitialize($x, y, \alpha, \beta, \tau, \eta, score, chance, \pi, tabu$)
2:     if $y = 1$ then
3:         local float $sum := 0$
4:         for $i := 1$ to $n$ do
5:             $sum := sum + \tau[x, i]^\alpha \cdot \eta[x, i]^\beta$
6:         end for
7:         for $i := 1$ to $n$ do
8:             $chance[x, i] := \tau[x, i]^\alpha \cdot \eta[x, i]^\beta / sum$
9:             $tabu[x, i] := 1$
10:        end for
11:        $\pi[x, 1] := x$
12:        $score[x] := 0$
13:        $tabu[x, x] := 0$
14:    end if
15: end procedure
```

Table 1 summarizes complexity bounds derived from the previous work as well as new bounds resulting from the improvements presented in this paper. Since a single iteration of the parallel Ant System algorithm requires both tour construction and pheromone update, the bound becomes the worse of the two.

3.4 Empirical Comparison

We implemented different pheromone update methods on the GPU. We used Nvidia CUDA, which was also used in recent papers [4, 12, 17] studying the parallel GPU implementation of the Ant System algorithm. Compared to the GPU, the PRAM model is much simpler. While programs on the PRAM execute in SIMD (single instruction, multiple data) lock-step fashion, the GPU model of execution is the

Algorithm 7 Parallel Tour Construction

```
procedure PTourConstr($x, y, R, \eta, score, chance, \pi, tabu, edge$)
    for $k := 2$ to $n$ do
        $R[x,y] := chance[\pi[x,k-1],y] \cdot rand() \cdot tabu[x,y]$
        Compute ($\arg\max_{i\in\{1\ldots n\}} R[x,i]$) and store result in $\pi[x,k]$
        if $y = 1$ then
            $edge[x,\pi[x,k]] := \pi[x,k-1]$
            $tabu[x,\pi[x,k]] := 0$
            $score[x] := score[x] + \eta[\pi[x,k-1],\pi[x,k]]$
        end if
    end for
    if $y = 1$ then
        $edge[x,\pi[x,1]] := \pi[x,n]$
        $score[x] := score[x] + \eta[\pi[x,n],\pi[x,1]]$
    end if
end procedure
```

Algorithm 8 Parallel Pheromone Update

```
procedure PPheromoneUpdate($x, y, \tau, \rho, edge, score$)
    $\tau[x,y] := (1-\rho) \cdot \tau[x,y]$
    for $k := 1$ to $n$ do
        if $edge[k,y] = x$ then
            $\tau[x,y] := \tau[x,y] + score[k]$
        end if
    end for
end procedure
```

significantly more ambiguous SIMT (single instruction, multiple threads), where such lock-step guarantees are lost. Together with details like different levels of memory with different speeds and capacities, writing programs becomes a matter of mixing theoretical and practical considerations. With this paper we mainly focus on the theoretical aspects of such programs by studying them in the cleaner PRAM model, then transferring them over to the "messier" GPU.

The tests were run on an Nvidia GeForce GTX 560Ti using stock Nvidia frequencies. Test instances were taken from TSPLIB [14], which are standard test cases. We included some of the instances that have also been used by [4] to facilitate comparisons. We compared only the pheromone update stage, since our tour construction step is identical to the one presented in [4], thus we refer readers interested in comparisons between various tour construction methods or comparisons between the parallel and sequential code to that paper.

We tested three methods: atomic, scatter-gather and non-atomic fast. The atomic method updates the pheromone matrix using atomic instructions for addition. The scatter-gather method is the non-atomic method proposed by [4]. Finally,

Previous Work				
		CREW	**CRCW**	**CMB. CRCW**
Tour [4]	$\mathbf{S(n)}$	$O(n \lg n)$	$O(n \lg \lg n)$	$O(n)$
	$\mathbf{W(n)}$	$O(n^3)$	$O(n^3)$	$O(n^3)$
PH [4]	$\mathbf{S(n)}$	$O(n^2)$	$O(n^2)$	$O(1)$
	$\mathbf{W(n)}$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Total	$\mathbf{S(n)}$	$O(n^2)$	$O(n^2)$	$O(n)$
	$\mathbf{W(n)}$	$O(n^3)$	$O(n^3)$	$O(n^3)$
This Paper				
		CREW	**CRCW**	
PH	$\mathbf{S(n)}$	$O(n)$	$O(n)$	
	$\mathbf{W(n)}$	$O(n^3)$	$O(n^3)$	
Total	$\mathbf{S(n)}$	$O(n \lg n)$	$O(n \lg \lg n)$	
	$\mathbf{W(n)}$	$O(n^3)$	$O(n^3)$	

Table 1. Previous and new bounds for the parallel Ant System, which is comprised of two sub-algorithms: tour construction and pheromone (PH) update. We denote the COMBINING CRCW model by CMB. CRCW. Step and work complexities are denoted by $S(n)$ and $W(n)$, respectively. All bounds assume n^2 processors.

	Method		
Instance	**Atomic** [4]	**Scatter-Gather** [4]	**Non-Atomic Fast**
att48	0.06	1.29	0.19
kroC100	0.11	17.35	0.51
a280	0.47	759.14	3.61
pcb442	0.82	4 681	11.5
d657	1.74	$22 \cdot 10^3$	34.7
pr1002	3.48	$118 \cdot 10^3$	114.8
pr2392	16.39	$3\,800 \cdot 10^3$	1 525.4

Table 2. Running time (milliseconds) of pheromone update methods on TSPLIB instances

the non-atomic fast method is the one suggested in this paper. We also remark that, in our case, the atomic update method made full use of $p = n^2$ threads, since we found its performance to be significantly better compared to $p = n$ threads, as used in [4]. The results are shown in Table 2 and are also shown as a plot in Figure 1.

It is reassuring to see that the theoretical improvements also translate into practice. While the atomic variant is significantly faster, many older GPUs still in use today do not have access to the appropriate atomic instructions. Thus, these results are practically relevant for GPU implementations if code is expected to work on all GPUs currently in use.

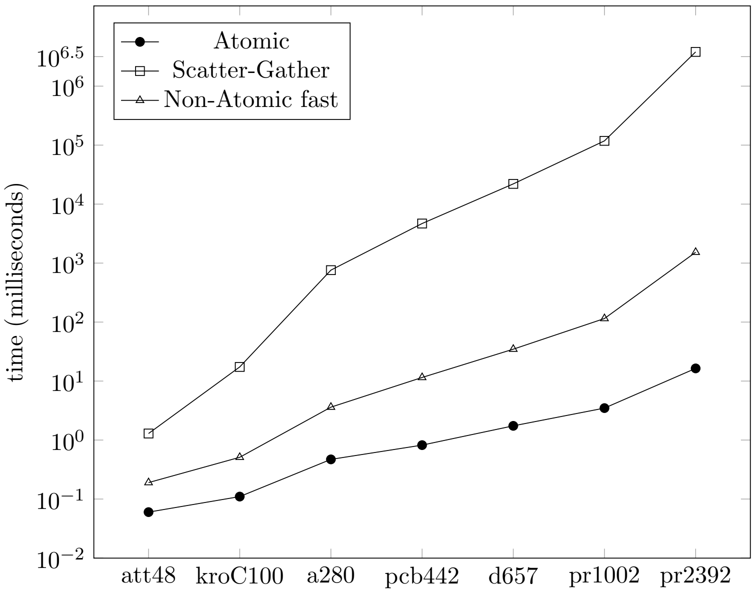

Figure 1. Plotted running times of pheromone update methods on TSPLIB instances

4 CONCLUSION

In this paper we have shown that recent parallel variants of the Ant System algorithm for the GPU systems can be easily modeled by the more general PRAM model. This makes them both simpler to understand and to analyze. The facilitation in a theoretical analysis allowed us to determine which parts of the algorithm needed improvement. It turned out that in two out of three variants of PRAM models studied, the parallel Ant System algorithm was dominated by the pheromone update. We proposed a new pheromone update method that improves the asymptotic bound of the parallel Ant System algorithm to such an extent, that the entire algorithm becomes dominated by the tour construction phase.

Future research directs us to study the possibility of application of the proposed pheromone update method to other algorithms in the ACO family. Moreover, optimization problems other than the TSP could be parallelized in a similar fashion. The algorithms could be studied under various other parallel computation models. Last but not least, we are also interested in other algorithms that could be more efficiently parallelized if they are split into two phases or more phases.

REFERENCES

[1] Bilchev, G.—Parmee, I. C.: The Ant Colony Metaphor for Searching Continuous Design Spaces. In: Fogarty, T. C. (Ed.): Evolutionary Computing (AISB EC 1995). Springer, Lecture Notes in Computer Science, Vol. 993, 1995, pp. 25–39.

[2] Bullnheimer, B.—Hartl, R. F.—Strauss, C.: An Improved Ant System Algorithm for the Vehicle Routing Problem. Annals of Operations Research, Vol. 89, 1999, pp. 319–328, doi: 10.1023/A:1018940026670.

[3] Bullnheimer, B.—Kotsis, G.—Strauss, C.: Parallelization Strategies for the Ant System. In: De Leone, R. et al. (Eds.): High Performance Algorithms and Software in Nonlinear Optimization. Springer, Boston, Applied Optimization, Vol. 24, 1998, pp. 87–100.

[4] Cecilia, J. M.—García, J. M.—Nisbet, A.—Amos, M.—Ujaldón, M.: Enhancing Data Parallelism for Ant Colony Optimization on GPUs. Journal of Parallel and Distributed Computing, Vol. 73, 2013, No. 1, pp. 42–51, doi: 10.1016/j.jpdc.2012.01.002.

[5] Delisle, P.— Krajecki, M.—Gravel, M.—Gagné, C.: Parallel Implementation of an Ant Colony Optimization Metaheuristic with OpenMP. Proceedings of the Third European Workshop on OpenMP, International Conference on Parallel Architectures and Compilation Techniques, 2001, pp. 8–12.

[6] Dorigo, M.—Gambardella, L. M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1, 1997, No. 1, pp. 53–66, doi: 10.1109/4235.585892.

[7] Dorigo, M.—Maniezzo, V.—Colorni, A.: Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), Vol. 26, 1996, No. 1, pp. 29–41, doi: 10.1109/3477.484436.

[8] Dorigo, M.—Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.

[9] Korošec, P.—Šilc, J.: Using Stigmergy to Solve Numerical Optimization Problems. Computing and Informatics, Vol. 27, 2008, No. 3, pp. 377–402.

[10] Korošec, P.—Šilc, J.—Filipič, B.: The Differential Ant-Stigmergy Algorithm. Information Sciences, Vol. 192, 2012, pp. 82–97, doi: 10.1016/j.ins.2010.05.002.

[11] Leguizamon, G.—Michalewicz, Z.: A New Version of Ant System for Subset Problems. In: Angeline, P. J. et al. (Eds.): Proceedings of the Congress on Evolutionary Computation (CEC '99), Washington, D.C., July 1999, pp. 1459–1464, doi: 10.1109/CEC.1999.782655.

[12] Li, J.—Hu, X.—Pang, Z.—Qian, K.: A Parallel Ant Colony Optimization Algorithm Based on Fine-Grained Model with GPU-Acceleration. International Journal of Innovative Computing, Information, and Control, Vol. 5, 2009, No. 11 (A), pp. 3707–3716.

[13] Maniezzo, V.—Colorni, A.: The Ant System Applied to the Quadratic Assignment Problem. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, 1999, No. 5, pp. 769–778, doi: 10.1109/69.806935.

[14] Reinelt, G.: TSPLIB – A Traveling Salesman Problem Library. INFORMS Journal on Computing, Vol. 3, 1991, No. 4, pp. 376–384, doi: 10.1287/ijoc.3.4.376.

[15] Shiloach, Y.—Vishkin, U.: Finding the Maximum, Merging, and Sorting in a Parallel Computation Model. Journal of Algorithms, Vol. 2, 1981, No. 1, pp. 88–102, doi: 10.1007/BFb0105127.

[16] Stützle, T.—Holger, H. H.: MAX-MIN Ant System. Future Generation Computer Systems, Vol. 16, 2000, No. 8, pp. 889–914.

[17] You, Y.-S.: Parallel Ant System for Traveling Salesman Problem on GPUs. In: Raidl, G. et al. (Eds.): Genetic and Evolutionary Computation Conference (GECCO 2009), New York, July 2009, pp. 1–2.

Andrej Brodnik received his Ph.D. from the University of Waterloo, Ontario, Canada. In 2002 he moved to University of Primorska. During the same time he also worked as Researcher and Adjoined Professor with the University of Technology in Luleå, Sweden. He authored several tens of various scientific papers. He is also author and co-author of patents in Sweden and USA. The CiteSeer and ACM Digital Library lists over 200 citations of his works. Currently he holds positions with the University of Ljubljana and the University of Primorska.

Marko Grgurovič is a Ph.D. student in computer science at the University of Primorska. He received his B.Sc. (2010) and M.Sc. (2012) degrees in computer science from the University of Primorska. His research interests lie in theoretical computer science, particularly in the design and analysis of algorithms.