Jarosław JAJCZYK

Poznan University of Technology, Institute of Electrical Engineering and Electronics

doi:10.15199/48.2015.07.12

Optimisation using a parallelised genetic algorithm on a personal computer

Abstract. *The paper discusses the possibility of using personal computers as computing units in parallelised computation. Optimisation computation based on a parallel genetic algorithm is proposed. A method of parallelised computing is discussed using a personal computer equipped with a multi-core processor. The article presents examples of results obtained as a result of parallelised computing.*

Streszczenie. *W pracy omówiono możliwości wykorzystania komputerów osobistych jako jednostek liczących w obliczeniach równoległych. Zaproponowano realizację obliczeń optymalizacyjnych bazujących na zrównoleglonym algorytmie genetycznym. Omówiono metodę zrównoleglenia obliczeń na komputerze osobistym wyposażonym w procesor wielordzeniowy. W artykule podano przykładowe wyniki uzyskane dzięki zrównolegleniu obliczeń (**Optymalizacja metodą zrównoleglonego algorytmu genetycznego na komputerze osobistym**).*

Keywords: optimisation, genetic algorithm, parallel computing.
Słowa kluczowe: optymalizacja, algorytm genetyczny, obliczenia równoległe.

Introduction

Recent years saw a dynamic growth of computer hardware. This resulted in an increase in the computing capacity of computers (including personal ones), which became efficient units for complex computation [1, 2, 3]. Computer efficiency was initially increased by raising the frequency of processor timing, then by introducing the multithreading technology, and finally by constructing multi-core units (16-core processors are currently produced) [4].

Unfortunately, an increased number of computing units does not always mean a shorter time of computing. If not adopted to a multi-core architecture, software is not more efficient when installed on such computers. This problem can be solved by preparing software using the .NET platform and its TPL library (Task Parallel Library), which features a set of classes to support the programmer while creating applications to parallelise computations (in the tasks where parallelising is possible) [4, 5, 6].

Parallel computing environment

As of the 4.0 version the .NET programming platform has been extended with the TPL library (Task Parallel Library), which is adapted to concurrent programming. TPL extends conventional threads using the Task class. This class makes it possible to execute tasks and loops in a parallel manner. Figure 1 presents an architecture of basic parallel programming techniques available from the .NET Framework platform in the TPL library [5, 6].

In the `System.Threading.Tasks` namespace, the Parallel class is also available, which can be used to parallelise threads. This class is especially useful in tasks that do not require threads to be synchronised. Methods of executing “for” and “foreach” loops in a parallel manner are made available by the Parallel class. When used, they burden all processor cores. The sequence of iterations and the division thereof among the processor cores are automatic [5, 6].

The `Parallel.For` method automatically synchronises all tasks it executes prior to completion. Within further repetitions, there is no danger of data erasure. In order for the `Parallel.For` loop to be used correctly, no recurrent dependency may occur between tasks executed in particular loop iterations. A further iteration cannot depend on the variable of the preceding iteration. The iterations of a parallel loop are not executed in the index sequence [5, 6].

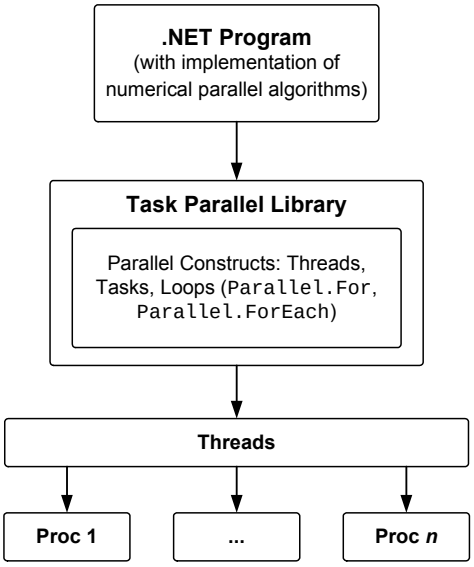

Fig. 1. Parallel programming architecture using the mechanism available in the .NET Framework platform

The parallel loop structures listed are the simplest forms of asynchronous computing tasks to be parallelised, but due to their construction they are not always the most effective in terms of numeric analyses. With the input into the division of tasks among particular cores, the speedup of computing is not proportional to the number of the calculating units used. The longer the time of the computing of one iteration, the larger it is. Then, the times of thread division and synchronization become negligible [5, 6, 9].

Optimisation algorithm parallelising

The Parallel class available from the .NET platform can be used effectively when tasks that do not require synchronisation are parallelised. This is the case when optimisation computing is used with a genetic algorithm method. In this method every generation has adjustment function values determined for particular individuals. These calculations can be carried out in parallel, being independent of one another (each individual is adjusted irrespective of the other individuals). Then, a series of genetic operations is conducted using a pool of individuals [7, 8, 9, 10].

With the Parallel class, it is possible to parallelise the computation of individuals' adjustment indices. In this case one (main) thread is responsible for all genetic operations to individuals. These operations are not subject to parallelising. The adjustment indices are computed in the `Parallel.For` loop with the available processor cores. (Fig. 2). Each loop iteration is executed by a different core (this process is subject to parallelisation).

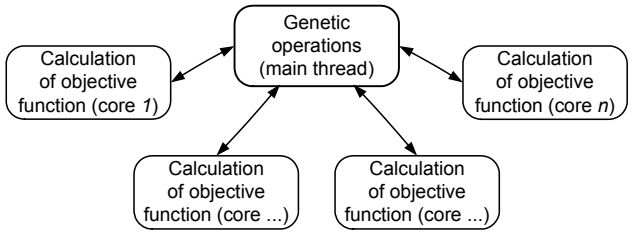

Fig. 2. Parallelised genetic algorithm organisation using mechanisms available from the NET Framework platform

When the `Parallel.For` loops are created, corresponding in iteration quantitative terms to the number of individuals in a generation, the computing results are turned over to the main thread, followed by genetic operations.

Object to be optimised

The advantages of using a computer equipped with a multi-core computer for parallelised computations were examined by optimising the dimensions of a three-phase, three-conductor unscreened high current busway with solid insulation. Figure 3 shows its cross-section.

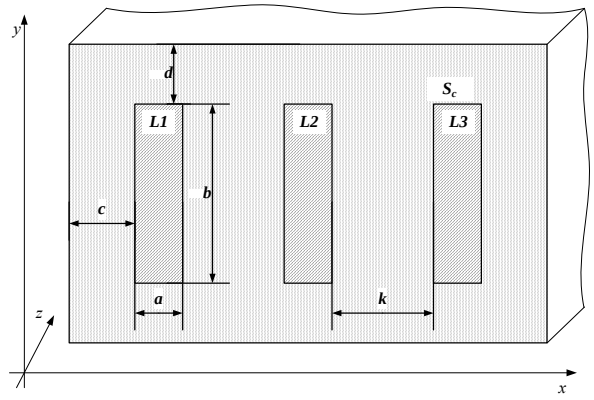

Fig. 3. Cross-section of the high current busway with solid insulation

Phase conductors, each of a S_c cross-section area, are embedded in solid insulation, made out of a component of epoxide resins. The geometry of the system is conditioned by five variables: *a, b* – dimensions of the cross-section of the phase conductor; and *c, d, k* – dimensions determining the distribution of conductors in the insulation.

All electrodynamics calculations start with defining the distribution of a current density *J(x,y)* in live working conductors with specified phase currents [2]. It can be obtained by solving the system of integral equations (1).

$$\begin{aligned} &\underline{J}(x,y) = -\frac{1}{2\pi} j\omega\mu\gamma \int_{S_c} \underline{J}(x',y') \ln \frac{1}{\sqrt{(x-x')^2+(y-y')^2}} dx'dy' \\ &\int_{S_c} \underline{J}(x',y')dx'dy' = \underline{I}_c \end{aligned} \tag{1}$$

where: μ – magnetic permeability of the conductor material; ω – pulsation; γ - electrical conductivity of the conductor material; *(x, y)* – the observation point; *(x', y')* – the source point; S_c - cross-section area of the conductor.

With the Joule law, the current density distribution makes it possible to determine the loss of power P_c per a length unit of particular phase conductors, as defined by the dependence:

$$P_c = \frac{1}{\gamma} \iint_{S_c} \left|\underline{J}(x',y')\right|^2 dx'dy' \tag{2}$$

Approximate current density vector distribution can be used to determine the volumetric density distribution of the thermal power ρ_c generated in the phase conductors [2]:

$$\rho_c(x,y) = \frac{\left|\underline{J}(x',y')\right|^2}{\gamma} \tag{3}$$

This is used to determine temperature distribution in the system. It is decisive in the busway geometric dimensions, conditioning the ability to carry heat away. The heat generated by active power inside the phase conductors complies with the Poisson's equation [2]:

$$\nabla^2 T(x,y) = -\frac{\rho_c(x,y)}{\lambda_c} \tag{4}$$

where: λ_c – thermal conductivity of the phase conductor material.

As regards temperature, Laplace's equation is complied with inside the insulator and outside the busway:

$$\nabla^2 T(x,y) = 0 \tag{5}$$

The heat energy generated in the phase conductors is transferred to the insulation where, as a result of conductivity, the heat is carried to the busway surface and then to the environment by convection and radiation. On the surface of the busway, the following equation is complied with [2]:

$$\lambda_i \frac{\partial T(x,y)}{\partial n} = -\alpha_{CR}[T(x,y) - T_o] \tag{6}$$

where: α_{CR} – heat transfer coefficient of convection and radiation (defining methods are given in [2]), T_o – ambient temperature.

The calculations also allow for a voltage gradient and the forces acting in the system. Details concerning the solving of the equations (1-6) and the determining of other parameters of electrodynamics can be found in the publications [2].

As an optimisation criterion, the paper assumes the minimisation of the busway production and operation costs for a preset time, and with a number of limitations. The objective function (7) is financial in character and uses geometric variables conditioning the cross-section area of the busway (investment outlays) and the active power loss (operations costs).

$$S(\mathbf{u}) = k_{invest.} + k_{exploit.} \tag{7}$$

where: $\mathbf{u}$ – decision variable vector; $k_{inwest.}$ – investment costs, $k_{exploit.}$ – operation costs.

A number of limitations must be provided for by the objective function $S(\mathbf{u})$ minimized in the process of optimization. The most important include the maximum temperature of the working conductor and insulator, maximum voltage gradients, maximum forces acting, in the steady and short-circuit state, as well as requirements of standards, for example, on short-circuit currents [2].

Computing results

The paper compares optimisation computing times using a genetic algorithm method and computers featuring different hardware configurations and conventional `for` and `Parallel.For` loops. A three-phase, unscreened high current busway with solid insulation is the object to be optimised [2].

Electrodynamics computing for that type of objects is complex, making the time of determining the adjustment of particular individuals relatively long. Hence, it might be expected that the time input to divide tasks for particular cores is negligible.

The tests used personal computers with different hardware configurations (table 1).

Table 1. Parameters of the computers used for computing

Parameter	Computer 1	Computer 2	Computer 3
Processor	Core2– T8100	i3 – 2120	i7 – 3770
Number of cores	2	2	4
Threads per core	1	2	2
Clock speed	2.10 GHz	3.30 GHz	3.40 GHz
Cache memory	3MB	3 MB	8 MB
RAM type	DDR2 2 GB	DDR3 8 GB	DDR3 16 GB

Computation time was measured for each computing unit in two variants, namely for sequential computation and parallel computation (using the Parallel class). Figure 4 presents the results obtained.

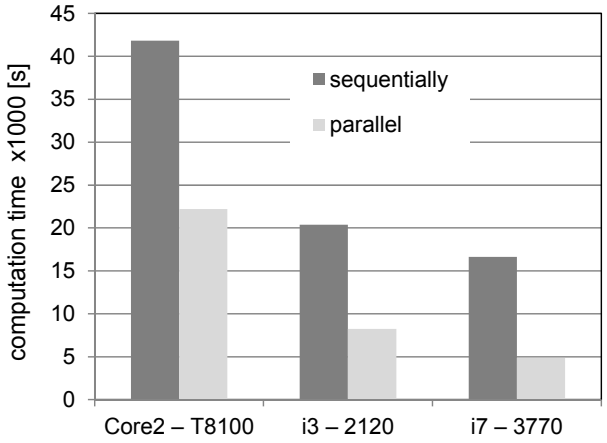

Fig. 4. Optimisation calculation time of a sample task depending on the processor type and calculation variant

The longest computation time was obtained with the computer of the smallest capacity, without parallelised computing. It was approximately 41830 seconds. The shortest optimisation computing time of the same task was 4850 seconds. It was obtained with a computer of the largest computing capacity with a processor of four cores simultaneously executing eight threads. In this case parallelized optimising computation was carried out using the `Parallel.For` loop.

IV. Conclusion

The tests carried out proved that, irrespective of the multi-core processor model used, the parallelisation of optimising computation using the genetic algorithm method with the TPL mechanisms available from the TPL library shortened the time of the execution of the computation. With the least efficient 2-core processor, the computing speedup was 1.9, and with the most effective, 4-core one, it was 3.4.

It was possible to shorten the computing time to such an extent because the time input into the division of tasks among the particular cores was negligible in relation to the electrodynamics computation with a single processor core.

REFERENCES

[1] Bednarek K., Kasprzyk L., Speeding up of electromagnetic and optimization calculations by the use of the parallel algorithms, *Przegląd Elektrotechniczny*, 85 (2009), nr 12, 65-68

[2] Jajczyk J., Optimization of the geometry of high currency busducts with solid insulation with the use of the modified genetic algorithm method, *Ph D thesis, Poznan University of Technology*, (2008)

[3] Skowronek K., Trzmiel G., The method for identification of fotocell in real time, Przegląd Elektrotechniczny, 83 (2007), nr 11, 108-110

[4] Wyrzykowski R., PC computer clusters and multi-core architectures. Structure and use, *Exit*, (2009)

[5] Warczak M., Matulewski J., Pawłaszek R., Sybilski P., Borycki D., Dziubak T., Programowanie równoległe i asynchroniczne w C# 5.0, Helion, (2013)

[6] Albahari J., Threading in C#, *O'Reilly Media*, (2010), www.albahari.com/threading/ (04.10.2014)

[7] Fogel D. B., Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, *IEEE Press/Wiley-Interscience*, (2006)

[8] Goldberg D.E., Genetic Algorithms in Search, Optimization and Machine Learning, *Addison-Wesley Publishing Company*, (1989)

[9] Jajczyk J., Optimization calculations with the genetic algorithm method on a computer cluster, *Przegląd Elektrotechniczny*, 90 (2014), nr 4, 232-234

[10] Ramirez J. A., Saldanha R. R., Takahashi R. H. C., Vasconcelos J. A., Improvements in Genetic Algorithms, *IEEE Transactions on Magnetics*, 37 (2001), No. 5, 3414-3417

***Authors**: Jarosław Jajczyk ,Phd, Eng., Poznan University of Technology, Faculty of Electrical Engineering and Electronics, ul. Piotrowo 3a, 60-965 Poznań, e-mail: jaroslaw.jajczyk@put.poznan.pl.*