BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 60, No. 2, 2012
DOI: 10.2478/v10175-012-0043-4

GPU-based tuning of quantum-inspired genetic algorithm for a combinatorial optimization problem

R. NOWOTNIAK* and J. KUCHARSKI

Computer Engineering Department, Technical University of Łódź, 18/22 Stefanowskiego St., 90-924 Łódź, Poland

Abstract. This paper concerns efficient parameters tuning (meta-optimization) of a state-of-the-art metaheuristic, Quantum-Inspired Genetic Algorithm (QIGA), in a GPU-based massively parallel computing environment (NVidia CUDA™technology). A novel approach to parallel implementation of the algorithm has been presented. In a block of threads, each thread transforms a separate quantum individual or different quantum gene; In each block, a separate experiment with different population is conducted. The computations have been distributed to eight GPU devices, and over 400× speedup has been gained in comparison to Intel Core i7 2.93GHz CPU. This approach allows efficient meta-optimization of the algorithm parameters. Two criteria for the meta-optimization of the rotation angles in quantum genes state space have been considered. Performance comparison has been performed on combinatorial optimization (knapsack problem), and it has been presented that the tuned algorithm is superior to Simple Genetic Algorithm and to original QIGA algorithm.

Key words: quantum-inspired genetic algorithm, evolutionary computing, meta-optimization, parallel algorithms, GPGPU.

1. Introduction

Quantum-Inspired Genetic Algorithm [1, 2] (QIGA) belongs to a new class of artificial intelligence techniques, drawing inspiration from both evolutionary [3] and quantum [4] computing. The algorithm is characterized by algorithmic operators mimicking computationally useful aspects of both the biological evolution and unitary evolution of quantum systems. QIGA algorithm is based on quantum mechanics concepts including qubits and superposition of states. Evolutionary computing methods can be applied to a broad range of search and optimization problems [5–7]. QIGA algorithm have demonstrated its particular efficacy for solving complex optimization problems. Recent years have witnessed successful applications of Quantum-Inspired Genetic Algorithms in variety of fields, including image processing [8–10], flow shop scheduling [11, 12], thermal unit commitment [13, 14], power system optimization [15, 16], localization of mobile robots [17] and many others. For a current and comprehensive survey of Quantum-Inspired Genetic Algorithms, the reader is referred to [18].

Modern advanced metaheuristics, such as Quantum-Inspired Genetic Algorithms, are usually characterized with numerous real or discrete parameters, e.g. population size, termination condition, number of generations, probability of mutation. Particularly, additional elements of randomness bring a "new dimension" into QIGAs. Thus, the algorithms are characterized by additional parameters such as rotation angles in the state space of genes modelled with qubits. Finding the right values in this multidimensional parameters space is laborious task, and it has a significant impact on the performance of the algorithm. Finding the best parameters can be treated as an optimization problem in itself, and this meta-optimization [19–21] process requires substantial computing power. Fortunately, the modern massively parallel computing environments (NVidia CUDA technology [22–24]) provide sufficient resources that allows tuning the algorithm with population-based heuristics automatically.

This paper is structured as follows. In Sec. 1, Quantum-Inspired Genetic Algorithm has been briefly presented. In Sec. 2, the proposed approach to parallelization implementation of the algorithm in NVidia CUDA architecture has been described. In Sec. 3, the concept of meta-optimization (parameters tuning) has been presented. In Sec. 4, experimental results have been provided and evaluated. In Sec. 5, the article has been briefly summarized, and final conclusions have been drawn.

2. Quantum-Inspired Genetic Algorithm

In QIGA algorithm[2], a novel representation of solutions, namely binary quantum coding, is employed. Instead of bits, *quantum genes* are modelled upon the concept of *qubits*, which brings an additional element of randomness and a "new dimension" into the algorithm. Qubit is a basic unit of quantum information, and it is a normalised vector in a two dimensional vector space spanned by the base vectors $|0\rangle$ and $|1\rangle$, as given in Eq. (1):

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

where $\alpha, \beta \in \mathbb{C}$ – probability amplitudes, $|0\rangle = [1 \quad 0]^T$, $|1\rangle = [0 \quad 1]^T$ and $|\alpha|^2 + |\beta|^2 = 1$. Observation of the quantum gene $|\Psi\rangle$ yields value 0 with probability $|\alpha|^2$ and value 1 with probability $|\beta|^2$.

With some simplification (imaginary element neglected), a state of binary quantum gene $|\Psi\rangle$ can depicted as a unit vector which has been presented in Fig. 1. Along with increase of

*e-mail: rnowotniak@kis.p.lodz.pl

the angle between the vector and the horizontal axis, the probability of observing value 1 grows, while the more horizontal direction of the vector, the higher probability of observing value 0. QIGA algorithm uses *binary quantum chromosomes* for representation of solutions, encoded as:

$$q = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{array}\right], \qquad (2)$$

where m denotes the length of the quantum chromosome q and each column corresponds to binary quantum gene $|\Psi\rangle_1, \dots, |\Psi\rangle_m$. Consequently, the state of the whole *quantum population* $Q = \{q_1, q_2, \dots, q_N\}$ can be simply illustrated by a matrix of vectors[25], which has been presented in Fig. 2. Each row in the figure corresponds to binary quantum chromosome, as in Eq. (2).

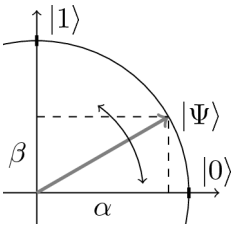

Fig. 1. Illustration of binary quantum gene state

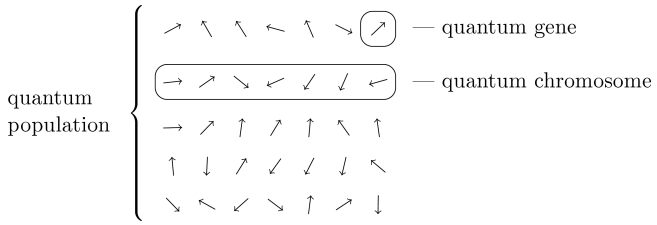

Fig. 2. Illustration of binary quantum population. Each arrow represent a state of a quantum gene

The full pseudocode of QIGA algorithm has been presented in Listing 1.

Listing 1 Quantum-inspired Genetic Algorithm

```
procedure QIGA
begin
t ← 0
initialize Q(0)
make P(0) by observing Q(0)
evaluate P(0)
store the best solution among P(0) in b
while not termination-criterion do
  t ← t + 1
  make P(t) by observing Q(t − 1) states
  evaluate P(t)
  update Q(t) using quantum gates U(θ_t)
  store the best solution among P(t) in b
end while
end
```

In the beginning of the algorithm, the genes of all individuals in the quantum population $Q(0)$ are initialized with linear superposition of states $\left(\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle\right)$. During the phenotype creation, states of all genes in quantum chromosomes are observed, i.e. the search space is sampled with respect to the probability distribution encoded in the quantum chromosomes. The evaluation of individuals' fitness is based on the observed classical population. The genetic operators applied in the algorithm are based on unitary *quantum rotation gates* $U(\theta_t)$, which rotate state vectors in the quantum gene state space. Proper rotation direction and angles are highly critical for efficiency of the algorithm, and their optimum values can be found either in manual time-consuming experimentation, or in an automated meta-optimization process [20, 26].

3. Implementation in NVidia CUDA architecture

In recent years, programmable Graphics Processing Units have evolved into massively parallel, multithreaded and many-core environments with tremendous computational power and high memory bandwidth [24, 27]. One of the leading General-Purpose Computing on Graphics Processing Units (GPGPU) nowadays is NVIDIA Compute Unified Device Architecture [22, 28] (CUDA™) technology. CUDA-enabled GPUs have hundreds of cores that can concurrently run thousands of computing threads. NVidia CUDA™technology has been already successfully applied in a vast number of different fields; For example, linear algebra [29, 30], signals processing [31], scientific simulations [32, 33], finance [34, 35] and others [36]. It is possible by the addition of programmable stages to the rendering pipelines, which allows programmers to use powerful parallel processing on non-graphics data.

Despite numerous impressive applications of General-Purpose Computing on Graphics Processing Units presented in the literature, the speedups reported in many papers should be taken with caution. In particular, parameters of the computing environment against which the comparison is made should be considered carefully. Many modern CPU processors have 4 computational cores and HyperThreading technology. Thus, if a comparison with single-core CPU is made, a factor of 6x – 8x should be taken off. Because GPU cards are often bought specifically on purpose of the research, the CPU can be a few years older. Therefore, a factor of 2x for each year of difference between CPU and GPU should be taken off. Also, the precision of floating point numbers should be taken into account. On most GPU cards, calculations on double precision numbers can be a few times slower in comparison to single precision. Consequently, the speed comparisons reported in many papers should be reduced by a factor of up to 30x.

In CUDA™, processing threads are grouped in blocks, and blocks constitutes a two-dimensional grid, which has been presented in Fig. 3. To utilize tremendous processing capabilities of modern GPU units, either existing libraries can be used for common operations of selected algorithms (like matrix multiplication, linear algebra, Fourier transform etc), or

one's own computational kernels can be written. In this paper, the second approach has been taken, and QIGA algorithm has been implemented entirely as a computational kernel running on GPU.

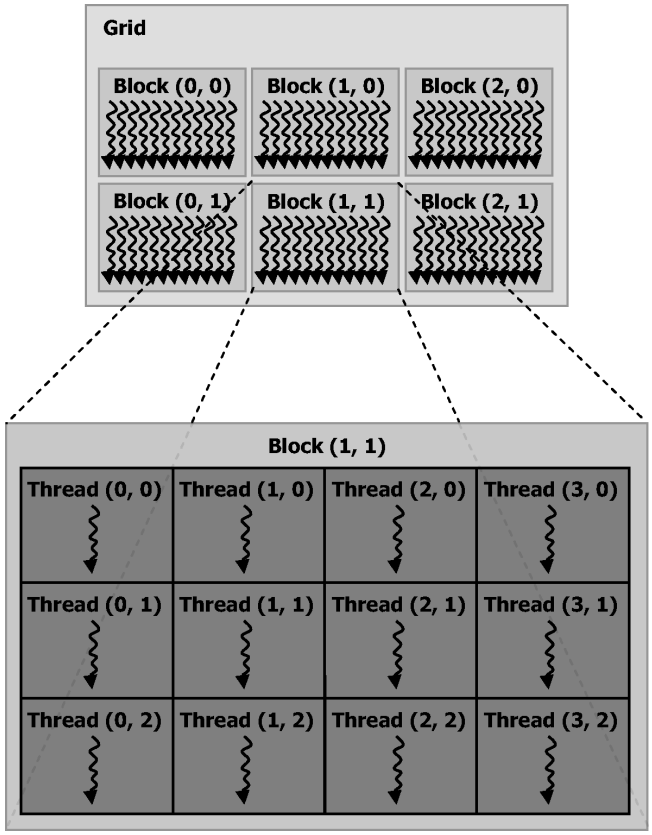

Fig. 3. Example of threads block (3×4) in the grid (2×3) on CUDA. Image source: [22]

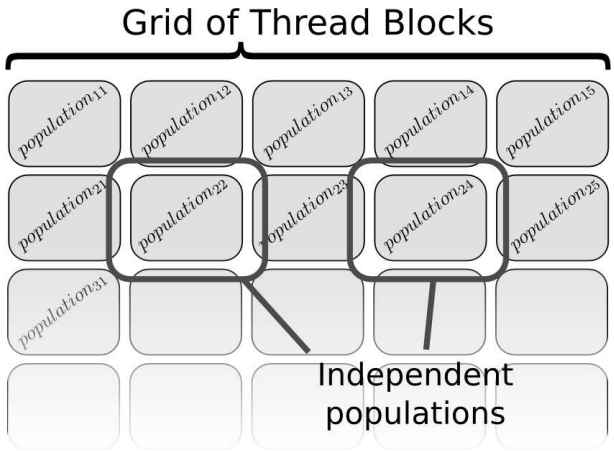

Fig. 4. Proposed approach to parallelization of the experimentation procedure

In several recent papers (e.g. [37–40]), successful GPU-based implementations of various metaheuristics have been presented. Usually, separate threads have been assigned to transformation and evaluation of separate individuals. However, this approach is particularly efficient for big populations only, i.e. several hounded of individuals, which is suitable only for specific optimization or search problems. In our approach, parallelization has been performed on two levels, which has been presented in Fig. 4. In a block of threads, each thread transforms a separate quantum individual or different quantum gene; In each block, a separate experiment with different population is conducted. If evaluation of the fitness function does not involve processing large amounts of data, essential data structures can be often stored entirely in the very fast shared memory (on-chip memory in GPU Streaming Multiprocessors). This makes the whole experimentation procedure feasible for efficient implementation on CUDA^{TM}. Moreover, due to embarrassingly parallel nature of the procedure, the speedup scales linearly to the number of multiprocessors and GPU devices.

4. Meta-optimization (parameters tuning)

Performance of evolutionary computing methods depends strongly on various parameters, such as mutation range, genetic operators application probabilities, learning factors, migration rate, selection pressure etc. Usually, the parameters have great impact on performance and efficacy of the algorithm, and often they are found by manual, time-consuming experimentation. Parameters values can be evaluated according to the performance of the algorithm. Such meta-fitness of an algorithm is a quantity that describes performance of the algorithm with given parameters set. For example, the meta-fitness can be based on the mean fitness value after the end of evolution, over several dozen runs. When the number of parameters increases, the time usage for exploring such parameters space increases exponentially. Moreover, because of stochastic nature of evolutionary algorithms, the algorithm meta-fitness needs to be based on an average over at least 50 executions of the algorithm [41]. Consequently, evaluation of meta-fitness is a very time-consuming and computationally exhaustive operation. For example, if a population consists of 100 individuals evolving for 1000 generations, one evaluation of the evolutionary algorithm meta-fitness requires evaluations of the individuals' fitness. As it is easy to see, an evaluation of meta-fitness takes several orders of magnitude longer than an evaluation of fitness. Therefore, an efficient method is needed to search the space of parameters.

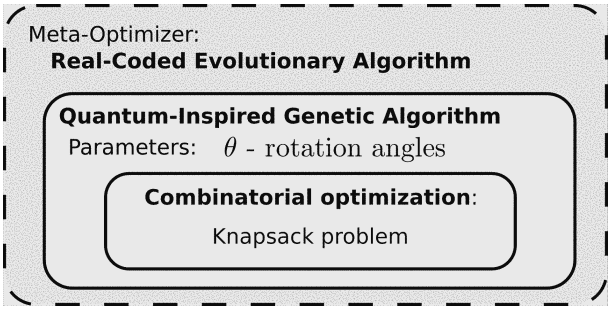

Fig. 5. Approach to meta-optimization taken in this paper

Meta-optimization is a systematic approach to this problem and the general idea behind it has been presented in Fig. 5. The meta-optimization algorithm assesses parameters of the

underlying optimizer according to its meta-fitness measure. In our research, Quantum-Inspired Genetic Algorithm has been implemented entirely as a computational kernel running on GPU, and selected parameters of the algorithm has been tuned in meta-optimization with real-coded evolutionary algorithm.

5. Experimental results

In this section, details of the performed numerical experiments have been provided which allows the reader to verify the presented results. In the experiments, two implementations of QIGA algorithm have been tested on combinatorial optimization (knapsack problem), and their results have been compared to Simple Genetic Algorithm [42] (SGA). For comprehensive details of QIGA algorithm application to the knapsack problem, the reader is referred to [2, 43].

Firstly, QIGA algorithm has been implemented as a typical sequential program in ANSI C running on CPU for comparison. Secondly, implementation in CUDA™architecture has been created. Execution time of the two implementations has been compared. Finally, the efficient GPU-based implementation has been used in a meta-optimization process to tune selected parameters (rotation angles in quantum genes state space) of Quantum-Inspired Genetic Algorithm.

The knapsack consisting of $m = 250$ items has been considered. Similarly to [2], a strongly correlated set of data has been generated (i.e. hard version of the problem where precious items are heavy):

$$\begin{cases} w_i = \text{uniformly random } [1, 10) \\ p_i = w_i + 5 \end{cases} \tag{3}$$

where w_i denotes weight of the i-th item, and p_i denotes profit of the item. The profit $f(x)$ of a binary solution x is evaluated by $f(x) = \sum_{i=1}^{m} p_i x_i$, and this value has been used in the experiments as a fitness value of the individuals in evolutionary algorithms. The knapsack capacity C has been set as follows:

$$C = \frac{1}{2} \sum_{i=1}^{m} w_i. \tag{4}$$

If a binary solution which does not satisfy the constraints is generated (i.e. the knapsack too heavy), the *repair procedure* is involved. The repair procedure is the same for SGA and QIGA algorithm, and it works as follows. If a too heavy knapsack is generated, the items are consequently removed until the constraint is satisfied. If a knapsack is too light (underfilled), the repair procedure consequently adds items to the knapsack, as long as the allowed weight constrain is satisfied. For the pseudocode of the repair procedure, the reader is referred to [2].

In QIGA, the population size was set to 10 quantum individuals, evolving for 500 generations. In SGA, the population size was set to 100 individuals (binary solutions), evolving for 50 generations. Thus, the total number of fitness evaluations was equal in both algorithms. In SGA, single point crossover operator with probability $P_c = 0.65$ and mutation operator with probability $P_m = 0.05$ were used. The selection was based on the roulette wheel method.

In each generation, the i-th qubit value $[\alpha_i \;\; \beta_i]^T$ is updated (update step in Listing 1) with the rotation matrix $U(\theta_i)$ as follows:

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = U(\theta_i) \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}. \tag{5}$$

The rotation angle θ_i is determined as $s(\alpha_i\beta_i)\Delta\theta_i$, where $s(\alpha_i\beta_i)$ is a sign (rotation direction), and $\Delta\theta_i$ is the rotation angle value. $s(\alpha_i\beta_i)$ and $\Delta\theta_i$ are given in lookup tables which have been presented in Table 1 and Table 2, respectively. $f(x)$ denotes fitness of the binary chromosome x, and $f(b)$ denotes fitness of the best individual b ever found during the evolution. The lookup tables and other parameters values have been taken from [2] directly.

Table 1
Lookup table of the rotation angle

x_i	b_i	$f(x) \geq f(b)$	$\Delta\theta_i$	
0	0	False	0	
0	0	True	0	
0	1	False	0	
0	1	True	0.05π	
1	0	False	0.01π	Subject to
1	0	True	0.025π	meta-optimization
1	1	False	0.005π	
1	1	True	0.025π	

The values of $\Delta\theta_i$ in the three first rows in Table 1 are 0. Because the corresponding rotation directions $s(\alpha_i\beta_i)$ are 0 (the first three rows in Table 2), the first three rotation angles do not matter. We will consider the rotation angles in the last five rows in Table 1 as a subject to the meta-optimization process in the final part of the experiment.

Table 2
Lookup table of the rotation direction

x_i	b_i	$f(x) \geq f(b)$	$s(\alpha_i\beta_i)$			
			$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	False	0	0	0	0
0	0	True	0	0	0	0
0	1	False	0	0	0	0
0	1	True	-1	$+1$	± 1	0
1	0	False	-1	$+1$	± 1	0
1	0	True	$+1$	-1	0	± 1
1	1	False	$+1$	-1	0	± 1
1	1	True	$+1$	-1	0	± 1

In Han's implementation[2] (Visual C++ 6.0, Pentium-III 500 MHz), about 0.724 evolutions per second is performed (cf. Table 2 in [2]). In our CPU implementation (ANSI C, Intel Core i7, 2.93 GHz), about 7.324 evolutions per second is performed. In the GPU implementation (nVidia CUDA C, GTX-295 dual-GPU graphic card), evolving independent populations in the grid of size 50x20, about 882.7 evolutions per second is performed. Thus, the speedup gained on GTX-295 is about 120x in comparison to the sequential implementation. Execution time comparison for this configuration has been presented in Fig. 6. Also, speedup gained for different grid sizes (different numbers of independent populations evolutions) has been presented in Fig. 7. As can be seen from the graph, an average speedup gained is over 120x. Irregularities in the graph are due to the number of multiprocessors that are available on GTX 295 card (30 multiprocessors for one GPU core). Slightly better speedups are obtained for the numbers of experiments that are a multiple of available multiprocessors. Along with increase of the number of parallel populations and higher multiprocessors occupancy, such correlation (irregularities) disappears. For more than 8000 independent populations (the grid consisting of 8000 populations blocks), the speedup is stable and over 125x.

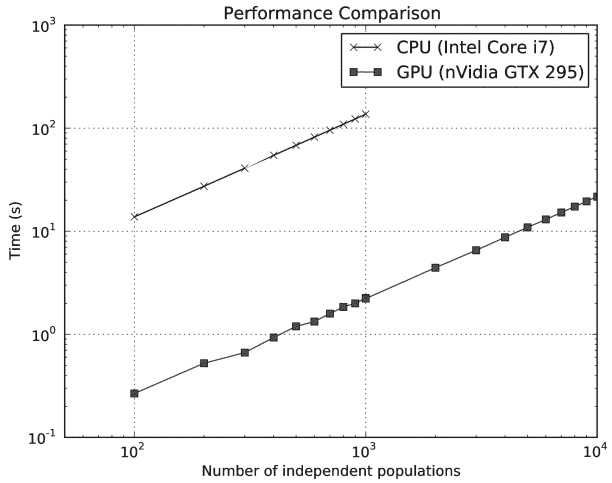

Fig. 6. Execution time comparison (Intel i7 CPU vs GPU GTX 295)

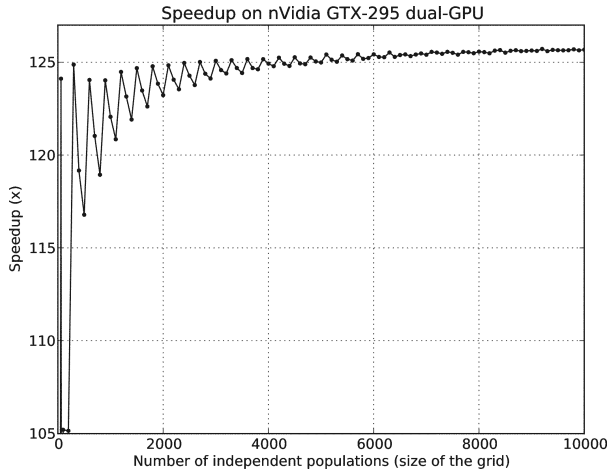

Fig. 7. Speedup on GPU GTX 295 for different grid sizes

Also, an experiment has been conducted with distributed calculations on eight GPU devices (4 x Tesla T10 GPU, GTX 285, dual-GPU GTX 295 and Tesla C2070 GPU). On this configuration, the speedup gained was over 400x (evolution of over 2941 independent populations per second). Therefore, the speedup gained is unarguably significant and superior to a cluster of several latest multi-core CPUs.

Finally, our GPU-based implementation of QIGA algorithm has been used for the meta-optimization process. In this part of the experiment, the search space of a single rotation angle $\Delta\theta_i$ has been set in the interval $[0°, 20°]$ ($[0, 0.349]$ in radians). Let us denote the meta-fitness value $\widetilde{f}$ of the QIGA algorithm as:

$$\widetilde{f}(\Delta\theta) : [0°, 20°]^5 \mapsto \mathbb{R}^+ \tag{6}$$

Two possible definitions of the QIGA algorithm meta-fitness $\widetilde{f}$ have been considered (two meta-optimization criteria):

1. the fitness value after 5000 fitness evaluations (500 generations for populations consisting of 10 quantum individuals), average over 50 evolutions – objective of the meta-optimization was to maximize this criterion
2. the number of fitness evaluations, after which a certain fitness value has been reached, average over 50 evolutions. In the experiment, this level has been set arbitrarily to 1450 – objective of the meta-optimization was to minimize this criterion

In meta-optimization, the QIGA algorithm has been tuned with respect to the first criterion, and then, with respect to the second. The objective of the first stage was to find rotation angles that result in the best final fitness values, and the objective of the second stage was to ensure a good convergence in the beginning of the evolution.

As an overlaid meta-optimizer, real-coded evolutionary algorithm from Python PyEvolve [44] library has been used as presented in Fig. 5. As a crossover operator, single point crossover with probability $P_c = 0.9$ has been used. Also, Gaussian mutation with standard deviation $\sigma = 5°$ has been applied with probability $P_m = 0.066$. Maximum generations number was 50 generations, population size (consisting of the parameters sets $\Delta\theta$) was 10.

On one GTX-295 graphic card, a single run of the meta-optimization process was approximately 70 seconds long. The meta-optimization process has been started 20 times. In Fig. 8, performance comparison (average over 50 runs) of the four algorithms has been presented: Simple Genetic Algorithm, original Quantum-Inspired Genetic Algorithm (parameters from [2]) and two tuned Quantum-Inspired Genetic Algorithm (with respect to the criterion 1) and 2), respectively). In the end of the evolution, tuned QIGA clearly outperforms SGA and original QIGA algorithm.

In Table 3, original rotation angles have been given in radians and degrees, and the corresponding final knapsack profit (average over 50 runs) in original QIGA algorithm. In Table 4, 20 sets of the rotation angles found in meta-optimization have

been provided and corresponding average final knapsack profits. Table 5 provides statistics of the values presented in Table 4.

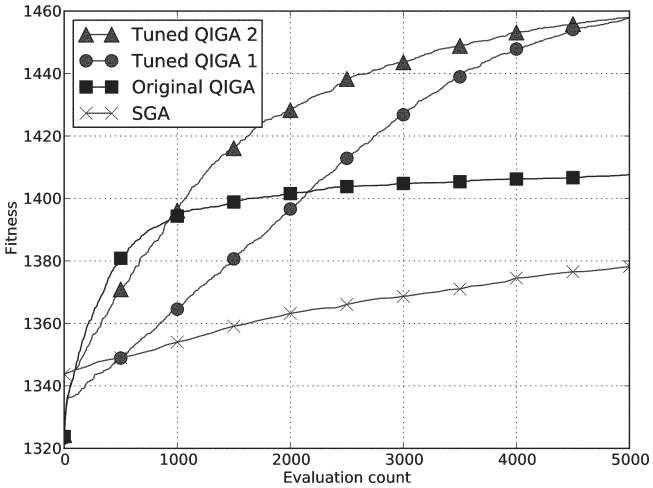

Fig. 8. Performance comparison of the algorithms

Table 3
Original QIGA rotation angles (in radians and degrees, respectively), drawn from [2]

$\Delta\theta$					Final knapsack profit
0.05π	0.01π	0.025π	0.005π	0.025π	
0.157	0.031	0.079	0.016	0.079	1408.25
9°	1.8°	4.5°	0.9°	4.5°	

Table 4
The best sets of rotation angles $\Delta\theta$ for Quantum-Inspired Genetic Algorithm found in the meta-optimization process

$\Delta\theta$					Final knapsack profit
0.000	0.044	0.223	0.254	0.151	1462.65
0.128	0.042	0.262	0.262	0.200	1462.45
0.159	0.043	0.246	0.262	0.251	1462.28
0.056	0.040	0.235	0.262	0.244	1462.04
0.037	0.042	0.012	0.262	0.262	1461.55
0.193	0.049	0.256	0.262	0.181	1461.14
0.248	0.039	0.080	0.262	0.140	1460.18
0.067	0.041	0.247	0.262	0.063	1459.92
0.113	0.038	0.128	0.262	0.205	1459.48
0.000	0.038	0.349	0.334	0.349	1458.86
0.117	0.036	0.349	0.349	0.000	1458.79
0.022	0.054	0.076	0.262	0.132	1458.46
0.125	0.048	0.060	0.212	0.023	1458.40
0.063	0.038	0.239	0.326	0.320	1458.14
0.157	0.034	0.256	0.349	0.081	1456.82
0.065	0.051	0.035	0.242	0.135	1456.53
0.146	0.053	0.246	0.202	0.154	1456.44
0.037	0.034	0.141	0.262	0.000	1456.31
0.281	0.032	0.206	0.348	0.137	1456.09
0.156	0.055	0.232	0.231	0.178	1455.35

Table 5
Statistics of the results from Table 4

$\Delta\theta$					
0.108	0.042	0.193	0.273	0.160	Average
0.115	0.041	0.233	0.262	0.152	Median
0.075	0.006	0.096	0.042	0.094	Std dev.

In Table 4 and 5, it is easy to see which rotation angles are significant and which are not for the performance of the algorithm. For example, standard deviation is smallest in the second column (0.006) in Table 5, which means that in each run of the meta-optimization, approximately the same value of the second rotation angle has been found (circa 0.042, original value was 0.031). Standard deviations in the first, third and fifth columns are at least ten times bigger. Therefore, we conclude that these rotation angles are less significant for the performance of the algorithm. The fourth rotation angle has medium significance (standard deviation 0.042), and its average value is 0.273, which is much bigger than the original value of this parameter (0.016, fourth column in Table 3).

Eventually, SGA, original QIGA and the tuned QIGA algorithm with the best parameters have been run 30000 times, and the results distributions have been presented as histograms in Fig. 9. In the histograms, periodical peaks are visible. The visible pattern is related directly to the discrete nature of the knapsack problem [43]. It is beyond the scope of this paper, but it needs some explanation. The peak is visible every 5 values on the horizontal axis. The step length is a result of the data generation procedure in Eq. (3), and the peaks are due to composition of the repair procedure with the evolution process. According to central limit theorem [45], distribution of the sum or the average of a large number of random variables is approximately normally distributed. Thus, if some radical simplification had been made (i.e. no repair procedure), the histogram would have been similar to normal distribution.

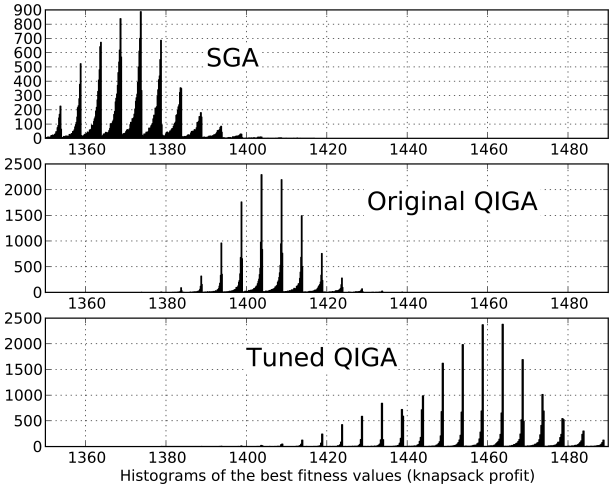

Fig. 9. Results distribution comparison on histograms

6. Conclusions

In this article, meta-optimization (parameters tuning) of Quantum-Inspired Genetic Algorithm in GPU-based massive-

ly parallel environment (CUDA^{TM}technology) has been performed. The proposed approach to parallelization is twofold: In a block of threads, each thread transforms a separate individual or different gene; In each block, evolution of a separate population with same or different parameters is conducted. As a result, on eight GPU devices, over 400x speedup has been gained. The speedup gained allowed efficient meta-optimization of Quantum-Inspired Genetic Algorithm for a combinatorial optimization problem. Two criteria for the meta-optimization of the rotation angles in quantum genes state space have been considered. The tuned QIGA algorithm performs much better than the original algorithm.

Modern massively parallel computing environments provide resources for successful application of contemporary population-based heuristics for meta-optimization of state-of-the-art evolutionary algorithms. The presented approach to parallelization of the experimentation procedure can be applied to a broad class of metaheuristics. Also, it can be implemented in similar and competitive to CUDA technologies, e.g. OpenCL [46, 47], BrookGPU [48].

Acknowledgements. Robert Nowotniak, a co-author of the present paper, is a scholarship holder of project entitled "Innovative education" supported by European Social Fund. This research was supported in part by PL-Grid Infrastructure.

REFERENCES

[1] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms", *Proc. IEEE Evolutionary Computation* 1, 61–66 (1996).

[2] K.H. Han and J.H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem", *Proc. Congress on Evolutionary Computation* 1, 1354–1360 (2000).

[3] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, 1996.

[4] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2000.

[5] P. Jantos, D. Grzechca, and J. Rutkowski, "Evolutionary algorithms for global parametric fault diagnosis in analogue integrated circuits", *Bull. Pol. Ac.: Tech.* 60 (1), 133–142 (2012).

[6] A. Slowik, "Application of evolutionary algorithm to design minimal phase digital filters with non–standard amplitude characteristics and finite bit word length", *Bull. Pol. Ac.: Tech.* 59 (2), 125–135 (2011).

[7] L. Chomatek and M. Rudnicki, "Application of genetically evolved neural networks to dynamic terrain generation", *Bull. Pol. Ac.: Tech.* 59 (1), 3–8 (2011).

[8] Ł. Jopek, R. Nowotniak, M. Postolski, L. Babout, and M. Janaszewski, "Application of Quantum Genetic Algorithms in Feature Selection Problem", *Scientific Bull. Ac. Sci. and Technology, Automatics* 13 (3), 1219–1231 (2009).

[9] H. Talbi, M. Batouche, and A. Draa, "A quantum-inspired genetic algorithm for multi-source affine image registration", in: *Image Analysis and Recognition*, pp. 147–154, Springer, Berlin, 2004.

[10] H. Talbi, M. Batouche, and A. Draa, "A quantum-inspired evolutionary algorithm for multiobjective image segmentation", *Int. J. Mathematical, Physical and Engineering Sciences* 1, 109–114 (2007).

[11] L. Wang, H. Wu, F. Tang, and D.Z. Zheng, "A hybrid quantum-inspired genetic algorithm for flow shop scheduling", in: *Advances in Intelligent Computing*, pp. 636–644, Springer, Berlin, 2005.

[12] B.B. Li and L. Wang, "A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling", *IEEE Trans. Systems, Man, and Cybernetics, Cybernetics* B 37, 576–591 (2007).

[13] Y.W. Jeong, J.B. Park, J.R. Shin, and K.Y. Lee, "A thermal unit commitment approach using an improved quantum evolutionary algorithm", *Electric Power Components and Systems* 37, 770–786 (2009).

[14] T. Lau, C. Chung, K. Wong, T. Chung, and S. Ho, "Quantum-inspired evolutionary algorithm approach for unit commitment", *IEEE Trans. Power Systems* 24, 1503–1512 (2009).

[15] L. Su-Hua, W. Yao-Wu, P. Lei, and X. Xin-Yin, "Application of quantum-inspired evolutionary algorithm in reactive power optimization", *Relay* 33, 30–35 (2005).

[16] J.G. Vlachogiannis and K.Y. Lee, "Quantum-inspired evolutionary algorithm for real and reactive power dispatch", *IEEE Trans. Power Systems* 23, 1627–1636 (2003).

[17] S. Jeżewski, M. Łaski, and R. Nowotniak, "Comparison of algorithms for simultaneous localization and mapping problem for mobile robot", *Sci. Bull. Ac. Sci. and Technology, Automatics* 14, 439–452 (2010).

[18] G. Zhang, "Quantum-inspired evolutionary algorithms: a survey and empirical study", *J. Heuristics* 17, 1–49 (2010).

[19] J.J. Grefenstette, "Optimization of control parameters for genetic algorithms", *IEEE Trans. Systems, Man and Cybernetics* 16, 122–128 (1986).

[20] R. Nowotniak and J. Kucharski, "Meta-optimization of quantum-inspired evolutionary algorithm", *Proc. XVII Int. Conf. on Information Technology Systems* 1, CD-ROM (2011).

[21] M.E.H. Pedersen, *Tuning & Simplifying Heuristical Optimization*, University of Southampton, Southampton, 2010.

[22] NVidia Corporation, *Compute Unified Device Architecture Programming Guide*, NVIDIA, Santa Clara, 2007.

[23] J. Owens, "GPU architecture overview", *ACM SIGGRAPH* 1, 5–9 (2007).

[24] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell, "A survey of general-purpose computation on graphics hardware", *Computer Graphics Forum* 26, 80–113 (2007).

[25] R. Nowotniak and J. Kucharski, "Building blocks propagation in quantum-inspired genetic algorithm", *Sci. Bull. Ac. Sci. and Technology, Automatics* 14, 795–810 (2010).

[26] A.R. Khorsand and T.M.R Akbarzadeh, "Quantum gate optimization in a meta-level genetic quantum algorithm", *IEEE Int. Conf. Systems, Man and Cybernetics* 4, 3055–3062 (2005).

[27] R. Fernando, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Addison-Wesley Professional, London, 2004.

[28] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA", *ACM Queue* 6, 40–53 (2008).

[29] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms", *ACM SIGGRAPH 2005 Courses* 1, 908–916 (2005).

[30] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems", *Parallel Computing* 36, 232–240 (2010).

[31] O. Fialka and M. Cadik, "FFT and convolution performance in image filtering on GPU", *10th Int. Conf. on Information Visualization* 1, 609–614 (2006).

[32] M.J. Harris, "Fast fluid dynamics simulation on the GPU", *GPU Gems* 1, 637–665 (2004).

[33] T. Preis, P. Virnau, W. Paul, and J.J. Schneider, "GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model", *J. Computational Physics* 228, 4468–4477 (2009).

[34] M. Lee, J. Jeon, J. Bae, and H.S. Jang, "Parallel implementation of a financial application on a GPU", *Proc. 2nd Int. Conf. Interaction Sciences: Information Technology, Culture and Human* 1, 1136–1141 (2009).

[35] T. Preis, P. Virnau, W. Paul, and J.J. Schneider, "Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets", *New J. Physics* 11, 93–124 (2009).

[36] K. Moreland and E. Angel, "The FFT on a GPU", *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graphics hardware* 1, 112–119 (2003).

[37] W. Banzhaf and S. Harding, "Accelerating evolutionary computation with graphics processing units", *Proc. 11th Annual Conf. Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* 1, 3237–3286 (2009).

[38] F. Krüger, O. Maitre, S. Jiménez, L. Baumes, and P. Collet, "Speedups between 70x and 120x for a generic local search (memetic) algorithm on a single GPGPU chip", *Applications of Evolutionary Computation* 1, 501–511 (2010).

[39] K.L. Fok, T.T. Wong, and M.L. Wong, "Evolutionary computing on consumer graphics hardware", *IEEE Intelligent Systems* 22, 69–78 (2007).

[40] M.L. Wong, "Parallel multi-objective evolutionary algorithms on graphics processing units", *Proc. 11th Annual Conf. Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* 1, 2515–2522 (2009).

[41] S. Luke, *Essentials of metaheuristics* lulu.com, 2009.

[42] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, London, 1989.

[43] K.H. Han and J.H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization", *IEEE Trans. Evolutionary Computation* 6, 580–593 (2002).

[44] C.S. Perone, "PyEvolve: a Python open–source framework for genetic algorithms", *ACM SIGEVOlution* 4, 12–20 (2009).

[45] R. Durrett, *Probability: Theory and Examples*, International Thomson Publishing Company, New York, 1996.

[46] J.E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems", *Computing in Science and Engineering* 12, 66–73 (2010).

[47] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, and S. Miki, *The OpenCL Programming Book*, Fixstars Corporation, London, 2009.

[48] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware", *ACM Trans. on Graphics* 23, 777–786 (2004).