Zuzanna KRAWCZYK[1], **Jacek STARZYŃSKI**[1,2]

[1]Warsaw University of Technology, [2]Military University of Technology, Institute of Electronic Systems

Parallel implementation of exact formulae for magnetic field of axisymmetric current distributions

Abstract. *This paper discusses parallel implementation of Python program which computes magnetic induction of a cylindrical coils. The speed-up which can be obtained by use of two Python libraries – MPI4Py and Parallel Python is compared. The use of exact analytical expressions and their parallel implementations allow to achieve computational speed appropriate for practical applications, significantly better than in the case of general numerical methods like FEM.*

Streszczenie. *Artykuł przedstawia równoległą implementację, w języku Python, obliczeń indukcji pola magnetycznego od cewek cylindrycznych oraz ósemkowej. Porównano przyspieszenie działania programu osiągnięte przy wykorzystaniu dwóch bibliotek umożliwiających zrównoleglenie jego kodu: MPI4Py i Parallel Python. Użycie dokładnych wyrażeń analitycznych i ich równoległa implementacja pozwalają na osiągnięcie szybkości obliczeń odpowiedniej dla zastosowań praktycznych, istotnie większej niż w przypadku metod numerycznych ogólnego zastosowania, takich jak metoda elementów skończonych.* ***(Równoległa implementacja analitycznego modelu pola magnetycznego generowanego przez cewki cylindryczne)***

Keywords: coils,magnetic induction, Python, Parallel Python, MPI4Py, parallel processing, Biot-Savart law
Słowa kluczowe: cewki, indukcja pola magnetycznego, Python, Parallel Python, MPI4Py, przetwarzanie równoległe, prawo Biota-Savarta

Introduction

Magnetic field of cylindrical and figure-of-eight (FOE) coil may be used for medical purposes, for instance in the case of magnetic stimulation of nervous system in epilepsy treatment. To simulate of such treatment, the magnetic field needs to be calculated in a large number of points. Computation can be performed with numerical methods as finite element method or boundary element method, but results of such computations can be not accurate enough close to the coil. Moreover, such computation may be time- and resource-consuming as it requires to discretize not only the simulated object but the coil and surroundings as well.

Alternative approach is based on the implementation of exact formulae for magnetic induction. Formulae employed in our implementation were originally presented in paper [1] by John T. Conway. Analytical expressions obtained by Conway are composed of one-dimensional integrals containing a combination of trigonometric and elementary functions. Implementation of such expressions allows to compute magnetic field much more efficiently than it was the case with the earlier, more straightforward approach using elliptic integrals (the latter method is time consuming due to the necessity to integrate some quickly varying functions).

For massive simulation problems even the use of Conway formulae turns out to be not sufficient. The code has to be additionally parallelized in order to improve the performance and bring it to the acceptable level.

Analytical formulae for magnetic induction

A cylindrical coil can be characterized by the following three parameters: inner radius R_1, outer radius R_2 and height L, as shown on Fig. 1. Due to axial symmetry the field can be described in two-dimensional, cylindrical coordinates r, z.

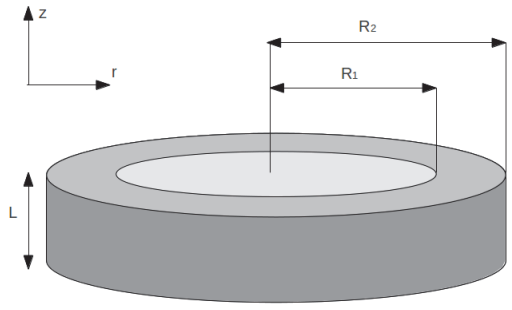

Fig. 1. Cylindrical coil

Many more complicated patterns can be composed from simple coils and their field can be calculated on the basis of superposition principle. For instance, a FOE coil is composed of two identical cylindrical coils with opposite directions of electric current. Thus it is sufficient to focus on the computation of magnetic induction from a cylindrical coil. In paper [1] the field is given by following expressions:

$$(1)\quad B_z(r,z) = \begin{cases} T_2(r,|z|) & \\ -T_2(r,|L-z|) & \text{for } z<0 \\ T_B(r) - T_2(r,|z|) & \\ -T_2(r,|L-z|) & \text{for } 0 \le z < L \\ T_2(r,|L-z|) & \\ -T_2(r,|z|) & \text{for } z > 0 \end{cases}$$

$$(2)\qquad B_r(r,z) = T_3(r,|L-z|) - T_3(r,|z|),$$

where

$$(3)\quad T_B(r) = \begin{cases} \mu_0 I_\phi (R_2 - R_1) & \text{for } r < R_1 \\ \mu_0 I_\phi (R_2 - r) & \text{for } R_1 \le r < R_2 \\ 0 & \text{for } r > R_2. \end{cases}$$

(4)

$$\begin{aligned} T_1(r,a) = &-\frac{\mu_0 I_\phi r a}{4} \ln\left(\frac{R_2 + r + \sqrt{(R_2+r)^2+a^2}}{R_1 + r + \sqrt{(R_1+r)^2+a^2}}\right) \\ &+ \frac{\mu_0 I_\phi {R_2}^3}{2\pi} \int_0^\pi \frac{sin^2\phi d\phi}{X(R_2,r,a,\phi)(a+X(R_2,r,a,\phi))} \\ &- \frac{\mu_0 I_\phi {R_1}^3}{2\pi} \int_0^\pi \frac{sin^2\phi d\phi}{X(R_1,r,a,\phi)(a+X(R_1,r,a,\phi))} \\ &- \frac{\mu_0 I_\phi a}{2\pi} \int_0^\pi (cos\phi)(X(R_2,r,a,\phi) - X(R_1,r,a,\phi))d\phi \\ &- \frac{\mu_0 I_\phi r}{2\pi} \int_0^\pi \arctan\left(\frac{r\sin\phi}{a}\right) \\ &\times \left(\frac{{R_2}^2}{X(R_2,r,a,\phi)} - \frac{{R_1}^2}{X(R_1,r,a,\phi)}\right) \cos\phi \sin\phi d\phi \\ &+ \frac{\mu_0 I_\phi r^2 a}{4\pi} \int_0^\pi \frac{(\phi + \sin\phi\cos\phi)\sin\phi}{R_2 - r\cos\phi + X(R_2,r,a,\phi)} \\ &\times \left(1 + \frac{R_2}{X(R_2,r,a,\phi)}\right) d\phi \\ &- \frac{\mu_0 I_\phi r^2 a}{4\pi} \int_0^\pi \frac{(\phi + \sin\phi\cos\phi)\sin\phi}{R_1 - r\cos\phi + X(R_1,r,a,\phi)} \\ &\times \left(1 + \frac{R_1}{X(R_1,r,a,\phi)}\right) d\phi, \end{aligned}$$

$$
\begin{aligned}
T_2(r,a) =& \frac{\mu_0 I_\phi}{2}(R_2 - R_1) \\
&- \frac{\mu_0 I_\phi a}{2} \times \ln\left(\frac{R_2 + r + \sqrt{(R_2+r)^2 + a^2}}{R_1 + r + \sqrt{(R_1+r)^2 + a^2}}\right) \\
&+ \frac{\mu_0 I_\phi a r}{2\pi} \int_0^\pi \frac{\phi \sin\phi}{R_2 - r\cos\phi + X(R_2, r, a, \phi)} \\
&\times \left(1 + \frac{R_2}{X(R_2, r, a, \phi)}\right) d\phi \\
&- \frac{\mu_0 I_\phi a r}{2\pi} \int_0^\pi \frac{\phi \sin\phi}{R_1 - r\cos\phi + X(R_1, r, a, \phi)} \\
&\times \left(1 + \frac{R_1}{X(R_1, r, a, \phi)}\right) d\phi \\
&- \frac{\mu_0 I_\phi r}{2\pi} \int_0^\pi \arctan\left(\frac{r \sin\phi}{a}\right) \\
&\times \left(\frac{{R_2}^2}{X(R_2, r, a, \phi)} - \frac{{R_1}^2}{X(R_1, r, a, \phi)}\right) sin\phi d\phi,
\end{aligned} \tag{5}
$$

(6)

$$
\begin{aligned}
T_3(r,a) =& \frac{\mu_0 I_\phi r}{4} \ln\left(\frac{R_2 + r + \sqrt{(R_2+r)^2 + a^2}}{R_1 + r + \sqrt{(R_1+r)^2 + a^2}}\right) \\
&+ \frac{\mu_0 I_\phi}{2\pi} \int_0^\pi cos\phi (X(R_2, r, a, \phi) - X(R_1, r, a, \phi)) d\phi \\
&+ \frac{\mu_0 I_\phi r^2}{4\pi} \int_0^\pi \frac{(\phi + \sin\phi\cos\phi)\sin\phi}{R_2 - r\cos\phi + X(R_2, r, a, \phi)} \\
&\times \left(1 + \frac{R_2}{X(R_2, r, a, \phi)}\right) d\phi \\
&- \frac{\mu_0 I_\phi r^2}{4\pi} \int_0^\pi \frac{(\phi + \sin\phi\cos\phi)\sin\phi}{R_1 - r\cos\phi + X(R_1, r, a, \phi)} \\
&\times \left(1 + \frac{R_1}{X(R_1, r, a, \phi)}\right) d\phi.
\end{aligned}
$$

In the above formulae a stays for $\mid z \mid$ or $\mid L - z \mid$ and expression X is given by:

$$
X(R, r, a, \phi) = \sqrt{R^2 + r^2 + a^2 - 2Rr\cos\phi}. \tag{7}
$$

In order to find out if the use of equations (1)–(2) helps to obtain a significant speed-up, time of sequential computations with the use of Conway expressions was compared with computations in which coefficients (4)–(6) were replaced with expressions using Bessel functions:

(8)

$$
T_1(r,a) = \frac{\mu_0 I_\phi}{2} \int_{R1}^{R2} \int_0^\infty r\prime \frac{J_1(sr\prime) J_1(sr)}{s} e^{-as}\, dsdr\prime
$$

(9)

$$
T_2(r,a) = \frac{\mu_0 I_\phi}{2} \int_{R1}^{R2} \int_0^\infty r\prime J_1(sr\prime) J_0(sr) e^{-as}\, dsdr\prime
$$

(10)

$$
T_3(r,a) = \frac{\mu_0 I_\phi}{2} \int_{R1}^{R2} \int_0^\infty r\prime J_1(sr\prime) J_1(sr) e^{-as}\, dsdr\prime
$$

Time of computations performed on mesh consisting of 676 points with the use of Bessel functions was more than 10 times longer than the time of computations with the use of Conway expressions.

Implementation

The program for computation of magnetic induction was written in Python [2], employing scientific libraries NumPy and SciPy to perform integration and faster vector operations. Matplotlib [3] library was used for visualisation of results. Good performance of Python compiled code is widely reported and according to author's experience this language is an ideal tool for scientific computation due its extremely flexible and friendly syntax not sacrificing performance. Python is available for practically any hardware architecture and operating system which are transparent for the program.

Parallelization

The code was parallelized with the use of two Python libraries: Parallel Python [4] and MPI4Py [5].

Parallel Python is a small, easy to install and to configure library which allows to parallelize code of program on clusters and multiprocessor machines. The library is based on a simple model of task distribution between processors. Moreover, it provides for automatic discovery of available processors, load balancing and authentication based on SHA for network connections. Despite the fact that Parallel Python has been written in high-level programming language, it uses for process communication the socket module which is just an interface to the fast, standard BSD socket implementation. This assures both efficient data transfer and fast program execution. Parallel Python is distributed as an open source software.

MPI4Py is a set of Python bindings for MPI. It introduces an object interface for point-to-point and group communication between processes. The package allows to send single-segment objects, like NumPy arrays. In conjunction with the ability to define in Python own memory-layouts it allows for efficient algorithm implementation.

The execution time measurements were done for a cylindrical coil of the height 7 cm, the outer diameter 5 cm and the inner diameter 3 cm. Regular grids of 6 different sizes: 5x5, 10x10, 20x20, 50x50, 80x80 and 100x100 were spanned for r, z coordinates in range $r \in < 0, 0.1 >, z \in < 0, .1 >$. For each grid the computation process was repeated 10 times. Several options of parallel execution of the code were tested: with the use of Parallel Python and different MPI functions performing point-to-point communication (mpisend, mpiisend methods) and collective communication (functions gather and gatherv). One single-core and one-double-core machines working in parallel (3 cores totally) were used in the tests. Averaged results of tests are shown in Table 1.

Table 1. Computation time of $\vec{B}$ field value for a given option and size of mesh, in seconds.

Option	5x5	10x10	20x20	50x50	80x80	100x100
seq.	0.475	1.654	7.353	58.376	145.327	228.773
mpisend	0.272	0.998	3.743	23.194	58.495	92.917
mpiisend	0.275	0.993	3.714	52.848	59.338	94.624
mpigather	0.276	1.002	3.736	23.205	58.068	92.710
mpiigatherv	0.269	0.991	3.704	22.972	58.622	91.988
pp	0.657	1.091	2.796	17.202	40.371	63.775

The best computing times for the meshes with the largest number of points were obtained with the use of Parallel Python library. For the mesh with 10000 nodes, the program performed computations 3.59 times faster than its sequential version. The speed-up obtained with MPI4Py library is similar for all its implemented communication functions with the best times for the function mpigatherv (2.49 times faster than the sequential version of program, for the mesh with 1000 nodes).

Parallel efficiency of the Parallel Python version is presented in Fig. 2. 50x50 grid was used in those tests. As expected, the execution time drops as $1/p$ function of cores number p.

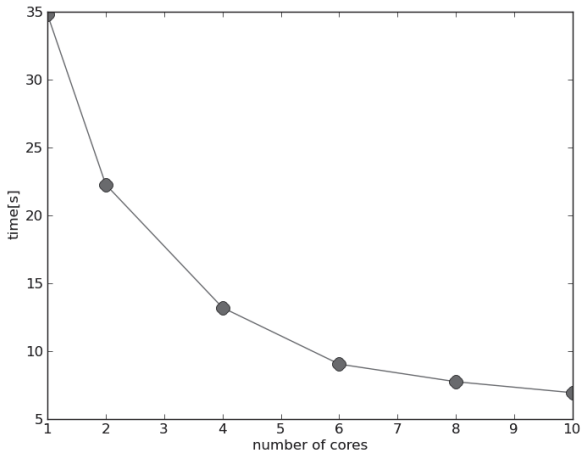

Fig. 2. Execution time (pp) for various number of cores

Results

Computations of magnetic induction were performed for a cylindrical and a FOE coil. Due to the cylindrical symmetry of a simple coil it suffices to consider the field, say, in OXZ plane. In case of FOE coil the magnetic induction is computed in three planes: OXZ, OYZ, and OXY of the Cartesian coordinate system. The program allows also to create two-dimensional graphs of lines of constant field values, both for a cylindrical (Fig. 3) and FOE coil (Fig. 4) as well as flux density images in 2D cross-sections (Fig. 5).

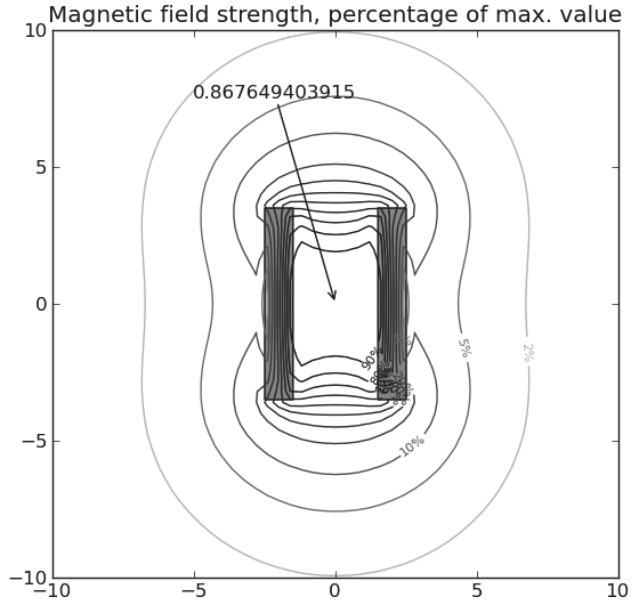

Fig. 3. Constant field (magnitude) lines, for cylindrical coil

Conclusion

We have shown that the parallelization of the code implementing the exact analytical formulae for the magnetic field of a cylindrical coil allows to achieve computation time suitable for the use of this method in many practical applications including massive simulation problems.

Use of the program is not restricted to the cylindrical coils—in many situation more complicated shapes may be simulated by superposition of cylindrical components.

Implementation in Python and use of Parallel Python is straightforward and easy. SciPy and NumPy simplify the effective implementation of massively parallel operations on vectors and Parallel Python quasi-automatically allows us to benefit from multicore and distributed environments.

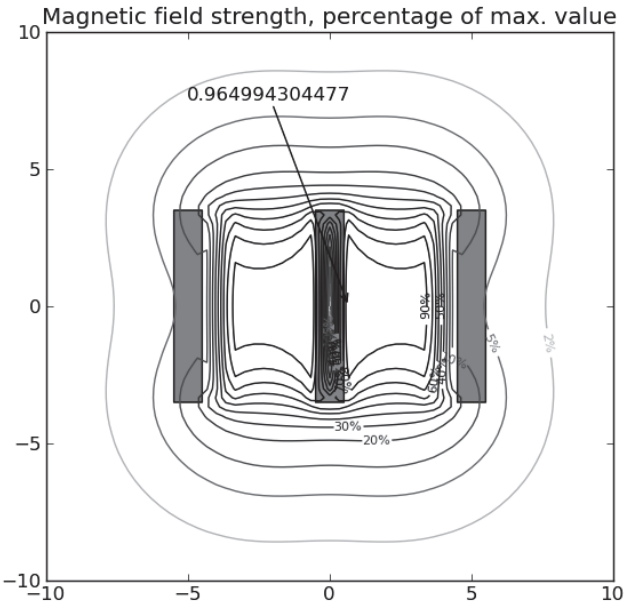

Fig. 4. Constant field (magnitude) lines, for FOE coil

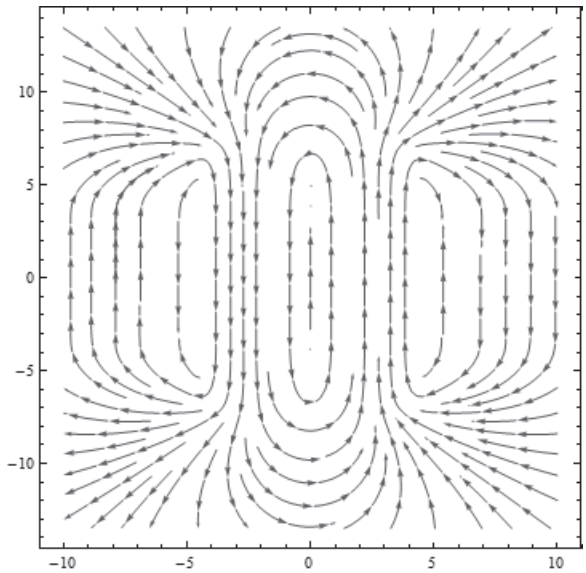

Fig. 5. Flux lines for FOE coil

Acknowledgement: this work was partially supported by Polish Ministry of Science and Higher Education (research grant no. N N510 148838).

BIBLIOGRAPHY

[1] Conway J.T. Trigonometric Integrals for the Magnetic Field of the Coil of Rectangular Crossection, IEEE Transactions on Magnetics 5, 2006, pages 1538-1548, vol. 42
[2] Python Programming Language – Official Website, http://www.python.org/
[3] Matplotlib – a python 2D plotting library, release 1.1.1, http://matplotlib.sourceforge.net/
[4] Parallel Python, http://www.parallelpython.com/
[5] Lisandro Dalcin: Mpi for Python, version 1.3, January 2012, http://mpi4py.scipy.org/

Authors: Zuzanna Krawczyk, Jacek Starzyński, Institute of Theory of Electrical Engineering, Measurement and Information Systems, Faculty of Electrical Engineering, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warszawa, Poland, email: {krawczyz,jstar}@ee.pw.edu.pl