Article

Multi-swarm PSO algorithm for the Quadratic Assignment Problem: a massively parallel implementation on the OpenCL platform *

Piotr Szwed [1,†,‡]*, and Wojciech Chmiel [2,‡]

[1] AGH University of Science and Technology; pszwed@agh.edu.pl
[2] AGH University of Science and Technology; wch@agh.edu.pl
* Correspondence: wch@agh.edu.pl; Tel.: +48 126 172 812
‡ These authors contributed equally to this work.

Academic Editor: name
Version January 24, 2018 submitted to

Abstract: This paper presents a multi-swarm PSO algorithm for the Quadratic Assignment Problem (QAP) implemented on the OpenCL platform. Our work was motivated by results of time efficiency tests performed for single-swarm algorithm implementation that showed clearly that the benefits of a parallel execution platform can be fully exploited provided the processed population is large. The described algorithm can be executed in two modes: with independent swarms or with migration. We discuss the algorithm construction as well as we report results of tests performed on several problem instances from the QAPLIB library. During the experiments the algorithm was configured to process large populations. This allowed us to collect statistical data related to values of goal function reached by individual particles. We use them to demonstrate on two test cases that although single particles seem to behave chaotically during the optimization process, when the whole population is analyzed, the probability that a particle will select a near-optimal solution grows.

Keywords: QAP, PSO, OpenCL, GPU calculation, particle swarm optimization, multi-swarm, discrete optimization

1. Introduction

The Quadratic Assignment Problem (QAP) [**? ? ?**] is a well known combinatorial problem that can be used as optimization model in many areas and is one of the most fundamental and difficult combinatorial problem which are the subject operation research[**? ? ?**].

The QAP problem generalizes a large number of theoretical issues and models several practical problems such as the graph partitioning, maximal clique, linear arrangement problem, balancing of jet turbines, less-than-truckload (*LTL*), very-large-scale integration (*VLSI*), backboard wiring problem and molecular fitting.

In the QAP problem the goal is to find an assignment of n-*facilities* to n-*locations* that minimizes the total sum of distances between facilities' locations multiplied by flows between these facilities. As the problem is NP hard [**?**], it can be solved optimally only for small problem instances whereas for larger problems (n>30) the approximation methods have to be used, for example heuristic algorithms [**? ? ?**]. One of the discussed methods [**? ?**] is the Particle Swarm Optimization (PSO). In this method a population of particles moves in the solution space to find an optimal problem solution. Usually two features are associated with each particle: its position (encoding the solution) and velocity. The particles explore the solution space by changing their position on the basis of the information about values of

*Work has been financed by the National Centre for Research and Development, grant number DZP/RID-I-68/14/NCBIR/2016.

 doi:10.20944/preprints201801.0230.v1

goal function for previously reached positions and the best solution in the swarm [**?**]. In our recent work [**?**] we have developed the PSO algorithm for the Quadratic Assignment Problem on OpenCL platform. The algorithm was capable of processing one swarm, in which particles shared information about the globally best solution to update their search directions. Following typical patterns for GPU based calculations, the implementation was a combination of parallel tasks (kernels) executed on GPU orchestrated by sequential operations run on the host (CPU). Such organization of computations involves inevitable overhead related to data transfer between the host and the GPU device. The time efficiency test reported in [**?**] showed clearly that the benefits of a parallel execution platform can be fully exploited if processed populations are large, e.g. if they comprise several hundreds or thousands particles. For smaller populations sequential algorithm implementation was superior both as regards the total swarm processing time and the time required to process one particle. This suggested a natural improvement of the previously developed algorithm: by scaling it up to high numbers of particles organized into several swarms.

In this paper we discuss a multi-swarm implementation the PSO algorithm for the QAP problem on OpenCL platform. The algorithm can be executed in two modes: with independent swarms, each maintaining its best solution, or with migration between swarms. We describe the algorithm construction as well as we report tests performed on several problem instances from the QAPLIB library [**?**]. Their results show advantages of massive parallel computing: the obtained solutions are very close to optimal or best known for particular problem instances.

The developed algorithm is not designed to exploit the problem specificity (see for example [**?**]) as well as it is not intended to compete with supercomputer or grid based implementations providing exact solutions for the QAP problem[**?**]. On the contrary, we are targeting low-end GPU devices, which are present in most laptops and workstations in everyday use, and accept near-optimal solutions.

During the tests the algorithm was configured to process large numbers of particles (in the order of 10000). This allowed us to collect data related to goal function values reached by individual particles and present such statistical measures as percentile ranks and probability mass functions for the whole populations or selected swarms.

The paper is organized as follows: next Section **??** discusses the QAP problem as well as the PSO method. It is followed by Section **??** which describes the adaptation of the PSO algorithm to the QAP problem and the parallel implementation on the OpenCL platform. Experiments performed and their results are presented in Section **??**. Section **??** provides concluding remarks.

2. Related works

2.1. Quadratic Assignment Problem

In 1957 Koopmans and Beckman defined Quadratic Assignment Problem as a mathematical model describing assignment of economic activities to a set of locations [**?**].

Let $V = \{1, ..., n\}$ be a set of *locations* (nodes) linked by n^2 arcs. Each arc linking a pair of nodes (k, l) is attributed with a non-negative weight d_{kl} interpreted as a distance. Distances are usually presented in form of $n \times n$ distance matrix $D = [d_{kl}]$. The next problem component is a set of facilities $N = \{1, ..., n\}$ and a $n \times n$ non-negative flow matrix $F = [f_{ij}]$, whose elements describe flows between pairs of facilities (i, j). The problem goal is to find an assignment $\pi \colon N \to V$ that minimizes the total cost calculated as sum of flows f_{ij} between pairs of facilities (i, j) multiplied by distances $d_{\pi(i)\pi(j)}$ between pairs of locations $(\pi(i), \pi(j))$, to which they are assigned. The permutation π can be encoded as n^2 binary variables x_{ki}, where $k = \pi(i)$, what gives the following problem statement:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} f_{ij} d_{kl} x_{ki} x_{lj} \tag{1}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

subject to:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} x_{ij} = 1, \qquad \text{for } 1 \leq j \leq n \\ &\textstyle\sum_{j=1}^{n} x_{ij} = 1, \qquad \text{for } 1 \leq i \leq n \\ &x_{ij} \in \{0,1\} \end{aligned} \tag{2}$$

The $n \times n$ matrix $X = [x_{ki}]$ satisfying (??) is called permutation matrix. In most cases matrix D and F are symmetric. Moreover, their diagonal elements are often equal 0. Otherwise, the component $f_{ii}d_{kk}x_{ki}x_{ki}$ can be extracted as a linear part of the goal function interpreted as an installation cost of i-th facility at k-th location .

The Qudratic Assignmet Problem mathematically models the problem from various areas such as distributed computing, transportation [?], architecture (flow of patients between location in a hospital), task scheduling, electronics (VLSI design), creating the control panels and manufacturing [?], statistical data analysis, balancing of running jet turbine [?], the analysis of reaction chemistry and genetics [?].

The QAP probem is strongly $\mathcal{NP}$*-hard* [?]. Sahni and Gonzalez proved that existence of a polynomial time algorithm for solving the QAP problem implies an existence of a polynomial time algorithm for an $\mathcal{NP}$*-complete* decision problem such as existing Hamiltonian cycle.

In many research works the QAP problem is considered one of the most challenging optimization problem. This in particular regards problem instances gathered in a publicly available and continuously updated the QAPLIB library [? ?]. A practical size limit for problems that can be solved with exact algorithms is about $n = 30$ [?]. In many cases optimal solutions were found with branch and bound algorithm requiring high computational power offered by computational grids [?] or supercomputing clusters equipped with a few dozen of processor cores and hundreds gigabytes of memory [?]. On the other hand, in [?] a very successful approach exploiting the problem structure was reported. It allowed to solve several hard problems from the QAPLIB using very little resources.

A number of heuristic algorithms allowing to find a near-optimal solutions for the QAP problem were proposed. They include Genetic Algorithm [?], various versions of Tabu search [?], Ant Colonies [? ?] and Bees algorithm [?]. Another method, being discussed further, is Particle Swarm Optimization [? ?] .

2.2. Particle Swarm Optimization

The PSO algorithm was developed to solve optimization problems in continuous domain [?]. A set of particles moves through a solution space and updates their state at discrete time steps in order to find an optimal or the best solution to the considered problem. In the classic formulation of the PSO algorithm each particle has the two properties: the position $x(t)$ and the velocity $v(t)$. To determine the new position the algorithm uses both above particle proprieties, its best position reached so far $p^L(t)$ and information about the best solution found by the whole swarm (or the particle neighborhood) $p^G(t)$. The state equation for a particle is given by the formula (??). The three coefficients c_1, c_2, c_3 appearing in the formula are called *inertia*, *cognition* (or *self recognition*) and *social* factors, respectively. Their typical values for continuous problems are the following: $c_1 \in [0.4, 0.9]$, $c_2 = 2$ and $c_3 = 2$. The r_1, r_2 are random numbers uniformly distributed in the $[0, 1]$ interval.

$$\begin{aligned} v(t+1) &= c_1 \cdot v(t) + c_2 \cdot r_2(t) \cdot (p^L(t) - x(t)) \; + c_3 \cdot r_3(t) \cdot (p^G(t) - x(t)) \\ x(t+1) &= x(t) + v(t) \end{aligned} \tag{3}$$

An adaptation of the PSO method to a discrete domain consists in giving interpretation to such concepts as position, velocity and neighborhood. Moreover, in some cases, equivalents of scalar addition, subtraction and multiplication for the arguments being solutions and velocities should be

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

defined. Examples of such interpretations can be found in [?] for the TSP and [?] for the QAP problem.

A PSO algorithm for solving the QAP problem using similar representations of particle state was proposed by Liu et al. [?]. Although the approach presented there was inspiring, the paper gives very little information on efficiency of the developed algorithm.

2.3. GPU based calculations

Recently many computationally demanding applications has been redesigned to exploit the capabilities offered by massively parallel computing GPU platforms. They include such tasks as: physically based simulations, signal processing, ray tracing, geometric computing and data mining [?]. Several attempts have been also made to develop various population based optimization algorithms on GPUs including: the particle swarm optimization [?], the ant colony optimization [?], the genetic [?] and memetic algorithm [?]. The described implementations benefit from capabilities offered by GPUs by processing whole populations by fast GPU cores running in parallel.

3. Algorithm design and implementation

In this section we describe the algorithm design, in particular the adaptation of Particle Swarm Optimization metaheuristic to the QAP problem, as well as a specific algorithm implementation on OpenCL platform. As it was stated in Section **??**, the PSO uses generic concepts of position x and velocity v that can be mapped to a particular problem in various ways. Designing an algorithm for a GPU platform requires decisions on how to divide it into parts that are either executed sequentially at the host side or in parallel on the device.

3.1. PSO adaptation for the QAP problem

In the presented approach a state of a particle is defined by a pair of matrices $(X_{n\times n}, V_{n\times n})$, representing its position and velocity, respectively. An assignment of facilities to locations is encoded by the permutation matrix $X = [x_{ij}]_{n\times n}$, whose elements x_{ij} are equal to 1, if j^{th} facility is assigned to i^{th} location, and take value 0 otherwise. The velocity V defines the moving direction of the particles in the solution space. In the case of discrete problems, a high positive value of the v_{ij} can be interpreted as an indication that the assignment $x_{ij} = 1$ should be made. Otherwise, if $v_{ij} \leq 0$, then $x_{ij} = 0$ should be preferred. The state of a particle reached in the t^{th} iteration is denoted by $(X(t), V(t))$. In each iteration it is updated according to formulas (**??**) and (**??**).

$$V(t+1) = S_v\left(c_1 \cdot V(t) + c_2 \cdot r_2(t) \cdot (P^L(t) - X(t)) + c_3 \cdot r_3(t) \cdot (P^G(t) - X(t))\right) \tag{4}$$

$$X(t+1) = S_x(X(t) + V(t)) \tag{5}$$

Parameters $r_2(t)$ and $r_3(t)$ are random numbers from $[0, 1]$ and are generated in each iteration for every particle separately. They are introduced to model a random choice between *inertia* - the movements in the previous direction (c_1), *self recognition* - the movements in the direction of the best solution found by the particle in the past (c_2) and *social behavior* - the movements in the direction of the the global best solution. All operators used in (**??**) and (**??**) are defined as the standard operators from the linear algebra domain.

To adapt the algorithm to particular needs of the QAP problem, instead of redefining them for a particular problem, see e.g. [?], we propose to use aggregation functions S_v and S_x.

The goal of the function S_v is to keep velocities within a reasonable range. A typical approach in the PSO algorithm implementations is to clamp values of velocity vector elements at a certain value $vmax$ to avoid infinite growth, which would result in explosion of particles positions [?]. Function S_v implementing this approach is referred as *raw* in Table **??**.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

In this particular algorithm implementation rather an opposite effect occurred more frequently: in the case of inertia factor less than 1, e.g. $c_1 = 0.5$, after a few iterations velocities of all particles tended to 0 and their positions converged to the best solution found earlier. This can be explained by observing that a particle position $X(t)$ is actually a permutation matrix, i.e. is filled with zeros and ones. Hence, during velocity updates according to formula (**??**), the differences $P^L(t) - X(t)$ and $P^G(t) - X(t)$ are always bounded and their contributions to final velocity vector are small. To alleviate such effect, we proposed another function that performs column normalization: for each j-th column the sum of absolute values of the elements $n_j = \sum_{i=1}^{n} |v_{ij}|$ is calculated and then the following assignment is made: $v_{ij} \leftarrow v_{ij}/n_j$. This function is referenced further as *norm*.

Update of particles positions in continuous PSO consists in adding their state components $X(t+1) = X(t) + V(t)$. The new solution is always feasible, i.e. it is possible to calculate the value of goal function for $X(t+1)$, although the solution itself may lie outside the bounds. In the discussed adaptation of the PSO algorithm to the discrete problem the sum of $X(t)$ and $V(t)$ may have values in $[-v_{max}, v_{max}+1]$, whereas feasible solutions can be only valid permutation matrices satisfying (**??**), i.e. having exactly one 1 in each row and column.

In formula (**??**) we introduced a function S_x that is responsible for converting $X(t) + V(t)$ into a permutation matrix. Speaking strictly, S_x is rather a procedure, as it incorporates some random choices. In our experiments three variants of S_x procedures were used:

1. $GlobalMax(X)$ – in n iterations, where n is the size of X, determines the pivot row r_m and column c_m, such that element $x_{r_m c_m}$ is the maximal element in unvisited part of the matrix X, than sets $x_{r_m c_m}$ to 1 and clears other elements in the row r_m and c_m [**?**].
2. $PickColumn(X)$ – picks randomly a pivot column c from X, than finds a maximum element $x_{r_m c}$ in the column c, replaces it by 1 and clears other elements in row r_m and column c.
3. $SecondTarget(X, Z, d)$ – similar to $GlobalMax(X)$, however during the first d iterations elements x_{ij}, such that $z_{ij} = 1$ are ignored (as the parameter Z the solution X from the last iteration is used (see Algorithm **??**).

The $SecondTarget(X, Z, d)$ procedure was introduced to avoid undesired behavior, which can be sometimes observed for $GlobalMax(X)$ [**?**]. In spite of the fact, that particles have velocities far from zero, often they got stuck. We we will discuss this effect on a small 3×3 example. Let us consider the following values of particle state components $X(t)$ and $V(t)$.

$$X(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad V(t) = \begin{bmatrix} 7 & 1 & 3 \\ 0 & 4 & 5 \\ 2 & 3 & 2 \end{bmatrix}$$

Calculation of matrix $X(t) + V(t)$ and than application of $GlobalMax$ yields the matrix $X(t+1)$, which is equal to previous position $X(t)$.

$$X(t) + V(t) = \begin{bmatrix} 8 & 1 & 3 \\ 0 & 4 & 6 \\ 2 & 4 & 2 \end{bmatrix} \quad S_x(X(t) + V(t)) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

In such situation, due to the deterministic character of $GlobalMax$, the solution $X(t)$ remains unchanged for several iterations, until another particle reaches a new global minimum, what in consequence modifies $(P^G(t) - X(t)))$ component of formula (**??**) for velocity calculation.

This problem can be resolved by altering the order in which pivot elements are chosen in consecutive iterations. An example is $PickColumn$ procedure, which is more robust, as it introduces some randomness while drawing columns. $SecondTarget$ algorithm follows another idea: it excludes form the selection process first high ranked $d < n$ elements that in the previous solution $X(t)$ were already set to 1. Hence, if d value is large enough, it directs to a second solution close to $X(t) + V(t)$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

Let us denote by $X \oslash_d V$ a pair $(X+V, P)$ where $P = SecondTarget(X+V, X, d)$. Possible values of $X \oslash_d V$, where $d = 1$ and $d = 2$, are shown below. Elements of a new solution P are marked with circles, whereas the upper index indicates the iteration, in which the element was chosen.

$$X \oslash_1 V = \begin{bmatrix} ⑧^2 & 1 & 3 \\ 0 & 4 & ⑥^1 \\ 2 & ④^3 & 2 \end{bmatrix}$$

$$X \oslash_2 V = \begin{bmatrix} 8 & 1 & ③^2 \\ ⓪^3 & 4 & 6 \\ 2 & ④^1 & 2 \end{bmatrix} \text{ or } X \oslash_2 V = \begin{bmatrix} 8 & 1 & ③^2 \\ 0 & ④^1 & 6 \\ ②^3 & 4 & 2 \end{bmatrix}$$

It can be observed that for $d = 1$ the value is exactly the same, as it would result from the *GlobalMax*, however setting $d = 2$ allows to reach a different solution. The pseudocode of *SecondTarget* procedure is listed in Algorithm **??**.

Algorithm 1 Aggregation procedure *SecondTarget*

Require: $X = Z + V$ - new solution requiring normalization
Require: Z - previous solution
Require: *depth* - number of iterations, in which during selection of maximum element the algorithm ignore positions, where corresponding element of Z is equal to 1

procedure SECONDTARGET(X,Z,*depth*)
  $R \leftarrow \{1, \ldots, n\}$
  $C \leftarrow \{1, \ldots, n\}$
  for i in (1,n) **do**
    Calculate M, the set of maximum elements
    if $i \leq depth$ **then**
      Ignore elements x_{ij} such that $z_{ij} = 1$
      $M \leftarrow \{(r,c) : z_{rc} \neq 1 \wedge \forall_{i \in R, j \in C, z_{ij} \neq 1}(x_{rc} \geq x_{ij})\}$
    else
      $M \leftarrow \{(r,c) : \forall_{i \in R, j \in C}(x_{rc} \geq x_{ij})\}$
    end if
    Randomly select (r,c) from M
    $R \leftarrow R \setminus \{r\}$ ▷ Update the sets R and C
    $C \leftarrow C \setminus \{c\}$
    for i in $(1,n)$ **do**
      $x_{ri} \leftarrow 0$ ▷ Clear r-th row
      $x_{ic} \leftarrow 0$ ▷ Clear c-th column
    end for
    $x_{rc} \leftarrow 1$ ▷ Assign 1 to the maximum element
  end for
  return X
end procedure

3.2. Migration

The intended goal of the migration mechanism is to improve the algorithm exploration capabilities by exchanging information between swarms. Actually, it is not a true migration, as particles do not move. Instead we modify stored $P^G[k]$ solutions (global best solution for a k-th swarm) replacing it by randomly picked solution from a swarm that performed better (see Algortithm **??**).

The newly set $P^G[k]$ value influences the velocity vector for all particles in k-th swarm according to the formula (**??**). It may happen that the goal function value corresponding to the assigned solution $P^G[k]$) is worse than the previous one. It is accepted, as the migration is primarily designed to increase diversity within swarms.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

Algorithm 2 Migration procedure

Require: d - migration depth satisfying $d < m/2$, where m is the number of swarms
Require: P^G - table of best solutions for m swarms
Require: X - set of all solutions
1: **procedure** MIGRATION(d,P^G,X)
2: Sort swarms according to their P^G_k values into a sequence $(s_1, s_2 \ldots, s_{m-2}, s_{m-1})$
3: **for** k in (1, d) **do**
4: Randomly choose a solution x_{kj} belonging to the swarm s_k
5: Assign $P^G[s_{m-k-1}] \leftarrow x_{kj}$
6: Update the best goal function value for the swarm $m - k - 1$
7: **end for**
8: **end procedure**

It should be mentioned that a naive approach consisting in copying best P^G values between the swarms would be incorrect. (Consider replacing line 5 of Algorithm **??** with: $P^G[s_{m-k-1}] \leftarrow P^G[s_k]$.) In such case during algorithm stagnation spanning over several iterations: in the first iteration the best value $P^G[1]$ would be cloned, in the second two copies would be created, in the third four and so on. Finally, after k iterations 2^k swarms would follow the same direction. In the first group of experiments reported in Section **??** we used up to 250 swarms. It means that after 8 iterations all swarms would be dominated by a single solution.

3.3. OpenCL algorithm implementation

The OpenCL [**? ?**] is a standard of parallel computing for heterogeneous platforms including GPU, multicore CPU, DSP and FPGA. It defines a common language, programming interfaces and hardware abstraction. OpenCL allows to accelerate computations by decomposing them into a set of parallel tasks called *work items*, which are typically scheduled to operate on separate data.

A program on the OpenCL platform is a combination of sequential code executed by the CPU *host* and parallel procedures called *kernels* executed by multicore *devices*. Kernels are written in a restricted C language; the restrictions concern keywords, datatypes and available library functions. The OpenCL provides automatic translation of kernels into the instruction set of the target device. The process occurs once, when they are first time loaded, and takes about 500ms.

The OpenCL supports 1D, 2D or 3D organization of data (arrays, matrices and volumes). Hence, data items can be addressed by 1 to 3 indexes and an address within the data range being a single index, a pair or a triple can be used to schedule a kernel instance denoted by the term *work item*. To give an example, a $m \times n$ array of data can be processed in parallel by $n \cdot m$ work items, which receive at their start a pair of indexes (i, j), $0 \leq i < m$ and $0 \leq j < n$. These indexes can be used to select data items assigned to kernels.

Work items can be organized into *workgroups*. Within a workgroup, they may share fast local memory and synchronize their activities using a *local barrier* mechanism. The OpenCL supports three types of memory access: global (that is exchanged between the host and the device), local for a work group and private for a work item. In spite of some advantages offered by the decomposition of processing into workgroups, we decided not use this mechanism due to several platform restrictions limiting the number of work items within a workgroup and amount of accessible memory.

The algorithm was implemented in Java language using *aparapi* platform [**?**], which provides the OpenCL bindings as well as a runtime capable of converting Java bytecodes into the OpenCL kernels. The host part of the program was executed on a Java virtual machine, and the kernel code, originally written in Java, was automatically translated into C code by the aparapi and further processed by the OpenCL for execution on a GPU device.

The basic functional blocks of the algorithm are presented in Fig. **??**. Implemented kernels are marked with gray color. They include particles position update (*Apply* S_x), velocities update (*Update*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

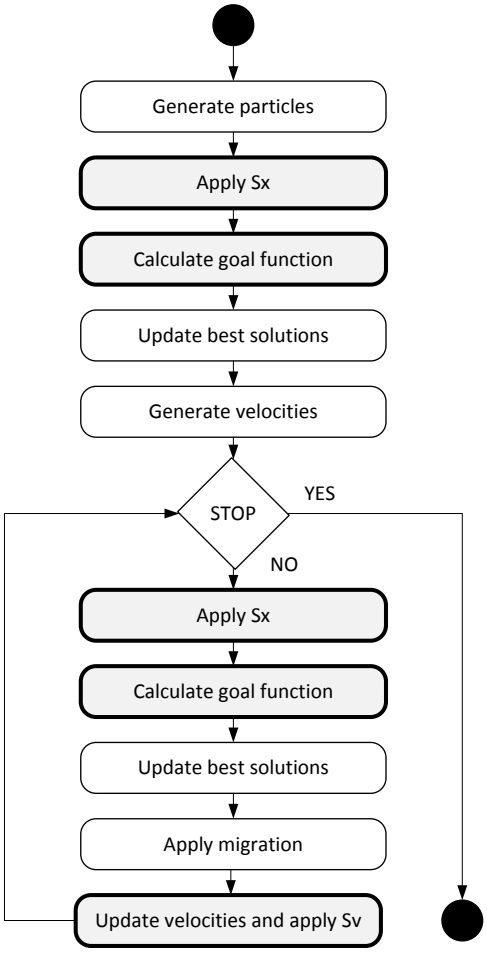

Figure 1. Functional blocks of OpenCL based algorithm implementation.

velocity and apply S_v) and calculation of the goal function. Other steps are implemented at the host side. This regards also the migration procedure, which sorts swarms indexes in a table according to their P^G value and copies randomly picked entries.

Data ranges selection is an important decision in OpenCL program design. Data used by a particle comprise a number of matrices (see Fig. **??**): X and X_{new} – solutions, P^L – local best particle solution and V – velocity. They are all stored in large flattened tables shared by all particles. Appropriate table part belonging to a particle can be identified based on the *particle id* transferred to a kernel. Moreover, while updating velocity, the particles reference a table P^G indexed by the *swarm id*.

The memory layout in Fig. **??** suggests 3D range, whose dimensions are: row, column and particle number. However, the proposed algorithms rather operate on the whole matrices than their parts. Therefore, we decided to use one dimension (particle id) for S_x and the goal function calculation, and the two dimensions (particle id, swarm id) for velocity kernels.

It should be mentioned that requirements of parallel processing limits applicability of object oriented design at the host side. In particular we avoid creating particles or swarms with their own memory and then copying small chunks between the host and the device. Instead we rather use a flyweight design pattern [**?**]. If a particle abstraction is needed, a single object can be configured to see parts of large global arrays X, V as its own memory and perform required operations, e.g. initialization with random values.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

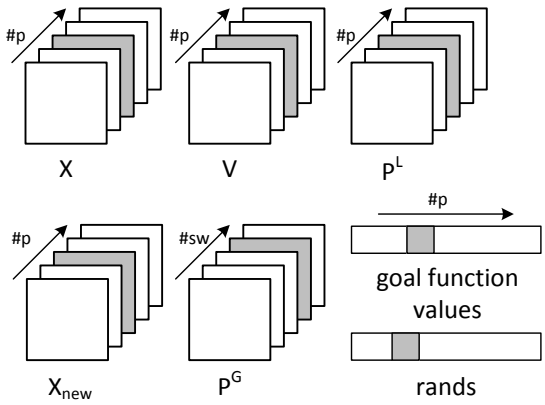

Figure 2. Global variables used in the algorithm implementation.

4. Experiments and results

In this section we report results of conducted experiments, which aimed at establishing the optimization performance of the implemented algorithm, as well as to collect data related to its statistical properties.

4.1. Optimization results

The algorithm was tested on several problem instances form the QAPLIB [**?**], whose size ranged between 12 and 150. Their results are gathered in Table **??** and Table **??**. The selection of algorithm configuration parameters (c_1, c_2 and c_3 factors, as well as the kernels used) was based on previous results published in [**?**]. In all cases the second target S_x aggregation kernel was applied (see Algorithm **??**), which in previous experiments occurred the most successful.

During all tests reported in Table **??**, apart the last, the total numbers of particles were large: 10000-12500. For the last case only 2500 particles were used due to the 1GB memory limit of the GPU device (AMD Radeon HD 6750M card). In this case the consumed GPU memory ranged about 950 MB.

The results show that algorithm is capable of finding solutions with goal function values are close to reference numbers listed in the QAPLIB. The gap is between 0% and 6.4% for the biggest case *tai150b*. We have repeated tests for *tai60b* problem to compare the implemented multi-swarm algorithm with the previous single-swarm version published in [**?**]. Gap values for the best results obtained with the single swarm algorithm were around 7%-8%. For the multi-swarm implementation discussed here the gaps were between 0.64% and 2.03%.

The goal of the second group of experiments was to test the algorithm configured to employ large numbers of particles (50 000-100 000) for well known *esc32** problem instances from the QAPLIB. Altough they were considered hard, all of them have been recently solved optimally with exact algorithms [**? ?**].

The results are summarized in Table **??**. We used the following parameters: $c_1 = 0.8$, $c_2 = 0.5$ $c_3 = 0.5$, velocity kernel: normalized, S_x kernel: second target. During nearly all experiments optimal values of goal functions were reached in one algorithm run. Only the problem *esc32a* occurred difficult, therefore for this case the number of particles, as well as the upper iteration limits were increased to reach the optimal solution. What was somehow surprising, in all cases solutions differing from those listed in the QAPLIB were obtained. Unfortunately, our algorithm was not prepared to collect sets of optimal solutions, so we are not able to provide detailed results on their numbers.

It can be seen that optimal solutions for problem instances *esc32c–h* were found in relatively small numbers of iterations. In particular, for *esc32e* and *es32g*, which are characterized by small values of goal functions, optimal solutions were found during the initialization or in the first iteration.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

Table 1. Results of tests for various instance from the QAPLIB.

No	Instance	Size	Number of swarms	Swarm size	Total particles	Inertia c_1	Self recognition c_2	Social factor c_3	Velocity kernel	Migration factor	Reached goal	Reference value	Gap	Iteration
1	chr12a	12	200	50	10000	0.5	0.5	0.5	Norm	0	9 552	9 552	0.00%	21
2	bur26a	26	250	50	12500	0.8	0.5	0.5	Raw	33%	5 426 670	5 426 670	0.00%	156
3	bur26a	26	250	50	12500	0.8	0.5	0.5	Raw	0	5 429 693	5 426 670	0.06%	189
4	lipa50a	50	200	50	10000	0.5	0.5	0.5	Norm	0	62 794	62 093	1.13%	1640
5	tai60a	60	200	50	10000	0.8	0.5	0.5	Norm	0	7 539 614	7 205 962	4.63%	817
6	tai60a	60	300	50	15000	0.8	0.5	0.5	Raw	0	7 426 672	7 205 962	3.06%	917
7	tai60b	60	200	50	10000	0.8	0.3	0.3	Norm	0	620 557 952	608 215 054	2.03%	909
8	tai60b	60	200	50	10000	0.5	0.5	0.5	Norm	0	617 825 984	608 215 054	1.58%	1982
9	tai60b	60	200	50	10000	0.8	0.3	0.3	Norm	33%	612 078 720	608 215 054	0.64%	2220
10	tai60b	60	200	50	10000	0.8	0.3	0.3	Norm	33%	614 088 768	608 215 054	0.97%	1619
11	tai64c	64	200	50	10000	0.8	0.5	0.5	Norm	0	1 856 396	1 855 928	0.03%	228
12	esc64a	64	200	50	10000	0.8	0.5	0.5	Raw	0	116	116	0.00%	71
13	tai80a	80	200	50	10000	0.8	0.5	0.5	Raw	0	14 038 392	13 499 184	3.99%	1718
14	tai80b	80	200	50	10000	0.8	0.5	0.5	Raw	0	835 426 944	818 415 043	2.08%	1509
15	sko100a	100	200	50	10000	0.8	0.5	0.5	Raw	0	154 874	152 002	1.89%	1877
16	tai100b	100	200	50	10000	0.8	0.5	0.5	Raw	0	1 196 819 712	1 185 996 137	0.91%	1980
17	esc128	128	100	50	5000	0.8	0.5	0.5	Raw	0	64	64	0.00%	1875
18	tai150b	150	50	50	2500	0.8	0.5	0.5	Raw	0	530 816 224	498 896 643	6.40%	1894

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

Table 2. Results of tests for *esc32** instances from the QAPLIB (problem size $n = 32$). Reached optimal values are marked with asterisks.

Instance	Swarms	Particles	Total part.	Goal	Iter	Time/iter [ms]
esc32a	50	1000	100000	138	412	3590.08
esc32a	10	5000	100000	134	909	3636.76
esc32a	50	2000	100000	130*	2407	3653.88
esc32b	50	1000	50000	168*	684	3637.84
esc32c	50	1000	50000	642*	22	3695.19
esc32d	50	1000	50000	400*	75	3675.32
esc32e	50	1000	50000	2*	0	3670.38
esc32g	50	1000	50000	6*	1	3625.17
esc32h	50	1000	50000	438*	77	3625.17

The disadvantage of the presented algorithm is that it uses internally matrix representation for solutions and velocities. In consequence the memory consumption is proportional to n^2, where n is the problem size. The same regards the time complexity, which for goal function and S_x procedures can be estimated as $o(n^3)$. This makes optimization of large problems time consuming (e.g. even 400 sec for one iteration for *tai150b*). However, for for medium size problem instances, the iteration times are much smaller, in spite of large populations used. For two runs of the algorithm *bur26a* reported in Table **??**, where during each iteration 12500 particles were processed, the average iteration time was equal 1.73 sec. For 50000-10000 particles and problems of size $n = 32$ the average iteration time reported in Table **??** was less than 3.7 seconds.

4.2. Statistical results

An obvious benefit of massive parallel computations is the capability of processing large populations (see Table **??**). Such approach to optimization may resemble a little bit a *brutal force* attack: the solution space is randomly sampled millions of times to hit the best solution. No doubt that such approach can be more successful if combined with a correctly designed exploration mechanism that directs the random search process towards solutions providing good or near-optimal solutions. In this section we analyze collected statistical data related to the algorithm execution to show that the optimization performance of the algorithm can be attributed not only to large sizes of processed population, but also to the implemented exploration mechanism.

The PSO algorithm can be considered a stochastic process controlled by random variables $r_2(t)$ and $r_3(t)$ appearing in its state equation (**??**). Such analysis for continuous problems were conducted in [**?**]. On the other hand, the observable algorithm outcomes, i.e. the values of goal functions $f(x_i(t))$ for solutions x_i, $i = 1, \ldots, n$ reached in consecutive time moments $t \in \{1, 2, 3, \ldots\}$ can be also treated as random variables, whose distributions change over time t. Our intuition is that a correctly designed algorithm should result in a nonstationary stochastic process $\{f(x_i(t)) \colon t \in T\}$, characterized by growing probability that next values of goal functions in the analyzed population are closer to the optimal solution.

To demonstrate such behavior of the implemented algorithm we have collected detailed information on goal function values during two optimization task for the problem instance *bur26a* reported in Table **??** (cases 2 and 3). For both of them the algorithm was configured to use 250 swarms comprising 50 particles. In the case 2 the migration mechanism was applied and the optimal solution was found in the iteration 156, in the case 3 (without migration) a solution very close to optimal (gap 0.06%) was reached in the iteration 189.

Fig. **??** shows values of goal function for two selected particles during run 3. The plots show the typical QAP problem specificity. The PSO algorithm and many other algorithms perform a local neighborhood search. For the QAP problem the neighborhood is characterized by great variations of goal function values. Although mean values of goal function decrease in first twenty or thirty

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

iterations, the particles behave randomly and nothing indicates that during subsequent iterations smaller values of goal functions would be reached more often.

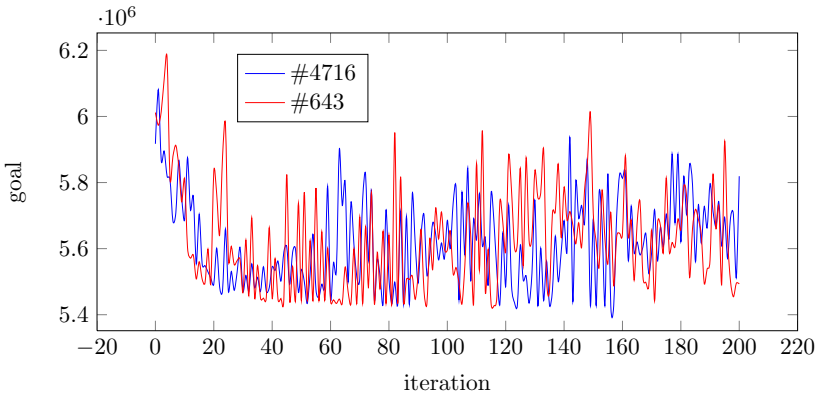

Figure 3. Variations of goal function values for two particles exploring the solutions space during the optimization process (bur26a problem instance).

In Fig. **??** percentile ranks (75%, 50% 25% and 5%) for two swarms, which reached best values in cases 2 and 3 are presented. Although the case 3 is characterized by less frequent changes of scores, than the case 2, probably this effect can not be attributed to the migration applied. It should be mentioned that for a swarm comprising 50 particles, the 0.05 percentile corresponds to just two of them.

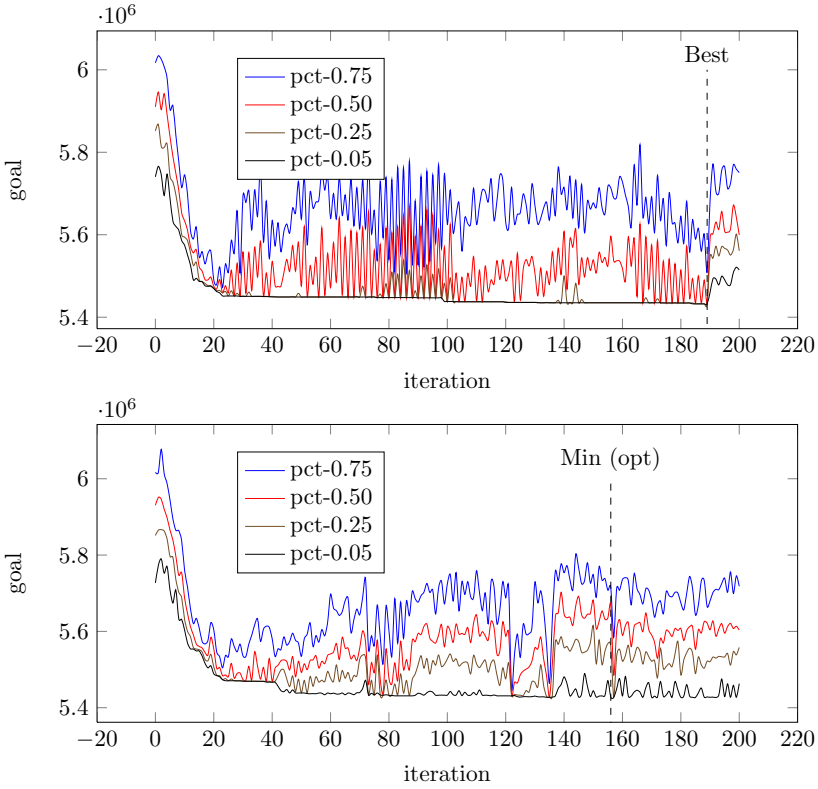

Figure 4. Two runs of bur26a optimization. Percentile ranks for 50 particles belonging to the most successful swarms: without migration (above) and with migration (below).

Collected percentile rank values for the whole population comprising 12500 particles are presented in Fig. **??**. For both cases the plots are clearly separated. It can be also observed that solutions very

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

close to optimal are practically reached between the iterations 20 (37.3 sec) and 40 (72.4 sec). For the whole population the 0.05 percentile represents 625 particles. Starting with the iteration 42 their score varies between $5.449048 \cdot 10^6$ and $5.432361 \cdot 10^6$, i.e. by about 0.3%.

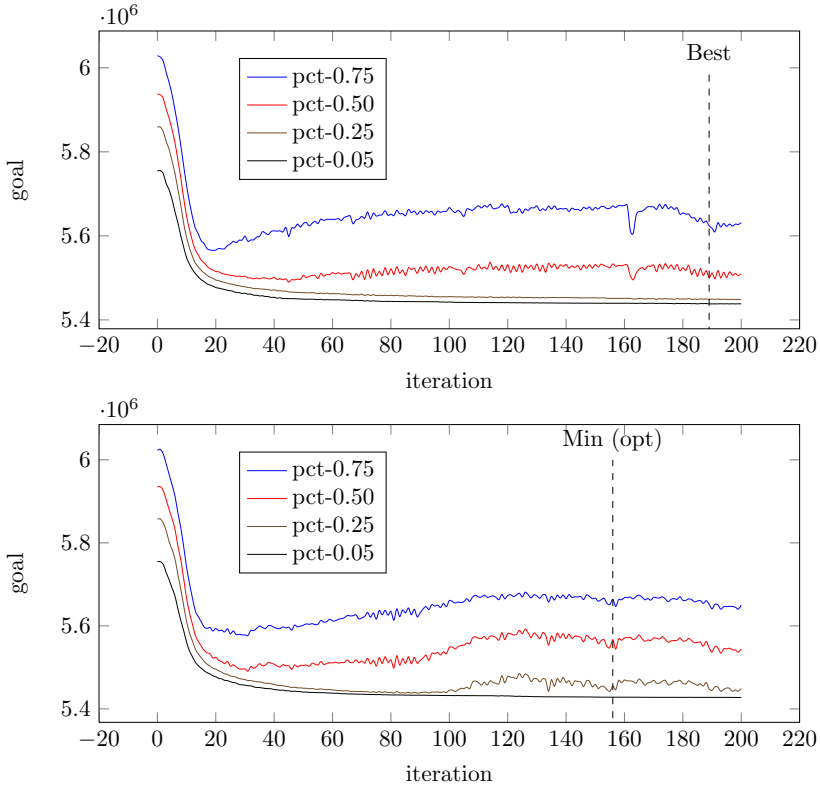

Figure 5. Two runs of bur26a optimization. Percentile ranks for all 12500 particles: without migration (above) and with migration (below).

Fig. **??** shows, how the probability distribution (the probability mass function - the PMF) changed during the optimization process. In both cases the the optimization process starts with a normal distribution with the mean value about 594500. In the subsequent iterations the maximum of the PMF grows and moves towards smaller values of the goal function. There is no fundamental difference between the two cases, however for the case 3 (with migration) maximal values of the PMF are higher. It can be also observed that in the iteration 30 (completed in 56 seconds) the probability of hitting a good solution is quite high, more then 10%.

Interpretation of the PMF for the two most successful swarms that reached best values in the discussed cases is not that obvious. For the case without migration (Fig. **??** above) there is a clear separation between the initial distribution and the distribution reached in the iteration, which yielded the best result. In the second case (with migration) a number of particles were concentrated around local minima.

The presented data shows advantages of optimization performed on massive parallel processing platforms. Due to high number of solutions analyzed simultaneously, the algorithm that does not exploit the problem structure can yield acceptable results in relatively small number of iterations (and time). For a low-end GPU devices, which was used during the test, *good enough* results were obtained after 56 seconds. It should be mentioned that for both presented cases the maximum number of iterations was set to 200. With 12500 particles, the ratio of potentially explored solutions to the whole solution space was equal $200 \cdot 12500/26! = 6.2 \cdot 10^{-21}$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 January 2018 doi:10.20944/preprints201801.0230.v1

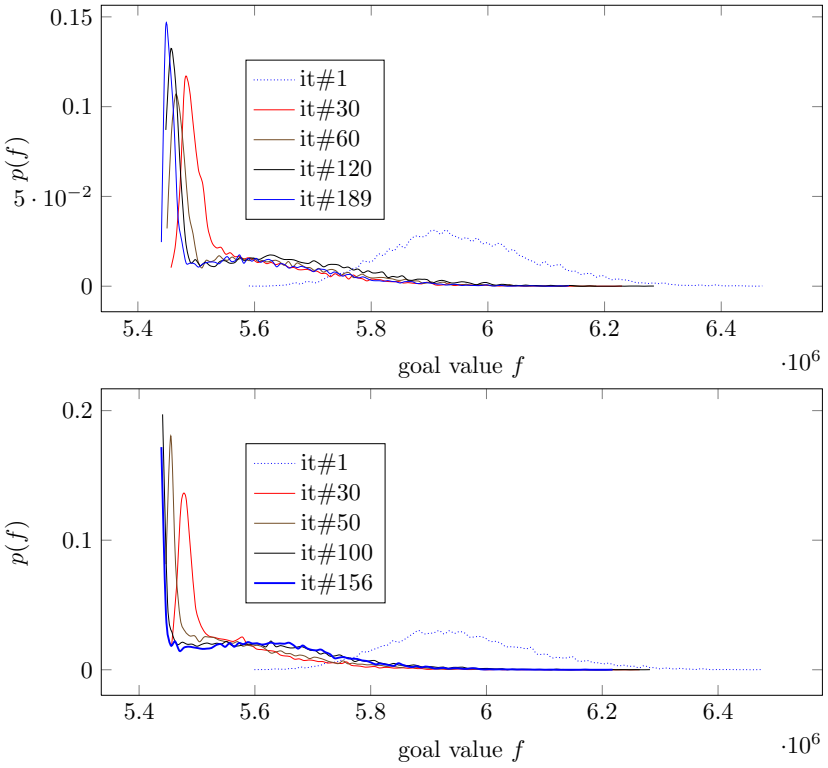

Figure 6. Probability mass functions for 12050 particles organized into 250 x 50 swarms during two runs: without migration (above) and with migration (below).

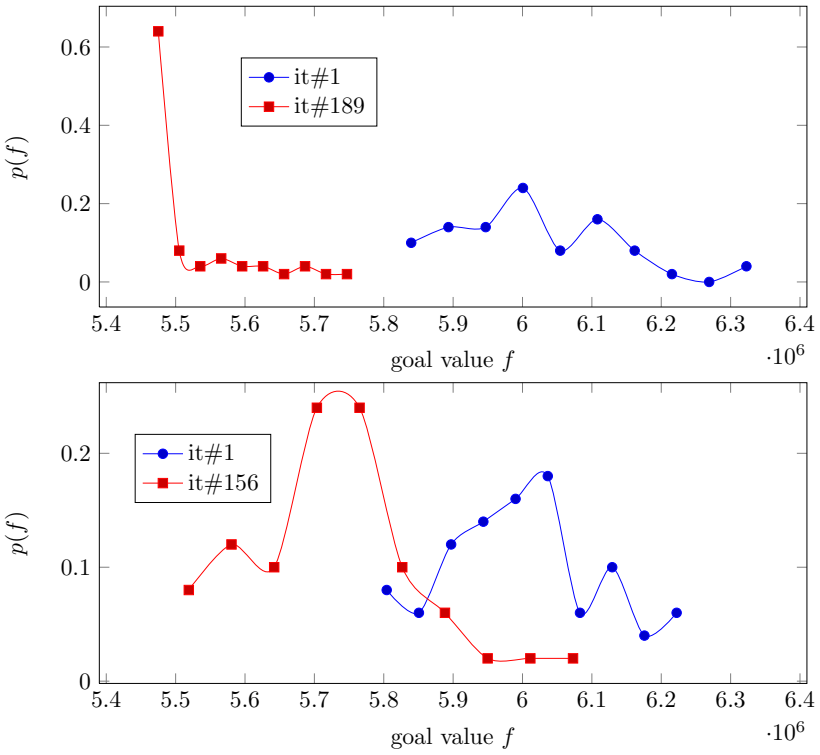

Figure 7. Probability mass functions for 50 particles belonging to the most successful swarms during two runs: without migration (above) and with migration (below). One point represents an upper bound for 5 particles.

 doi:10.20944/preprints201801.0230.v1

5. Conclusions

In this paper we describe a multi-swarm PSO algorithm for solving the QAP problem designed for the OpenCL platform. The algorithm is capable of processing in parallel large number of particles organized into several swarms that either run independently or communicate with use of the migration mechanism. Several solutions related to particle state representation and particle movement were inspired by the work of Liu at al. [**?**], however, they were refined here to provide better performance.

We tested the algorithm on several problem instances from the QAPLIB library obtaining good results (small gaps between reached solutions and reference values). However, it seems that for problem instances of large sizes the selected representation of solutions in form of permutation matrices hinders potential benefits of parallel processing.

During the experiments the algorithm was configured to process large populations. This allowed us to collect statistical data related to goal function values reached by individual particles. We used them to demonstrate on two cases that although single particles seem to behave chaotically during the optimization process, when the whole population is analyzed, the probability that a particle will select a near-optimal solution grows. This growth is significant for a number of initial iterations, then its speed diminishes and finally reaches zero.

Statistical analysis of experimental data collected during optimization process may help to tune the algorithm parameters, as well as to establish realistic limits related to expected improvement of goal functions. This in particular regards practical applications of optimization techniques, in which recurring optimization problems appear, i.e. the problems with similar size, complexity and structure. Such problems can be near-optimally solved in bounded time on massive parallel computation platforms even, if low-end devices are used.

References

. Koopmans, T.C.; Beckmann, M.J. Assignment problems and the location of economic activities. *Econometrica* **1957**, *25*, 53–76.

. Çela, E. *The quadratic assignment problem: theory and algorithms*; Combinatorial Optimization, Springer: Boston, 1998.

. Chmiel, W.; Kadłuczka, P.; Kwiecień, J.; Filipowicz, B. A comparison of nature inspired algorithms for the quadratic assignment problem. *Bulletin of the Polish Academy of Sciences Technical Sciences* **2017**, *65*, 513–523.

. Bermudez, R.; Cole, M.H. A Genetic Algorithm Approach to Door Assignments in Breakbulk Terminals. Technical Report MBTC-1102, Mack-Blackwell Transportation Center, University of Arkansas, Fayetteville, Arkansas, 2001.

. Mason, A.; Rönnqvist, M. Solution methods for the balancing of jet turbines. *Computers & OR* **1997**, *24*, 153–167.

. Grötschel, M. Discrete Mathematics in Manufacturing. ICIAM 1991: Proceedings of the Second International Conference on Industrial and Applied Mathematics; Malley, R.E.O., Ed. SIAM, 1991, pp. 119–145.

. Sahni, S.; Gonzalez, T. P-Complete Approximation Problems. *J. ACM* **1976**, *23*, 555–565.

. Taillard, E.D. Comparison of iterative searches for the quadratic assignment problem. *Location Science* **1995**, *3*, 87 – 105.

. Misevicius, A. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum* **2012**, *34*, 665–690.

. Chmiel, W.; Kadłuczka, P.; Packanik, G. Performance Of Swarm Algorithms For Permutation Problems. *Automatyka* **2009**, *15*, 117–126.

. Onwubolu, G.C.; Sharma, A. Particle Swarm Optimization for the assignment of facilities to locations. In *New Optimization Techniques in Engineering*; Springer, 2004; pp. 567–584.

. Liu, H.; Abraham, A.; Zhang, J. A Particle Swarm Approach to Quadratic Assignment Problems. In *Soft Computing in Industrial Applications*; Saad, A.; Dahal, K.; Sarfraz, M.; Roy, R., Eds.; Springer Berlin Heidelberg, 2007; Vol. 39, *Advances in Soft Computing*, pp. 213–222.

391 . Eberhart, R.; Kennedy, J. A new optimizer using particle swarm theory. Micro Machine and Human
392 Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on, 1995, pp. 39–43.
393 . Szwed, P.; Chmiel, W.; Kadłuczka, P. OpenCL implementation of PSO algorithm for the Quadratic
394 Assignment Problem. In *Artificial Intelligence and Soft Computing*; Rutkowski, L.; Korytkowski, M.; Scherer,
395 R.; Tadeusiewicz, R.; Zadeh, L.A.; Zurada, J.M., Eds.; Springer International Publishing, 2015; Vol. Accepted
396 for ICAISC'2015 Conference, *Lecture Notes in Computer Science*.
397 . Peter Hahn and Miguel Anjos. QAPLIB Home Page. http://anjos.mgi.polymtl.ca/qaplib/. Online: last
398 accessed: Jan 2015.
399 . Fischetti, M.; Monaci, M.; Salvagnin, D. Three Ideas for the Quadratic Assignment Problem. *Operations*
400 *Research* **2012**, *60*, 954–964.
401 . Anstreicher, K.; Brixius, N.; Goux, J.P.; Linderoth, J. Solving large quadratic assignment problems on
402 computational grids. *Mathematical Programming* **2002**, *91*, 563–588.
403 . Phillips, A.T.; Rosen, J.B. A Quadratic Assignment Formulation of the Molecular Conformation Problem.
404 *JOURNAL OF GLOBAL OPTIMIZATION* **1994**, *4*, 229–241.
405 . Burkard, R.E.; Karisch, S.E.; Rendl, F. QAPLIB - A Quadratic Assignment Problem Library. *Journal of Global*
406 *Optimization* **1997**, *10*, 391–403.
407 . Hahn, P.M.; Zhu, Y.R.; Guignard, M.; Smith, J.M. Exact solution of emerging quadratic assignment
408 problems. *International Transactions in Operational Research* **2010**, *17*, 525–552.
409 . Hahn, P.; Roth, A.; Saltzman, M.; Guignard, M. Memory-Aware Parallelized RLT3 for solving Quadratic
410 Assignment Problems. *Optimization online* **2013**.
411 . Ahuja, R.K.; Orlin, J.B.; Tiwari, A. A greedy genetic algorithm for the quadratic assignment problem.
412 *Computers & Operations Research* **2000**, *27*, 917–934.
413 . Stützle, T.; Dorigo, M. ACO algorithms for the quadratic assignment problem. *New ideas in optimization*
414 **1999**, pp. 33–50.
415 . Gambardella, L.M.; Taillard, E.; Dorigo, M. Ant colonies for the quadratic assignment problem. *Journal of*
416 *the operational research society* **1999**, pp. 167–176.
417 . Fon, C.W.; Wong, K.Y. Investigating the performance of bees algorithm in solving quadratic assignment
418 problems. *International Journal of Operational Research* **2010**, *9*, 241–257.
419 . Clerc, M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New*
420 *optimization techniques in engineering*; Springer, 2004; pp. 219–239.
421 . Owens, J.D.; Luebke, D.; Govindaraju, N.; Harris, M.; Krüger, J.; Lefohn, A.E.; Purcell, T.J. A Survey of
422 general-purpose computation on graphics hardware. Computer graphics forum. Wiley Online Library,
423 2007, Vol. 26, pp. 80–113.
424 . Zhou, Y.; Tan, Y. GPU-based parallel particle swarm optimization. Evolutionary Computation, 2009.
425 CEC'09. IEEE Congress on. IEEE, 2009, pp. 1493–1500.
426 . Tsutsui, S.; Fujimoto, N. ACO with Tabu Search on GPUs for Fast Solution of the QAP. In *Massively Parallel*
427 *Evolutionary Computation on GPGPUs*; Tsutsui, S.; Collet, P., Eds.; Natural Computing Series, Springer Berlin
428 Heidelberg, 2013; pp. 179–202.
429 . Maitre, O. Genetic Programming on GPGPU Cards Using EASEA. In *Massively Parallel Evolutionary*
430 *Computation on GPGPUs*; Tsutsui, S.; Collet, P., Eds.; Natural Computing Series, Springer Berlin Heidelberg,
431 2013; pp. 227–248.
432 . Krüger, F.; Maitre, O.; Jiménez, S.; Baumes, L.A.; Collet, P. Generic Local Search (Memetic) Algorithm on a
433 Single GPGPU Chip. In *Massively Parallel Evolutionary Computation on GPGPUs*; Tsutsui, S.; Collet, P., Eds.;
434 Natural Computing Series, Springer Berlin Heidelberg, 2013; pp. 63–81.
435 . Bratton, D.; Kennedy, J. Defining a standard for particle swarm optimization. Swarm Intelligence
436 Symposium, 2007. SIS 2007. IEEE. IEEE, 2007, pp. 120–127.
437 . Howes, L.; Munshi, A. The OpenCL Specification. https://www.khronos.org/registry/cl/specs/opencl-2.
438 0.pdf. Online: last accessed: Jan 2015.
439 . Stone, J.E.; Gohara, D.; Shi, G. OpenCL: A parallel programming standard for heterogeneous computing
440 systems. *Computing in science & engineering* **2010**, *12*, 66.
441 . Howes, L.; Munshi, A. Aparapi - AMD. http://developer.amd.com/tools-and-sdks/opencl-zone/
442 aparapi/. Online: last accessed: Jan 2015.

 doi:10.20944/preprints201801.0230.v1

443 . Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*;
444 Pearson Education, 1994.
445 . Nyberg, A.; Westerlund, T. A new exact discrete linear reformulation of the quadratic assignment problem.
446 *European Journal of Operational Research* **2012**, *220*, 314–319.
447 . Fernńdez-Martínez, J.; Garcńa Gonzalo, E. The PSO family: deduction, stochastic analysis and comparison.
448 *Swarm Intelligence* **2009**, *3*, 245–273.