Marcin POŁOMSKI

Silesian University of Technology, Faculty of Electrical Engineering, Institute of Electrical Engineering and Computer Science

doi:10.15199/48.2017.01.75

Improving performance of non-interior point based optimal power flow algorithm computations

Abstract. *This paper presents the non-interior point method (NIP) based optimal power flow (OPF) algorithm parallelization and performance improvement experiments. The aim is to investigate the impact of parallelization techniques on overall OPF computations speedup. Presented approach takes advantage of the structure of algorithm and exploits it with the usage of multithreading to gain computation speedup. Obtained results give insight into the impact of multithreading techniques and algorithm initialization techniques.*

Streszczenie. *W artykule zaprezentowano wyniki eksperymentów prowadzących do redukcji czasu realizacji obliczeń algorytmu metody non-interior point (NIP) w zastosowaniu do zadania optymalizacji rozpływu mocy (OPF). Celem pracy było zbadanie wpływu zastosowania technik zrównoleglenia obliczeń na czas realizacji zadania OPF. W zaprezentowanym podejściu brano pod uwagę strukturę algorytmu oraz wykorzystano implementację wielowątkową. Uzyskane wyniki pokazują wpływ wielowątkowej implementacji oraz zastosowanych technik inicjalizacji algorytmu na czas obliczeń. (**Poprawa wydajności algorytmu metody non-interior point w zastosowaniu do zadania optymalizacji rozpływu mocy**).*

Keywords: non-interior point method, optimal power flow, parallel computing.
Słowa kluczowe: metoda non-interior point, optymalizacja rozpływu mocy, obliczenia równoległe.

Introduction

Optimal power flow (OPF), in general, is related to power system operational and planning optimization methods [1]. The main idea behind OPF is to determine the optimal values for control variables while respecting various constraints. Over the years, a number of methods have been applied to solve the OPF problem [2], using various optimization techniques, but their effectiveness and performance is highly dependent on the size of a power system being optimized [3]. The development of the OPF recently has tracked significant progress both in numerical optimization techniques and computer techniques application. In recent years, application of interior point (IP) and non-interior point (NIP) methods to solve OPF problem has been paid great attention [4], [5]. This is due to the fact that IP method based algorithms are among the fastest algorithms, well suited to solve large-scale nonlinear optimization problems [5].

And yet, it's still a great striving to improve those methods, especially in comparison with the current trends in the development of parallel programming techniques. The vast availability of multi-core CPUs, and multi-threaded programming libraries, inspires the search for solutions affecting not only the computational performance, but also application responsiveness.

Optimal power flow formulation

In general, the OPF problem may be defined mathematically as the nonlinear programming problem (NLP), with equality and inequality constraints, which is written as follows:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}), \\ &\mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ &\mathbf{g}(\mathbf{x}) \le \mathbf{0}, \end{aligned} \tag{1}$$

where: $\mathbf{x}$ – the decision variables, $f(\mathbf{x})$ – objective function, $\mathbf{h}(\mathbf{x})$ – vector of equality constraints, $\mathbf{g}(\mathbf{x})$ – vector of inequality constraints.

In the OPF formulation, the decision variables are represented by voltage magnitudes and angles, active and reactive power generation values. The objective function $f(\mathbf{x})$ may represent total cost losses, total MW generation, transmission losses, etc. The equality constraints $\mathbf{h}(\mathbf{x})$ represents the power flow equations and $\mathbf{g}(\mathbf{x})$ includes functions corresponding to the inequality constraints and variables limits, such as bus voltage limits, active and reactive power generation limits, lines power flow limits.

Non-interior point method

One way to solve the problem (1), with respect to the interior-point method formulation [6], is to transform the inequality constraints into equality constraints by incorporating a barrier function and the non-negative slack variables vector $\mathbf{z}$,

$$\begin{aligned} &\min_{\mathbf{x}} \left(f(\mathbf{x}) - \mu_k \sum_{i=1}^{n_g} \ln(z_i) \right), \\ &\mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ &\mathbf{g}(\mathbf{x}) + \mathbf{z} = \mathbf{0}, \\ &\mathbf{z} \ge \mathbf{0}, \end{aligned} \tag{2}$$

where: n_g – number of inequality constraints, μ_k – barrier parameter, $\mathbf{z}$ – vector of slack variables.

According to logarithmic barrier function, as the parameter μ_k approaches to zero (in *k*-th iteration), the solution of problem (2) approaches to the solution of problem (1). The Karush–Kuhn–Tucker (KKT) optimality conditions for the (2) can be written as:

$$\begin{bmatrix} \mathbf{Z}\boldsymbol{\pi} - \mu_k \mathbf{e} \\ \mathbf{g}(\mathbf{x}) + \mathbf{z} \\ \nabla_{\mathbf{x}} f(\mathbf{x}) + (\nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x}))^{\mathrm{T}} \boldsymbol{\lambda} + (\nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}))^{\mathrm{T}} \boldsymbol{\pi} \\ \mathbf{h}(\mathbf{x}) \end{bmatrix} = \mathbf{0}, \tag{3}$$

where: $\nabla_{\mathbf{x}}$ – partial derivative with respect to $\mathbf{x}$, $\mathbf{e} = [1, 1, ...1]^{\mathrm{T}}$, $\mathbf{Z} = diag[z_i;\ i = 1,..., n_g]$, $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ – the vectors of Lagrange multipliers, corresponding to equality and inequality constraints.

The system of nonlinear equations (3) forms the basis of the calculation process of the interior point method. The equations (3) are solved using a variant of Newton's method. The proper way of determining the value of barrier parameter μ_k, and controlling the Newton's method step size in subsequent iterations, are amongst the most important elements affecting convergence of IP method.

The non-interior point method adopts the technique used in the class of complementarity problems [4]. It involves the introduction of so-called smoothing function (with additional parameter μ):

(4) $\varphi_\mu(z,\pi)=0 \;\Leftrightarrow\; \pi>0,\; z>0,\; \pi z=\mu$ for μ > 0,

in the place of complementarity conditions, in the system of equations resulting from KKT conditions. It leads to the following system of nonlinear equations:

$$\text{(5)}\quad \mathbf{\Psi}(\mathbf{y})=\begin{bmatrix}\mathbf{r}_\mathbf{z}\\ \mathbf{r}_\pi\\ \mathbf{r}_\mathbf{x}\\ \mathbf{r}_\lambda\\ \mathbf{r}_\mu\end{bmatrix}=\begin{bmatrix}\mathbf{\Phi}_\mu(\mathbf{z},\boldsymbol{\pi})\\ \mathbf{g}(\mathbf{x})+\mathbf{z}\\ \nabla_\mathbf{x} f(\mathbf{x})+(\nabla_\mathbf{x}\mathbf{h}(\mathbf{x}))^\mathrm{T}\boldsymbol{\lambda}+(\nabla_\mathbf{x}\mathbf{g}(\mathbf{x}))^\mathrm{T}\boldsymbol{\pi}\\ \mathbf{h}(\mathbf{x})\\ \sigma\boldsymbol{\mu}\end{bmatrix}=\mathbf{0},$$

where: $\mathbf{\Phi}_\mu(\mathbf{z},\boldsymbol{\pi})=[\varphi_\mu(z_i,\pi_i);i=1,2,...,n_g]^\mathrm{T}$, $\boldsymbol{\mu}=\mu\mathbf{e}$, $\mathbf{y}=[\mathbf{z},\boldsymbol{\pi},\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}]^\mathrm{T}$, $\sigma\in(0,1)$.
Furthermore, the smoothing parameter μ (with scaling parameter σ) is treated as an additional variable, in the system of equations (5). In the experiments, the modified Fischer-Burmeister function of the following form has been used:

$$\text{(6)}\quad \varphi_\mu(z,\pi)=z+\pi-\sqrt{z^2+\pi^2+2\mu}\,.$$

It is assumed that the system (5) is solved using damped Newton's method, and in the subsequent iterations value of parameter μ is reduced to zero. At the *k*-th iteration, the update $\Delta\mathbf{y}$ is determined by solving the following sparse system of linear equations:

$$\text{(7)}\quad \nabla_\mathbf{y}\mathbf{\Psi}(\mathbf{y}^k)\Delta\mathbf{y}^k=-\mathbf{\Psi}(\mathbf{y}^k),$$

where: $\nabla_\mathbf{y}\mathbf{\Psi}_{\mu^k}(\mathbf{y}^k)$ – Jacobi matrix of vector function $\mathbf{\Psi}_{\mu^k}(\mathbf{y}^k)$ at point $\mathbf{y}^k$, $\Delta\mathbf{y}=[\Delta\mathbf{z},\Delta\boldsymbol{\pi},\Delta\mathbf{x},\Delta\boldsymbol{\lambda},\Delta\boldsymbol{\mu}]^\mathrm{T}$.

The solution of system of equations (7) corresponds to the solution of a linear system of equations, which in matrix notation takes the following form:

$$\text{(8)}\quad \begin{bmatrix}\mathbf{D}_\mathbf{z} & \mathbf{D}_\pi & \mathbf{0} & \mathbf{0} & \mathbf{D}_\mu\\ \mathbf{1} & \mathbf{0} & \nabla_\mathbf{x}\mathbf{g}(\mathbf{x}) & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \nabla_\mathbf{x}\mathbf{g}(\mathbf{x})^\mathrm{T} & \nabla_\mathbf{x}^2 L & \nabla_\mathbf{x}\mathbf{h}(\mathbf{x})^\mathrm{T} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \nabla_\mathbf{x}\mathbf{h}(\mathbf{x}) & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{z}\\ \Delta\boldsymbol{\pi}\\ \Delta\mathbf{x}\\ \Delta\boldsymbol{\lambda}\\ \Delta\boldsymbol{\mu}\end{bmatrix}=-\mathbf{\Psi}(\mathbf{y}),$$

where:
$\mathbf{D}_\mathbf{z}=\nabla_\mathbf{z}\mathbf{\Phi}_\mu=\mathrm{diag}[\partial\varphi_\mu(z_i,\pi_i)/\partial z_i;i=1,2,...,n_g]$,
$\mathbf{D}_\pi=\nabla_\pi\mathbf{\Phi}_\mu=\mathrm{diag}[\partial\varphi_\mu(z_i,\pi_i)/\partial\pi_i;i=1,2,...,n_g]$,
$\mathbf{D}_\mu=\nabla_\mu\mathbf{\Phi}_\mu=\mathrm{diag}[\partial\varphi_\mu(z_i,\pi_i)/\partial\mu;i=1,2,...,n_g]$,
$\nabla_\mathbf{x}^2 L=\nabla_\mathbf{x}^2 f(\mathbf{x})+\nabla_\mathbf{x}^2(\boldsymbol{\lambda}^\mathrm{T}\mathbf{h}(\mathbf{x}))+\nabla_\mathbf{x}^2(\boldsymbol{\pi}^\mathrm{T}\mathbf{g}(\mathbf{x}))$.

The new value of the vector of variables $\Delta\mathbf{y}$ is determined according to the equation:

$$\text{(9)}\quad \mathbf{y}^{k+1}=\mathbf{y}^k+\alpha^k\Delta\mathbf{y}^k,$$

where: k – number of iteration, α^k – step length in the direction of vector $\Delta\mathbf{y}^k$.

The choice of step length α^k, is based on the backtracking Armijo line search rule, in which the distance from the solution point is determined by the equation:

$$\text{(10)}\quad \theta_\mu(\mathbf{z},\boldsymbol{\pi})=(\mathbf{\Phi}_\mu(\mathbf{z},\boldsymbol{\pi})^\mathrm{T}\mathbf{\Phi}_\mu(\mathbf{z},\boldsymbol{\pi}))/2\,.$$

The choice of step length α^k is performed according to the following rule: for given values of coefficients, $\sigma\in(0,1]$, $\alpha_1\in(0,1]$ find such:

$$\text{(11)}\quad \alpha^k=\max\{\alpha_1^p : p=0,1,2,...\},$$

for which the following condition is fulfilled:

$$\text{(12)}\quad \theta_\mu\left(\mathbf{z}^k+\alpha_1^p\Delta\mathbf{z}^k,\boldsymbol{\pi}^k+\alpha_1^p\Delta\boldsymbol{\pi}^k\right)\le\beta\left(1-\sigma\alpha_1^p\right)\mu^k,$$

where $\beta=\theta_{\mu^0}(z^0,\pi^0)/\mu^0$.

For step length α^k determined in this manner, the new value of the variables vector $\mathbf{y}$ is determined, according to the equation (9).

Analytical reduction of the system

One of the improvements that has great impact on the NIP algorithm computation time [4] is the analytical reduction of the system of equations (8), which leads to the formulation of a reduced system:

$$\text{(13)}\quad \begin{bmatrix}\mathbf{W} & \nabla_\mathbf{x}\mathbf{h}(\mathbf{x})^\mathrm{T}\\ \nabla_\mathbf{x}\mathbf{h}(\mathbf{x}) & \mathbf{0}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\boldsymbol{\lambda}\end{bmatrix}=-\begin{bmatrix}\mathbf{s}\\ \mathbf{h}(\mathbf{x})\end{bmatrix},$$

where: $\mathbf{W}=\nabla_\mathbf{x}^2 L+\nabla_\mathbf{x}\mathbf{g}(\mathbf{x})^\mathrm{T}\mathbf{D}_\pi^{-1}\mathbf{D}_\mathbf{z}\nabla_\mathbf{x}\mathbf{g}(\mathbf{x})$,
$\mathbf{s}=\mathbf{r}_\mathbf{x}+\nabla_\mathbf{x}\mathbf{g}(\mathbf{x})^\mathrm{T}\mathbf{D}_\pi^{-1}(-\mathbf{r}_\mathbf{z}+\mathbf{D}_\mathbf{z}\mathbf{r}_\pi-\mathbf{D}_\mu\Delta\boldsymbol{\mu})$.

Then, variables update $\Delta\mathbf{z}$ and $\Delta\boldsymbol{\pi}$ values are computed using the following equations:

$$\text{(14)}\quad \begin{aligned}\Delta\mathbf{z}&=-\mathbf{r}_\pi-\nabla_\mathbf{x}\mathbf{g}(\mathbf{x})\Delta\mathbf{x},\\ \Delta\boldsymbol{\pi}&=\mathbf{D}_\pi^{-1}(-\mathbf{r}_\mathbf{z}-\mathbf{D}_\mathbf{z}\Delta\mathbf{z}-\sigma\mu\mathbf{D}_\mu).\end{aligned}$$

Parallelization and further algorithm improvements

Presented non-interior point algorithm variant is not directly dedicated (by design) to efficient parallel implementation. However, by thoroughly studying the structure and performance of the sequential variant of NIP algorithm, some improvements have been proposed.

The main idea behind the parallelization of the OPF computations using presented non-interior point algorithm implementation is to distribute the most time consuming, independent components computations across multiple computing units (processors).

Each iteration of NIP method involves the computation of vector of equality constraints $\mathbf{h}(\mathbf{x})$, vector of inequality constraints $\mathbf{g}(\mathbf{x})$, vector $\nabla_\mathbf{x} f(\mathbf{x})$ and matrices $\nabla_\mathbf{x}\mathbf{h}(\mathbf{x})$ and $\nabla_\mathbf{x}\mathbf{g}(\mathbf{x})$. These components are computed as independent tasks to form right-hand-side of (13). In order to obtain $\nabla_\mathbf{x}^2 L$ matrix, and hence $\mathbf{W}$ matrix, Hessian matrices: $\nabla_\mathbf{x}^2(\boldsymbol{\lambda}^\mathrm{T}\mathbf{h}(\mathbf{x}))$, $\nabla_\mathbf{x}^2(\boldsymbol{\pi}^\mathrm{T}\mathbf{g}(\mathbf{x}))$ and $\nabla_\mathbf{x}^2 f(\mathbf{x})$ are also computed as independent tasks. An important fact worth mentioning is, that in the implemented algorithm, individual components of the matrix (8) and hence (13) were determined analytically. In addition, each iteration also involves the variables update, which is also computed using multithreading.

Moreover, it is possible to obtain some further improvements that lead to the shortening the time of calculations, by the proper selection of algorithm start point. Proposed, modified non-interior point based optimal power flow algorithm initialization technique is based on power flow solution—in contrast to the standard method used [4] (so called *flat-start*), in which initial values of angles were

set to zero, and the values of voltage, active and reactive power modules at production buses were set at the average calculated from the minimum and maximum permissible values.

As observed, performing the power flow (PF) calculations before the actual OPF algorithm execution, and using the PF solution values as initial values to the OPF algorithm, reduces the number of iterations needed to obtain the final result. Therefore, reducing the number of iterations needed to obtain the solution, results in the shortening overall execution time.

Both of the mentioned algorithm improvements have been implemented and tested computationally.

Computational experiments

The experiments have been performed using software created in C# language, with C/C++ wrapper library to the Intel® Math Kernel Library (MKL) routines.

The developed software utilizes a custom object-oriented C# implementation of sparse matrix representation for storing data used in the application. The algorithms, mentioned, have been implemented in the C# language also. MKL library [7] has been used to solve sparse linear systems via LU factorization with sparse matrix minimum degree ordering.

In multithreaded variant of algorithm implementation the Task Parallel Library (TPL) [8], [9] has been used. The main advantage of using TPL is the fact, that it scales the degree of concurrency dynamically to most efficiently use all the processors that are available in the system. In addition, it handles the partitioning of the work, the scheduling of threads on the ThreadPool, cancellation support and state management.

Test systems data are derived from the MATPOWER package available for scientific and educational applications [10]. The 300, 2383 and 2746 bus test systems, were used in the simulations. Some statistics for the test power systems are presented in Table 1, where N_b denotes the number of buses, N_g denotes number of generator buses, N_o symbolizes the number of load buses and N_l represents the number of branches.

Table 1. Statistics for the test power systems

Test system	*Nb*	*Ng*	*No*	*Nl*
Case-300	300	69	231	411
Case-2383	2383	327	2056	2896
Case-2746	2746	363	2383	3279

Some statistics for the non-interior-point based non-linear programming problem (NLP) are displayed in Table 2, where, for each test system, is given: the number of decision variables n_x, the number of equality constraints n_h, the number of inequality constraints n_g, square matrix (13) size of the analytically reduced system, denoted by M.

Table 2. Sizes of the NLP model problem

Test system	n_x	n_h	n_g	*M*
Case-300	738	600	1698	1338
Case-2383	5420	4766	11866	10186
Case-2746	6220	5492	13506	11712

To compare the performance of the implemented sequential NIP based OPF method algorithm, the multi-threaded (MT) version of the algorithm has been implemented utilizing .NET 4 Task Parallel Library (TPL) [8], [9]. The results of experiments are shown in Table 3, where T_{SEQ} denotes sequential algorithm execution time, T_{MT} denotes multi-threaded algorithm execution time and finally T_{MTPF} denotes multi-threaded algorithm execution time with power flow initialization technique applied. S_M and S_{MPF} denote speedup of algorithms, respectively multi-threaded version and multi-threaded version with power flow initialization technique.

All of the experiments were performed utilizing implementation of non-interior point algorithm variant, with analytically reduced system of linear equations (13).

Computations were performed using the Intel Core i7-3770K @ 3.5 GHz machine with the 64-bit edition of the Microsoft Windows 10 operating system.

Table 3. NIP method based optimal power flow results obtained for sequential and improved algorithm implementation

Test system	T_{SEQ}	T_{MT}	$S_M = \frac{T_{SEQ}}{T_{MT}}$	T_{MTPF}	$S_{MPF} = \frac{T_{SEQ}}{T_{MTPF}}$
	ms	*ms*		*ms*	
Case-300	409	377	1,09	351	1,17
Case-2383	5191	4373	1,19	4098	1,27
Case-2746	5877	4639	1,27	4382	1,34

Figure 1 illustrates the overall optimal power flow computation time in milliseconds depending on the test system case. Growing value of speedup for larger system cases (figure 2) indicates future promise for using such algorithm improvements in order to obtain substantial performance benefits for large problems.

Fig. 1. NIP based OPF computation time in ms depending on the test system case

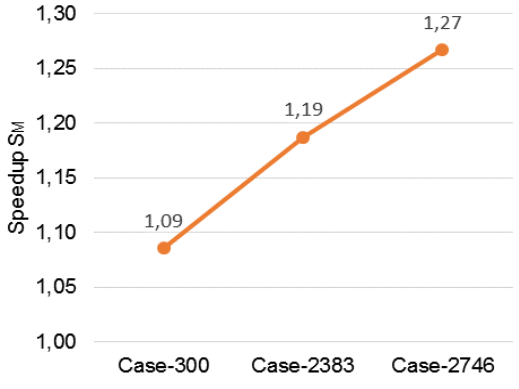

Fig. 2. Multithreaded variant of NIP based OPF speedup

Table 4 collects the results of experiments, that shows the influence of proposed initialization technique to the number of iterations of OPF algorithm and therefore to execution time.

Table 4. NIP method based optimal power flow results obtained for multithreaded variant of algorithm with PF initialization technique

Test system	*MT with flat start*		*MT with PF initialization technique*			
	No.of iter.	T_{MT}	T_{PF}	*No.of iter.*	T_{MT}	$T_{PF}+T_{MT}$
		ms	*ms*		*ms*	*ms*
Case-300	17	377	12	15	339	351
Case-2383	25	4373	70	23	4028	4098
Case-2746	24	4639	81	33	4300	4381

In table 4, *MT* denotes multithreaded variant of implemented algorithm, T_{PF} denotes power flow algorithm execution time. Additionally, for a specific test system, in a series of experiments, algorithm converged to the same solution point (with respect to the value of the objective function and variables vector values) meeting given calculation accuracy.

Conclusion

This paper discuses some issues that are directly related to an efficient implementation of the non-interior point algorithm for nonlinear formulation of optimal power flow problem.

Although, the nonlinear NIP algorithm is not well-suited to parallel implementations, it is worth to utilize the possibilities given by multithreaded architectures and compute independently some components during algorithm execution. Presented approach takes advantage of the problem structure in order to obtain increased efficiency for the overall method. The block structure of the linear system matrix solved in each iteration of the NIP method has been exploited in the parallel computation.

Task Parallel Library significantly simplifies parallel development by providing routines that can automatically distribute tasks across the computer's available CPUs and allows to create parallel code in a natural way without having to work directly with threads or the thread pool.

Accordingly to further improvements. The sparsity pattern of individual components of the matrix (13) as well as reordering to minimize fill-in remain the same. Since, the computation of linear system of equations within an iteration of NIP algorithm, demands great computational effort, it may be advantageous to use the same, given matrix sparsity pattern in subsequent iterations.

***Author**: Marcin Połomski, PhD, Eng., Silesian University of Technology, Faculty of Electrical Engineering, Institute of Electrical Engineering and Computer Science, ul. Akademicka 10B, 44-100 Gliwice, E-mail: marcin.polomski@polsl.pl.*

REFERENCES

[1] Momoh J., Electric Power System Applications of Optimization, *CRC Press*, 2008.

[2] Pandya K. S., Joshi S. K., A survey of optimal power flow methods. *Journal of Theoretical and Applied Information Technology*, vol. 4 (2008), no. 5, 450-458

[3] Baron B., Kraszewski T., Pasierbek A., Połomski M., Sokół R.: Performance comparison of conjugate gradient, quasi-newton and non-interior point optimization algorithms. *Międzynarodowa Konferencja z Podstaw Elektrotechniki i Teorii Obwodów IC-SPETO*, Ustroń 2009.

[4] Połomski M., Baron B.: Optimal Power Flow by Interior Point and Non Interior Point Modern Optimization Algorithms. Acta Energetica 1/14 (2013), 132-139.

[5] Benson H. Y., Shanno D. F., Vanderbei R. J.: A comparative study of large-scale nonlinear optimization algorithms. Department of Operations Research and Financial Engineering, Princeton University, Princeton, Tech. Rep. ORFE 01-04, 2001

[6] Torres G. L., Quintana V. H., An interior-point method for nonlinear optimal power flow using voltage rectangular coordinates. *IEEE Transactions on Power Systems* vol. 13 (1998), no. 4, 1211-1218

[7] Intel Math Kernel Library, https://software.intel.com/en-us/intel-mkl.

[8] Jajczyk J., Optimisation using a parallelised genetic algorithm on a personal computer, *Przegląd Elektrotechniczny*, 91 (2015), nr 7, 36-38

[9] Duffy J., Concurrent Programming on Windows. Microsoft .Net Development Series. *Addison-Wesley Professional*, 2008

[10] Zimmerman R. D., Murillo-Sanchez C. E., Thomas R. J., MATPOWER: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education, *Power Systems, IEEE Transactions on,* vol. 26 (2011), no. 1, 12-19.