Robert NOWOTNIAK[1], Jacek KUCHARSKI[1]

Computer Engineering Department, Technical University of Łódź (1)

GPU-based Tuning of Quantum-Inspired Genetic Algorithm for a Combinatorial Optimization Problem

***Abstract**. This paper concerns efficient parameters tuning (meta-optimization) of a state-of-the-art metaheuristic, Quantum-Inspired Genetic Algorithm, in a GPU-based massively parallel computing environment (NVidia CUDA™ technology). A novel approach to parallel implementation of the algorithm has been presented. The computations have been distributed to eight GPU devices, and over 400x speedup has been gained. This approach allows efficient meta-optimization of the algorithm which has been demonstrated on combinatorial optimization (knapsack problem).*

Keywords: quantum-inspired genetic algorithm, evolutionary computing, meta-optimization, parallel algorithms, GPGPU

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. The algorithm have demonstrated its efficacy for solving complex optimization problems. Recent years have witnessed successful applications of Quantum-Inspired Genetic Algorithms in variety of optimization problems, including image processing[5,6,7], flow shop scheduling[8,9], thermal unit commitment[10,11], power system optimization[12,13], localization of mobile robots[14] and many others. For a current and comprehensive survey of Quantum-Inspired Genetic Algorithms, the reader is referred to [15].

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[16,17,18] process requires substantial computing power. Fortunately, the modern massively parallel computing environments (NVidia CUDA™ technology[19,20,21]) provide sufficient resources that allows tuning the algorithm with population-based heuristics automatically.

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

1) 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 equation:

(1) $$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where: $\alpha, \beta \in \mathbf{C}$ - probability amplitudes, $|0\rangle = [1 \quad 0]^T$, $|1\rangle = [0 \quad 1]^T$ and $|\alpha|^2 + |\beta^2| = 1$.

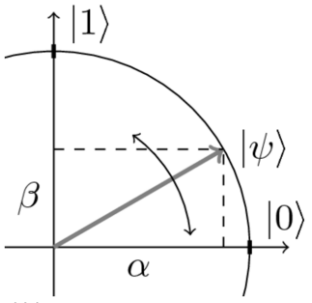

Fig.1. Illustration of binary quantum gene state

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 Figure 1. Along with increase of 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:

(2) $$q = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{bmatrix}$$

where each column corresponds to binary quantum gene $|\Psi\rangle_1, \ldots, |\Psi\rangle_m$. Consequently, the state of the whole quantum population $Q=\{q_1,q_2,\ldots,q_N\}$ can be simply illustrated by a matrix of vectors[22], which has been presented in Figure 2. Each row in the figure corresponds to binary quantum chromosome, as in equation (2).

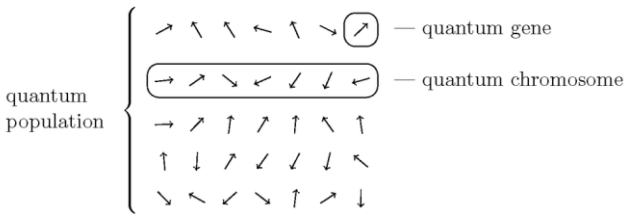

Fig. 2. Illustration of binary quantum population. Each arrow represent a state of a quantum gene.

The full pseudocode of QIGA algorithm has been presented in Figure 3.

```
procedure Quantum-Inspired Genetic Algorithm
begin
    t←0
    initialize Q(0)
    make P(0) by observing Q(0)
    evaluate P(0)
    store the best solution among P(0)
    while not termination-criterion do
        t←t+1
        make P(t) by observing Q(t-1) population
        evaluate P(t)
        update Q(t) using quantum gates U(θt)
        store the best solution among P(t)
    end while
end
```

Fig.3. Pseudo-code of Quantum-Inspired Genetic Algorithm

In the beginning of the algorithm, the genes of all individuals in the quantum population *Q(0)* are initialized with linear superposition of states $\left(\frac{\sqrt{2}}{2}\left|0\right\rangle + \frac{\sqrt{2}}{2}\left|1\right\rangle\right)$. 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 *quantum rotation gates,* which rotate state vectors in the quantum gene state space. Proper rotation 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[17,23].

2) 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[21,24]. One of the leading General-Purpose Computing on Graphics Processing Units (GPGPU) nowadays is NVIDIA Compute Unified Device Architecture[19,25] (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[26,27], signals processing[28], scientific simulations[29,30], finance[31,32] and others[33]. 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. In CUDA™, processing threads are grouped in blocks, and blocks constitutes a two-dimensional grid. 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.

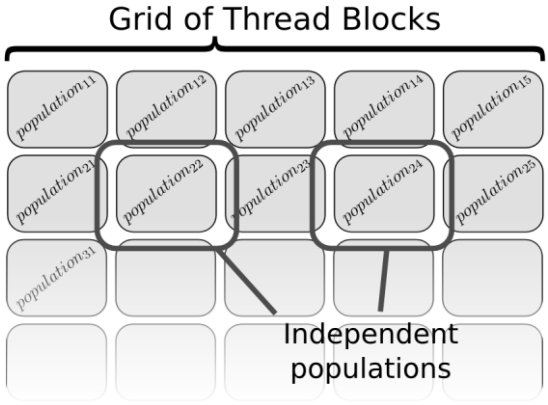

Fig. 4. Proposed approach to parallelization of the experimentation procedure

In several recent papers (e.g. [34,35,36,37]), 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 Figure 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™. Moreover, due to embarrassingly parallel nature of the procedure, the speedup scales linearly to the number of multiprocessors and GPU devices.

3) 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 [38]. 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 $50 \cdot 100 \cdot 1000 = 5000000$ 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.

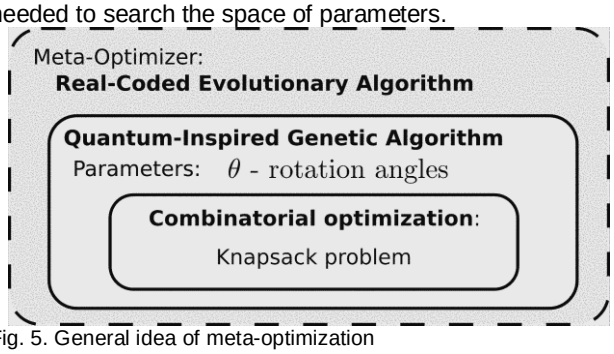

Fig. 5. General idea of meta-optimization

Meta-optimization is a systematic approach to this problem and the general idea behind it has been presented in Figure 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.

4) 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[39] (SGA). For comprehensive details of QIGA algorithm application to the knapsack problem, the reader is referred to [2,40].

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):

$$(3) \qquad \begin{cases} w_i = uniformlyr \quad andom\ [1,10\,) \\ p_i = w_i + 5 \end{cases}$$

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 $f(x) = \sum_{i=1}^{m} p_i x_i$. The knapsack capacity has been set as follows:

$$(4) \qquad C = \frac{1}{2} \sum_{i=1}^{m} w_i$$

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 (α_i, β_i) is updated (update step in Fig. 3) with the rotation matrix as follows:

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

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 has been presented in Table 1 and Table 2. 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	Subject to meta-optimization
1	0	False	0.01π	
1	0	True	0.025π	
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. Let us 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 500MHz), about 0.724 evolutions per second is performed (cf. Table 2 in [2]). In our CPU implementation (ANSI C, Intel Core i7, 2.93GHz), 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 Figure 6. 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 (500 generations of 10 quantum individuals in approximately 0.00034 seconds).

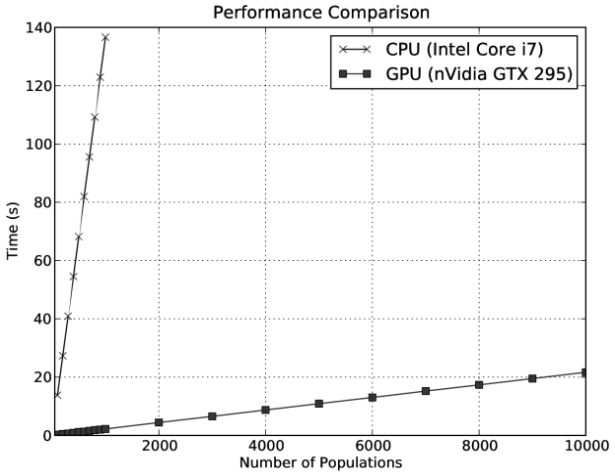

Fig. 6. Execution time comparison (Intel i7 CPU vs GPU GTX 295)

Finally, our GPU-based implementation of QIGA algorithm has been used for the meta-optimization process. As an overlaid meta-optimizer, real-coded evolutionary algorithm, implemented in Python and PyEvolve[41] library, has been used. In meta-optimization, the rotation angles $\Delta\theta_i$ in the interval $[0^\circ, 20^\circ]$ ($[0, 0.349]$ in radians) has been considered:

(5) $$\tilde{f} : [0^\circ, 20^\circ]^5 \mapsto \Re^+$$

The meta-fitness $\tilde{f}$ value of the algorithm has been defined as the mean fitness value after 5000 fitness evaluations (500 generations for populations consisting of 10 individuals), over 50 evolutions. As a crossover operator, single point crossover with probability P_c=0.9 has been used. Also, Gaussian mutation with standard deviation σ=5° has been applied with probability P_m=0.066. Maximum generations number was 50 generations, population size was 10. On GTX-295 graphic card, a single run of the meta-optimization process was approximately 7 seconds long. The plot of the evolution performing meta-optimization has been presented in Figure 7. Meta-optimization has been started 50 times, and box-and-whisker plot in Figure 7 presents results distribution every 6 generations.

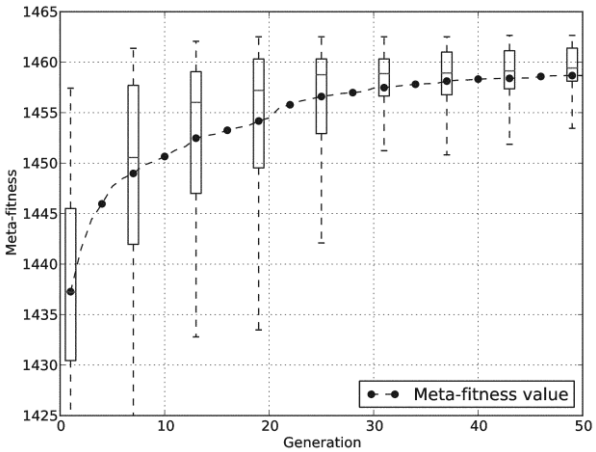

Fig. 7. Tuning parameters of QIGA in a meta-optimization process

In Figure 8, performance comparison (average over 50 executions) of the three algorithms has been presented: Simple Genetic Algorithm, original Quantum-Inspired Genetic Algorithm (parameters from [2]) and tuned Quantum-Inspired Genetic Algorithm. In the end of the evolution, tuned QIGA clearly outperforms the two other algorithms. However, it has a slower convergence rate in the beginning, because this criterion has not been taken into account in meta-fitness $\tilde{f}$ value calculation.

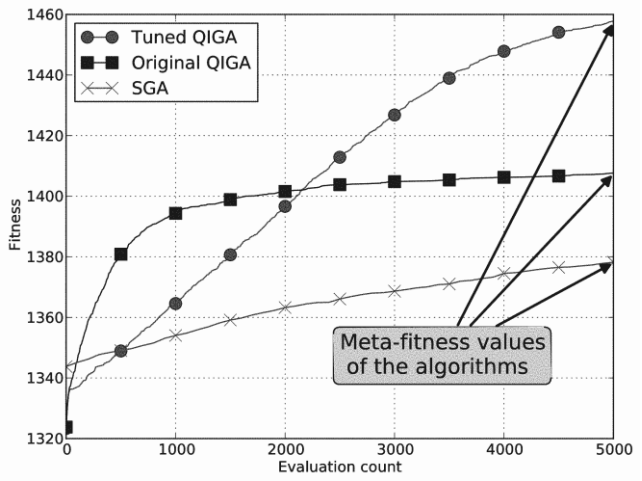

Fig. 8. Performance comparison of the algorithms

In Table 3, the five best rotation angles values have been given and their corresponding meta-fitness values. In the last row, original parameters from [2] are presented. In Table 3, it is easy to see that in each row in the first and fifth column, the values are completely different. Therefore, these rotation angles are not significant for the performance of the algorithm. On the contrary, in each row of the second, third and fourth column, the values are similar (approximately 0.036, 0.279 and 0.341, respectively), and these rotation angles are important for the performance of the algorithm. The values in the second column are approximately equal to the value 0.031 given originally in [2], whereas the rotation angles in the third and fourth column are considerable greater than the original values.

Table 3. The best rotation angles $\Delta\theta$ for Quantum-Inspired Genetic Algorithm found by meta-optimization process. Original values are given in the last row.

$\Delta\theta$					meta-fitness
0.000	0.038	0.349	0.334	0.349	1458.86
0.117	0.036	0.349	0.349	0.000	1458.79
0.063	0.038	0.239	0.326	0.320	1458.14
0.157	0.034	0.256	0.349	0.081	1456.82
0.281	0.032	0.206	0.348	0.137	1456.09
0.157	0.031	0.079	0.016	0.079	1408.25

Eventually, each algorithm has been run 30000 times, and the results distributions have been presented as histograms in Figure 9. In the histograms, periodical peaks are clearly visible. The visible pattern is related directly to the discrete nature of the knapsack problem [40]. 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 Equation (3), and the peaks are due to composition of the repair procedure with the evolution process. According to central limit theorem[42], 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.

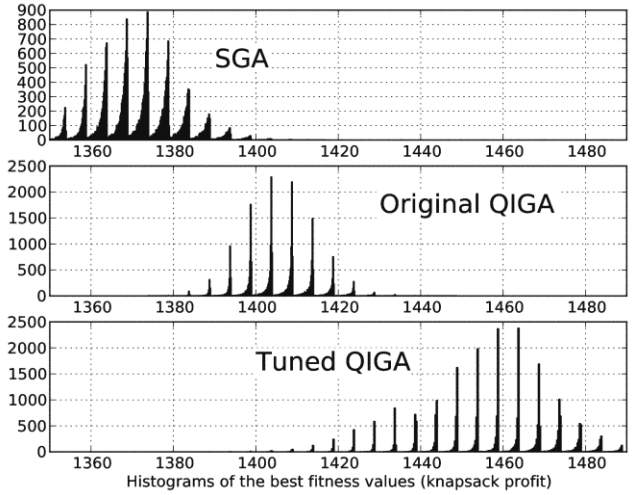

Fig. 9. Results distribution comparison on histograms

5) Conclusions

In this article, meta-optimization (parameters tuning) of Quantum-Inspired Genetic Algorithm in GPU-based massively parallel environment (CUDA™ technology) has been considered. 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. 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 [43,44], BrookGPU[45].

References

[1] Narayanan A., Moore M., Quantum-inspired genetic algorithms, *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on,* (1996), 61-66

[2] Han K.H., Kim J.H., Genetic quantum algorithm and its application to combinatorial optimization problem, *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, (2000), 1354-1360

[3] Michalewicz Z., Genetic algorithms data structures, *Springer* (1996)

[4] Nielsen M., Chuang I., Quantum computation and quantum information, *Cambridge University Press* (2000)

[5] Jopek Ł., Nowotniak R., Postolski M., Babout L., Janaszewski M., Application of Quantum Genetic Algorithms in Feature Selection Problem, *Scientific Bulletin of Academy of Science and Technology, Automatics,* (2009)

[6] Talbi H., Batouche M., Draa A., A quantum-inspired genetic algorithm for multi-source affine image registration, *Image Analysis and Recognition,* Springer, (2004), 147-154

[7] Talbi H., Batouche M., Draao A., A Quantum-Inspired Evolutionary Algorithm for Multiobjective Image Segmentation, *International Journal of Mathematical, Physical and Engineering Sciences,* No. 1 (2007), 109-114

[8] Wang L., Wu H., Tang F., Zheng D.Z., A hybrid quantum-inspired genetic algorithm for flow shop scheduling, *Advances in Intelligent Computing,* Springer, (2005), 636-644

[9] Li B.B., Wang L., A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling, *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* IEEE, No. 37 (2007), 576-591

[10] Jeong Y.W., Park J.B., Shin J.R., Lee K.Y., A thermal unit commitment approach using an improved quantum evolutionary algorithm, *Electric Power Components and Systems,* Taylor & Francis, No. 37 (2009), 770-786

[11] Lau T., Chung C., Wong K., Chung T., Ho S., Quantum-inspired evolutionary algorithm approach for unit commitment, *Power Systems, IEEE Transactions on,* IEEE, No. 24 (2009), 1503-1512

[12] Su-hua L., Yao-wu W., Lei P., Xin-yin X., Application of quantum-inspired evolutionary algorithm in reactive power optimization, *Relay,* No. 33 (2005), 30-35

[13] Vlachogiannis J.G., Lee K.Y., Quantum-inspired evolutionary algorithm for real and reactive power dispatch, *Power Systems, IEEE Transactions on,* IEEE, No. 23 (2008), 1627-1636

[14] Jeżewski S., Łaski M., Nowotniak R., Comparison of Algorithms for Simultaneous Localization and Mapping Problem for Mobile Robot, *Scientific Bulletin of Academy of Science and Technology, Automatics*, (2010)

[15] Zhang G., Quantum-inspired evolutionary algorithms: a survey and empirical study, *Journal of Heuristics,* Springer, (2010), 1-49

[16] Grefenstette J.J., Optimization of control parameters for genetic algorithms, *Systems, Man and Cybernetics, IEEE Transactions on,* IEEE, No. 16 (1986), 122-128

[17] Nowotniak R., Kucharski J., Meta-optimization of Quantum-Inspired Evolutionary Algorithm, *Proceedings of the XVII International Conference on Information Technology Systems, (2010)*

[18] Pedersen M.E.H., Tuning & Simplifying Heuristical Optimization, *University of Southampton, School of Engineering Sciences*, (2010)

[19] NVidia Corporation, Compute Unified Device Architecture Programming Guide, *NVIDIA: Santa Clara, CA,* (2007)

[20] Owens J., GPU architecture overview, *ACM SIGGRAPH*, (2007), 05-09

[21] Owens J.D., Luebke D., Govindaraju N., Harris M., Krüger J., Lefohn A.E., Purcell T.J., A Survey of General-Purpose Computation on Graphics Hardware, *Computer graphics forum*, No. 26 (2007), 80-113

[22] Nowotniak R., Kucharski J., Building Blocks Propagation in Quantum-Inspired Genetic Algorithm, *Scientific Bulletin of Academy of Science and Technology, Automatics*, (2010)

[23] Khorsand A.R., Akbarzadeh T M.R., Quantum gate optimization in a meta-level genetic quantum algorithm, *Systems, Man and Cybernetics*, 2005 IEEE International Conference on, No. 4 (2005), 3055-3062

[24] Fernando R., GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics, (2004)

[25] Nickolls J., Buck I., Garland M., Skadron K., Scalable parallel programming with CUDA, *Queue,* ACM, No. 6 (2008), 40-53

[26] Krüger J., Westermann R., Linear algebra operators for GPU implementation of numerical algorithms, *ACM SIGGRAPH 2005 Courses*, (2005)

[27] Tomov S., Dongarra J., Baboulin M., Towards dense linear algebra for hybrid GPU accelerated manycore systems, *Parallel Computing,* Elsevier, No. 36 (2010), 232-240

[28] Fialka O., Cadik M., FFT and convolution performance in image filtering on GPU, *Information Visualization, 2006. IV 2006. Tenth International Conference on*, (2006), 609-614

[29] Harris M.J., Fast fluid dynamics simulation on the GPU, GPU Gems, Addison Wesley (2004)

[30] Preis T., Virnau P., Paul W., Schneider J.J., GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model, *Journal of Computational Physics,* Elsevier, No. 228 (2009), 4468-4477

[31] Lee M., Jeon J., Bae J., Jang H.S., Parallel implementation of a financial application on a GPU, *Proceedings of the 2nd International Conference on Interaction*

Sciences: Information Technology, Culture and Human, (2009), 1136-1141
[32] Preis T., Virnau P., Paul W., Schneider J.J., Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets, *New Journal of Physics,* IOP Publishing, No. 11 (2009)
[33] Moreland K., Angel E., The FFT on a GPU, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, (2003), 112-119
[34] Banzhaf W., Harding S., Accelerating evolutionary computation with graphics processing units, *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, (2009), 3237-3286
[35] Krüger F., Maitre O., Jiménez S., Baumes L., Collet P., Speedups between× 70 and× 120 for a generic local search (memetic) algorithm on a single GPGPU chip, *Applications of Evolutionary Computation,* Springer, (2010), 501-511
[36] Fok K.L., Wong T.T., Wong M.L., Evolutionary computing on consumer graphics hardware, *Intelligent Systems, IEEE,* IEEE, No. 22 (2007), 69-78
[37] Wong M.L., Parallel multi-objective evolutionary algorithms on graphics processing units, *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, (2009), 2515-2522
[38] Luke S., Essentials of metaheuristics, *Available at http://cs.Gmu.Edu/~sean/book/metaheuristics*, (2009)
[39] Goldberg D.E., Genetic algorithms in search, optimization, and machine learning, Addison-Wesley Professional, (1989)
[40] Han K.H., Kim J.H., Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *Evolutionary Computation, IEEE Transactions on*, No. 6 (2002), 580-593
[41] Perone C.S., PyEvolve: a Python open-source framework for genetic algorithms, *ACM SIGEVOlution*, No. 4 (2009)
[42] Durrett R., Probability: theory and examples, *International Thomson Publishing Company*, (1996)
[43] Stone J.E., Gohara D., Shi G., OpenCL: A parallel programming standard for heterogeneous computing systems, *Computing in Science and Engineering,* No. 12 (2010), 66–73
[44] Tsuchiyama R., Nakamura T., Iizuka T., Asahara A., Miki S., The OpenCL Programming Book, Fixstars Corporation, (2009)
[45] Buck I., Foley T., Horn D., Sugerman J., Fatahalian K., Houston M., Hanrahan P., Brook for GPUs: stream computing on graphics hardware, *ACM Transactions on Graphics (TOG)*, No. 23 (2004), 777-786

***Authors**: Robert Nowotniak, MSc (PhD student), Computer Engineering Department, Technical University of Łódź, 90-924 Łódź, Stefanowskiego 18/22, Poland. e-mail: rnowotniak@kis.p.lodz.pl. Robert Nowotniak is a scholarship holder of project entitled "Innovative education ..." supported by European Social Fund.*
Jacek Kucharski, MSc, PhD, DSc, Computer Engineering Department, Technical University of Łódź, 90-924 Łódź, Stefanowskiego 18/22, Poland. e-mail: jkuchars@kis.p.lodz.pl