# Novel Hardware Accelerated Magnetic Field Calculation Approach for Circular Coils With Rectangular Cross Section

Davor Dobrota, Nikola Sočec, Lara Vrabac, and Dario Bojanjac, *Member, IEEE*

**This paper presents a framework for calculating the magnetic vector potential, magnetic field, and magnetic field gradient in the case of circular coils with a rectangular cross section. Filaments, pancakes, and thin coils are also included. Unlike surveyed approaches, emphasis is placed on an efficient implementation in a compiled programming language, specifically C++, with multithreading support and hardware accelerated computation. Since all elementary functions, except for the square root, are slow compared to basic arithmetic operations, the approach heavily utilises numerical integration. Specifically, the Gauss-Legendre quadrature was implemented with precomputed parameters. The result are precise and performant methods which can calculate over a million field values every second when using the processor and over 100 million values when using hardware acceleration. Hardware acceleration is implemented for Nvidia graphics cards using the CUDA toolkit. Precision was evaluated by generating field images and it was found that field computation is accurate outside of the coil with some artefacts inside. We aim to provide comprehensive precision testing and additional use cases in a follow-up paper. For ease of use, the C++ code is wrapped in a Python module which can also be used from MATLAB. The code is available on GitHub and the repository name is C-Coil.**

*Index Terms*—**Circular coil, CUDA, magnetic field, magnetic gradient, magnetic vector potential, multithreading, Python module.**

## I. Introduction

CIRCULAR coils are some of the most widely used electromagnetic elements, from small wireless charging coils in modern smartphones, to large superconducting magnets found in MRI machines and particle accelerators. Consequently, many approaches have been developed to calculate the relevant physical quantities of circular coils, with ever increasing efficiency and precision. Some of these quantities include: mutual inductance, force, torque, and values associated with magnetostatic fields - magnetic vector potential and magnetic flux density. The aforementioned quantities are usually tackled separately and often for specific cases, such as a common axis.

Our principal goal is to create a coherent framework which would allow for fast and precise calculation of magnetostatic fields, as well as interactions between circular coils in the form of mutual inductance, force, and torque. It is important to note that this model is satisfactory when the coil is made of a homogeneous material, with no radial current, or when the field near the coil is not of great interest, in the case of a compactly wound solenoid with a large number of turns. The coils are geometrically characterized by their inner radius $R$, thickness $a$, and length $b$. Additionally, the number of turns $N$ and the current $I$ are required to calculate the current density $J$.

Depending on the relationship of thickness and length, 4 types of coils are distinguished:

1) Filament (loop) - negligible length and thickness
2) Thin solenoid (thin) - negligible thickness
3) Pancake coil (flat) - negligible length
4) Circular coil with a rectangular cross section (thick)

All four types are included, but more emphasis is placed on thick coils, as few approaches tackle the problem of a large number of thick circular coils, positioned and oriented in an arbitrary way. Significant performance improvements are possible with a dedicated implementation optimized for multi-coil systems.

Many approaches and methods have been devised to calculate mutual inductance between pairs of circular coils. A large number of approaches, such as [1], [2], [5], [6], and [7], focus on the common axis (coaxial) case, while some of them tackle the parallel axis case, such as [4], [8], and [10]. The only method that calculates mutual inductance in the general case, with satisfactory precision (relative error smaller than 1E-6), is the one shown in [9]. This method represents the most important performance and precision benchmark for our approach; computation takes several seconds to attain 8 significant digits of precision. Another, more recent approach, presented in [11] is less relevant since the error often exceeds 0.1% and computation time is unknown.

Approaches concerning force and torque are less prevalent, most likely because force and toque can be approximated with a numerical derivative of mutual inductance. Approaches [3] and [6] tackle the coaxial force case, whereas [14] presents a precise general method for calculating force between two filaments. The general case with 3 thick coils is covered in [13] and [15], however, the claimed precision is comparatively low and derived from only one test case.

Unlike other surveyed approaches, our focus is, first and foremost, on achieving maximum attainable performance. That was primarily accomplished by using C++, a compiled programming language, which enables the use of AVX2 instructions [19] and powerful compiler optimisations. AVX2

Manuscript received October 14, 2022; revised TBD; accepted TBD. Date od publication TBD; date of current version TBD. Recommended for publication by ... (*Corresponding author: Davor Dobrota.*)

The authors are with the Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia (email: davor.dobrota@gmail.com; nikola.socec@gmail.com; vrabaclara@gmail.com; dario.bojanjac@fer.hr).

Color versions of one or more of the figures are available online at http://ieeexplore.ieee.org. (NOTE: Only Used with Printed Publications).

Digital Object Identifier TBD

instructions are SIMD (*Single Instruction Multiple Data*) instructions that can improve performance by several times in heavy floating-point compute workloads, while also being supported on most desktop and laptop processors.

Since most modern processors have 4 or more cores, and 8 or more hardware threads, extensive multithreading support has also been implemented. If 6 significant digits of precision are sufficient, hardware acceleration with a dedicated Nvidia GPU (*Graphics Processing Unit*) can further improve performance over multithreading by a factor of 10 to 100, depending on the GPU. Full precision can be obtained with specific GPUs which have a low penalty for executing double-precision calculations, such as the Nvidia Titan V.

The design of our approach has been constrained to simple expressions in order to efficiently implement it in the C++ programming language and Nvidia CUDA framework [20]. Basic arithmetic operations are exceedingly fast, especially when addition is coupled with multiplication in FMA (*Fused multiply-add*) instructions [21], often used to benchmark floating-point performance. All other mathematical functions should be used scarcely, except for the square root, which can be executed quickly on the processor, owing to the use of AVX2 instructions. Performance of the square root on the GPU is much lower than that of FMA, but better than other elementary functions. In contrast, other approaches use a very high level programming language, such as MATLAB (Mathworks Inc.) and Mathematica (Wolfram Research Inc.), which have support for powerful functions (e.g. elliptic integrals), but sacrifice performance.

The developed framework is called C-Coil (Circular Coil Object-oriented Interaction Library) [25]. It consists of multiple C++ classes centered around class Coil which models the aforementioned circular coils. Another noteworthy class is CoilGroup used for faster calculations with multi-coil systems, especially when using the GPU. This paper will tackle the field calculation side of our framework. Interactions between coils are easily computed directly from the vector potential and magnetic field. Interaction methods are already available for use in C-Coil, but their expressions and precision will be explored in a follow-up paper.

When the term "method" is used, we refer to the object oriented programming concept. The term "fields" encompasses the magnetic vector potential $\boldsymbol{A}$ (potential), magnetic flux density $\boldsymbol{B}$ (magnetic field), and magnetic field gradient $\mathbf{G}$ (total derivative of $\boldsymbol{B}$, gradient). The gradient has been added due to its use in MRI imaging and particle simulations. It can also be used to approximate the force on a coil if that coil is sufficiently small or far away.

## II. Basic expressions

Since our primary goal is to attain simple expressions, it makes sense to begin with the Biot-Savart law for vector potential and magnetic field [16], given by

$$\boldsymbol{A}[\boldsymbol{r}] = \frac{\mu_0}{4\pi} I \int_C \frac{d\boldsymbol{l}}{|\boldsymbol{r}-\boldsymbol{R}|}, \tag{1}$$

$$\boldsymbol{B}[\boldsymbol{r}] = \frac{\mu_0}{4\pi} I \int_C \frac{(d\boldsymbol{l}\times(\boldsymbol{r}-\boldsymbol{R}))}{|\boldsymbol{r}-\boldsymbol{R}|^3}. \tag{2}$$

This form is apt for expressing the fields due to single loop of wire, but for our purposes, a more useful form is the one with a specified current density $\boldsymbol{J}$,

$$\boldsymbol{A}[\boldsymbol{r}] = \frac{\mu_0}{4\pi} \iiint_\Omega \frac{\boldsymbol{J}dV}{|\boldsymbol{r}-\boldsymbol{R}|}, \tag{3}$$

$$\boldsymbol{B}[\boldsymbol{r}] = \frac{\mu_0}{4\pi} \iiint_\Omega \frac{(\boldsymbol{J}\times(\boldsymbol{r}-\boldsymbol{R}))dV}{|\boldsymbol{r}-\boldsymbol{R}|^3}. \tag{4}$$

The last field that we need to define is the gradient. Its definition follows from the force a magnetic dipole moment $\boldsymbol{m}$ experiences in a non-uniform magnetic field. The potential energy is given by $U = -\boldsymbol{m}\cdot\boldsymbol{B}$. The force and torque are

$$\boldsymbol{F}_{dipole}[\boldsymbol{r}] = \boldsymbol{\nabla}(\boldsymbol{m}\cdot\boldsymbol{B}[\boldsymbol{r}]) = (\boldsymbol{m}\cdot\boldsymbol{\nabla})\boldsymbol{B}[\boldsymbol{r}] = \mathbf{G}[\boldsymbol{r}]\boldsymbol{m}, \tag{5}$$

$$\boldsymbol{\tau}_{dipole}[\boldsymbol{r}] = \boldsymbol{m}\times\boldsymbol{B}[\boldsymbol{r}]. \tag{6}$$

Expression (5) holds for static fields and a fixed moment. The gradient $\mathbf{G}$ is then represented by a symmetric 3x3 matrix

$$\mathbf{G}[\boldsymbol{r}] = \begin{bmatrix} \frac{\partial B_x}{\partial x} & \frac{\partial B_x}{\partial y} & \frac{\partial B_x}{\partial z} \\ \frac{\partial B_y}{\partial x} & \frac{\partial B_y}{\partial y} & \frac{\partial B_y}{\partial z} \\ \frac{\partial B_z}{\partial x} & \frac{\partial B_z}{\partial y} & \frac{\partial B_z}{\partial x} \end{bmatrix} \tag{7}$$

Other approaches strive to reduce the number of integration layers for the sake of reducing computation times and increasing precision, but we are unable to do so to the same extent due to the aforementioned restrictions. To compensate for this, we chose to implement our approach using the highly accurate Gauss-Legendre quadrature while also devising an increment balancing algorithm to better allocate available computational resources to different integration layers.

## III. Calculation approach

### A. Coordinate Transformation

To make the calculation as simple as possible, the loop is positioned in the $xOy$ plane and centered at the origin. The cylindrical coordinate system is used due to the symmetry about the $z$-axis. A coordinate transformation is required to enable arbitrary position and orientation in space. To achieve this, two angles which rotate the normal vector of the plane in which the loop lies are chosen, such that they mimic spherical angles. The respective angles are given as $(\theta, \vartheta)$, $\theta \in [0, \pi]$, $\vartheta \in [0, 2\pi]$.

The basic rotation matrices which rotate the Cartesian basis vectors about a specified axis are $R_x[\theta]$, $R_y[\theta]$, $R_z[\theta]$. An arbitrary rotation can be achieved by multiplying certain arrangements of those 3 matrices with 3 appropriate rotation angles. These angles are Euler angles and there are 12 valid arrangements, usually called conventions. The $Z_1Y_2Z_3$ convention is well suited to our use case because of the loop's symmetry around the $z$-axis. It was concluded that choosing the same angle $\vartheta$ for rotations about $Z_1$ and $Z_3$, and angle $\theta$ for rotation about $Y_2$, produces the desired transformation. It must be noted that the loop will then rotate by $2\vartheta$ if $\theta = 0$,

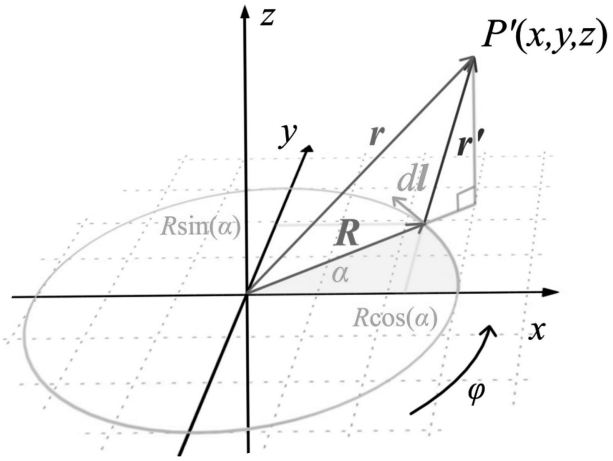



Fig. 1. Biot-Savart law schematic for a loop, after the transformation has been applied, $P'$ is the transformed original point $P$

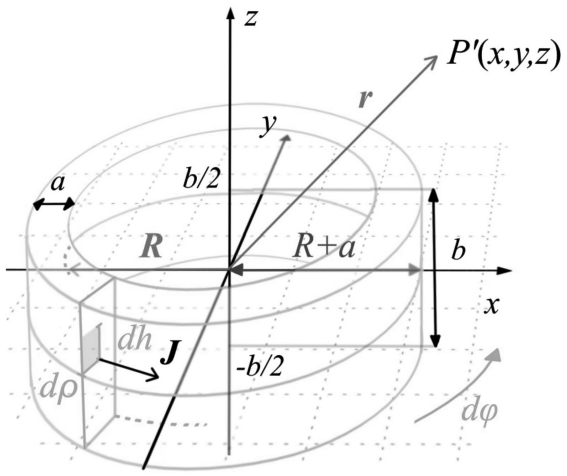



Fig. 2. Biot-Savart law schematic for a thick coil, after the transformation has been applied, $P'$ is the transformed original point $P$

but this is not an issue due to symmetry and is corrected by an inverse transformation. The rotation matrix is defined as

$$\mathbf{T}[\theta,\vartheta]=\begin{bmatrix} c_\theta c_\vartheta^2-s_\vartheta^2 & -c_\theta s_\vartheta c_\vartheta-s_\vartheta c_\vartheta & s_\theta c_\vartheta \\ c_\theta s_\theta c_\vartheta+s_\vartheta c_\vartheta & -c_\theta s_\vartheta^2+c_\vartheta^2 & s_\theta s_\vartheta \\ -s_\theta c_\vartheta & s_\theta s_\vartheta & c_\theta \end{bmatrix}, \\ c_\theta=\cos\theta, s_\theta=\sin\theta, \quad c_\vartheta=\cos\vartheta, s_\vartheta=\sin\vartheta. \tag{8}$$

The coil is positioned at $\boldsymbol{r}_c$. First, the input radius vector $\boldsymbol{r}$ is transformed to $\mathbf{T}[-\theta,-\vartheta](\boldsymbol{r}-\boldsymbol{r}_c)$, and the field is calculated in the transformed coordinate system. Transformation $\mathbf{T}[\theta,\vartheta]$ is then applied to the calculated field. The entire operation, in the case of magnetic field calculation, may be summarised as

$$\boldsymbol{B}_p[\boldsymbol{r}]=\mathbf{T}[\theta,\vartheta]\boldsymbol{B}[\mathbf{T}[-\theta,-\vartheta](\boldsymbol{r}-\boldsymbol{r}_c)]. \tag{9}$$

### *B. Field Expressions*

The problem is now greatly simplified and only the loop in $xOy$ plane needs to be considered. The setup is shown in Figure 1. To use the Biot-Savart law given by (1) and (2), the loop must first be written in parametric form. The position vector is written in cylindrical coordinates $(z,r,\alpha)$

$$\boldsymbol{R}=R\cos(\varphi+\alpha)\boldsymbol{i}+R\sin(\varphi+\alpha)\boldsymbol{j}, \tag{10}$$

$$\boldsymbol{r}=r\cos(\alpha)\boldsymbol{i}+r\sin(\alpha)\boldsymbol{j}+z\boldsymbol{k}. \tag{11}$$

$$d\boldsymbol{l}=\frac{d\boldsymbol{R}}{d\varphi}d\varphi=Rd\varphi(-\sin(\varphi+\alpha)\boldsymbol{i}+\cos(\varphi+\alpha)\boldsymbol{j}), \tag{12}$$

$$D=|\boldsymbol{r}-\boldsymbol{R}|^2=r^2+R^2+z^2-2rR\cos\varphi. \tag{13}$$

Inputting these expressions, the final integrals are obtained as

$$\boldsymbol{A}[\boldsymbol{r}]=\frac{\mu_0}{4\pi}I\int_0^{2\pi}d\varphi\frac{R\cos(\varphi)}{D^{1/2}}\widehat{\boldsymbol{\alpha}}, \tag{14}$$

$$\boldsymbol{B}[\boldsymbol{r}]=\frac{\mu_0}{4\pi}I\int_0^{2\pi}d\varphi\frac{zR\cos(\varphi)\,\widehat{\boldsymbol{r}}+\left(R^2-rR\cos(\varphi)\right)\widehat{\boldsymbol{z}}}{D^{3/2}}. \tag{15}$$

Expressions are given in cylindrical unit vectors as this is the simplest representation. Conversion to Cartesian unit vectors can be performed after calculation. The expression for the gradient is somewhat more complicated and it will be explicitly written only for the thick coil.

In the case of a thick coil, expanding (14) and (15) for varying $R$ and $z$ is the simplest approach. $R$ is replaced by radius $\rho$ of a particular loop, and $h$ is the $z$-axis offset of the loop. The general expressions are then obtained as

$$J=\frac{NI}{ab}, \tag{16}$$

$$D'=r^2+\rho^2+(z+h)^2-2\rho r\cos\varphi, \tag{17}$$

$$\boldsymbol{A}[\boldsymbol{r}]=\frac{\mu_0}{4\pi}J\int_R^{R+a}d\rho\int_{-b/2}^{b/2}dh\int_0^{2\pi}d\varphi\frac{\rho\cos\varphi}{D'^{1/2}}\widehat{\boldsymbol{\alpha}}, \tag{18}$$

$$\boldsymbol{B}[\boldsymbol{r}]=\frac{\mu_0}{4\pi}J\int_R^{R+a}d\rho\int_{-b/2}^{b/2}dh\int_0^{2\pi}d\varphi\frac{K_r\widehat{\boldsymbol{r}}+K_z\widehat{\boldsymbol{z}}}{D'^{3/2}}, \\ K_r=(z+h)\rho\cos\varphi, \quad K_z=\rho^2-\rho r\cos\varphi. \tag{19}$$

The gradient is considered next. Starting from the definition of the gradient matrix (7), and applying the Leibniz integral rule as well as the multi-variable chain rule to (19) yields

$$\mathbf{G}[\boldsymbol{r}]=\begin{bmatrix} P_2c_\alpha^2+P_1s_\alpha^2 & (P_2-P_1)s_\alpha c_\alpha & P_4c_\alpha \\ (P_2-P_1)s_\alpha c_\alpha & P_2s_\alpha^2+P_1c_\alpha^2 & P_4s_\alpha \\ P_4c_\alpha & P_4s_\alpha & P_3 \end{bmatrix}, \\ c_\theta=\cos\theta, s_\theta=\sin\theta, \quad c_\vartheta=\cos\vartheta, s_\vartheta=sin\vartheta, \tag{20}$$

$$P_1[\boldsymbol{r}]=\frac{\mu_0}{4\pi}J\int_R^{R+a}d\rho\int_{-b/2}^{b/2}dh\int_0^{2\pi}d\varphi P_{i1}, \\ P_{i1}=\frac{1}{r}\frac{(z+h)\rho\cos\varphi}{D'^{3/2}}, \tag{21}$$

$$P_2[\boldsymbol{r}]=\frac{\mu_0}{4\pi}J\int_R^{R+a}d\rho\int_{-b/2}^{b/2}dh\int_0^{2\pi}d\varphi P_{i2}, \\ P_{i2}=\frac{3\rho(z+h)(\rho\cos\varphi-r)\cos\varphi}{D'^{5/2}} \tag{22}$$

$$P_3[\boldsymbol{r}] = \frac{\mu_0}{4\pi} J \int_R^{R+a} d\rho \int_{-b/2}^{b/2} dh \int_0^{2\pi} d\varphi P_{i3}, \quad (23)$$

$$P_{i3} = \frac{3\rho\,(z+h)\,(r\cos\varphi - \rho)}{D'^{5/2}},$$

$$P_4[\boldsymbol{r}] = \frac{\mu_0}{4\pi} J \int_R^{R+a} d\rho \int_{-b/2}^{b/2} dh \int_0^{2\pi} d\varphi P_{i4}, \quad (24)$$

$$P_{i4} = \frac{\rho\cos\varphi\left(2\rho^2 + 2r^2 - (z+h)^2 - \rho r\cos\varphi\right) - 3\rho^2 r}{D'^{5/2}}$$

It is important to note that the use of cylindrical coordinates introduces a singular case when $r = 0$. The matrix is then simplified as expressions (21) and (24) become 0, while (22) and (23) are easily integrated. The identity $P_3 = -2P_2$ is obtained, and by using $\boldsymbol{\nabla} \cdot \boldsymbol{B} = 0$, the matrix simplifies to

$$\mathbf{G}_z[\boldsymbol{r}] = \begin{bmatrix} P_2 & 0 & 0 \\ 0 & P_2 & 0 \\ 0 & 0 & P_3 \end{bmatrix}. \quad (25)$$

These expressions are what we will be referring to as slow methods, because there are 3 layers of integration. It was noticed that they can be integrated over length $b$ and consequently a layer of integration can be eliminated. Those will be regarded as fast methods, and are used for field calculations with thin and thick coils. They will only be represented in the form of a sum in the next subsection. Further reducing the number of integration layers introduces many additional terms containing functions beside the square root, lowering performance and increasing complexity.

### C. Gauss-Legendre quadrature

The Gauss-Legendre quadrature [17] is a special case of Gaussian quadrature for definite integrals. When the change of interval is present, it is defined as

$$\int_a^b f[x]dx = \sum_i^n w_{n,i} \frac{b-a}{2} f\left[\frac{b+a}{2} + \frac{b-a}{2} p_{n,i}\right]. \quad (26)$$

The quadrature of order $n$ is defined by special weights $w_{n,i}$ and positions $p_{n,i}$ within the standard interval of integration $[-1, 1]$. Values were obtained from [18] for orders $[1, 100]$, and are implemented as a static matrix for maximum performance.

This particular quadrature is perfect for our use as it converges quickly, offers great precision, and is numerically stable for high orders. Additionally, when arguments are passed to the GPU for computation, they can be stored in arrays and filled up to a certain index. The maximum order for the GPU is set to 80 because the CUDA kernel argument space size is limited to $4096$ bytes at the time of writing. This is exceeded when using double precision with quadrature orders over 80. These limits, while providing substantial performance improvements, can reduce precision. The GPU limit is is not flexible, whereas on the processor, the integration interval can be divided into equal partitions. This allows for increased precision, if it is required, as a shorter integration interval means a quadrature of lower order is needed to reach the maximum theoretical precision of 15 significant digits.

Choosing order 1 for a certain layer effectively removes it. Slow methods are used for flat coils (2 layers) and loops (1 layer), whereas fast methods are used for thick coils (2 layers) and thin coils (1 layer). The points within the integration domain is discretized as

$$\rho_i = R + \frac{a}{2}(1 + p_{n_a,i}), \quad h_j = \frac{b}{2} p_{n_b,j}, \quad \varphi_k = \frac{\pi}{2}(1 + p_{n_\varphi,k}), \quad (27)$$

Figure 3. shows the final expressions for slow methods, and Figure 4. for fast methods.

Fig. 3. Gauss-Legendre sums for slow methods. $P_1$, $P_2$, $P_3$, and $P_4$ are components of matrix (20)

$$D_{i,j,k} = r^2 + {\rho_i}^2 + (z + h_j)^2 - 2\rho_i r\cos\varphi_k, \quad (28)$$

$$\boldsymbol{A}[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{\rho_i \cos\varphi_k}{\sqrt{D_{i,j,k}}} \widehat{\boldsymbol{\alpha}}, \quad (29)$$

$$\boldsymbol{B}[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{(z+h_j)\,\rho_i \cos\varphi_k \widehat{\boldsymbol{r}} + ({\rho_i}^2 - \rho_i r\cos\varphi_k)\widehat{\boldsymbol{z}}}{D_{i,j,k}\sqrt{D_{i,j,k}}}, \quad (30)$$

$$P_1[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{1}{r} \frac{(z+h_j)\,\rho_i \cos\varphi_k}{D_{i,j,k}\sqrt{D_{i,j,k}}}, \quad (31)$$

$$P_2[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{3\rho_i\,(z+h_j)\,(\rho_i \cos\varphi_k - r)\cos\varphi_k}{D_{i,j,k} D_{i,j,k}\sqrt{D_{i,j,k}}}, \quad (32)$$

$$P_3[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{3\rho_i\,(z+h_j)\,(r\cos\varphi_k - \rho_i)}{D_{i,j,k} D_{i,j,k}\sqrt{D_{i,j,k}}}, \quad (33)$$

$$P_4[z,r] = \frac{\mu_0}{2} NI \sum_i^{n_a} \frac{w_{n_a,i}}{2} \sum_j^{n_b} \frac{w_{n_b,j}}{2} \sum_k^{n_\varphi} \frac{w_{n_\varphi,k}}{2} \frac{\rho_i \cos\varphi_k \left(2{\rho_i}^2 + 2r^2 - (z+h_j)^2 - \rho_i r\cos\varphi_k\right) - 3{\rho_i}^2 r}{D_{i,j,k} D_{i,j,k}\sqrt{D_{i,j,k}}}. \quad (34)$$

$$h_t = z + \frac{b}{2}, \qquad h_b = z - \frac{b}{2}, \tag{35}$$
$$D_{t,i,k} = {h_t}^2 + r^2 + {\rho_i}^2 - 2\rho_i r\cos\varphi_k, \qquad D_{b,i,k} = {h_b}^2 + r^2 + {\rho_i}^2 - 2\rho_i r\cos\varphi_k,$$
$$C_{i,k} = r^2 + {\rho_i}^2 - 2\rho_i r\cos\varphi_k,$$
$$C_{1,i,k} = \left({\rho_i}^2 + r^2\right)\cos\varphi_k - 2\rho_i r, \qquad C_{2,i,k} = {\rho_i}^2 - r\rho_i\cos\varphi_k,$$
$$C_{3,i,k} = 2{\rho_i}^2 r^2\cos\varphi_k\left(\cos^2\varphi_k + 2\right) - \rho_i r\left(3\cos^2\varphi_k + 1\right)\left({\rho_i}^2 + r^2\right) + \cos\varphi_k(r^4 + {\rho_i}^4),$$

$$\boldsymbol{A}'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\rho_i\cos\varphi_k\left(\sinh^{-1}\frac{h_t}{\sqrt{C_{i,k}}} - \sinh^{-1}\frac{h_b}{\sqrt{C_{i,k}}}\right)\widehat{\boldsymbol{\alpha}}, \tag{36}$$

$$\boldsymbol{B}'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\left(\left(\frac{\rho_i\cos\varphi_k}{\sqrt{D_{b,i,k}}} - \frac{\rho_i\cos\varphi_k}{\sqrt{D_{t,i,k}}}\right)\widehat{\boldsymbol{r}} + \frac{C_{2,i,k}}{C_{1,i,k}}\left(\frac{1}{{D_{t,i,k}}^{3/2}} - \frac{1}{{D_{b,i,k}}^{3/2}}\right)\widehat{\boldsymbol{z}}\right), \tag{37}$$

$$P_1'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\frac{\rho_i\cos\varphi_k}{r}\left(\frac{1}{\sqrt{D_{b,i,k}}} - \frac{1}{\sqrt{D_{t,i,k}}}\right), \tag{38}$$

$$P_2'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\rho_i\cos\varphi_k C_{2,i,k}\left(\frac{1}{{D_{t,i,k}}^{3/2}} - \frac{1}{{D_{b,i,k}}^{3/2}}\right), \tag{39}$$

$$P_3'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\rho_i C_{2,i,k}\left(\frac{1}{{D_{t,i,k}}^{3/2}} - \frac{1}{{D_{b,i,k}}^{3/2}}\right), \tag{40}$$

$$P_4'[z,r] = \frac{\mu_0}{2}NI\sum_i^{n_a}\frac{w_{n_a,i}}{2}\sum_k^{n_\varphi}\frac{w_{n_\varphi,k}}{2}\frac{\rho_i}{C_{i,k}^2}\left(h_t\left(C_{1,i,k}D_{t,i,k} + \frac{C_{3,i,k}}{D_{t,i,k}^{3/2}}\right) - h_b\left(C_{1,i,k}D_{b,i,k} + \frac{C_{3,i,k}}{D_{b,i,k}^{3/2}}\right)\right). \tag{41}$$

Fig. 4. Gauss-Legendre sums for fast methods. $P_1$, $P_2$, $P_3$, and $P_4$ are components of matrix (20)

### D. Increment balancing

Increment balancing is an important feature of our approach as it allows users to select the number of increments that is used in computation, consequently determining execution time and precision. We define the precision factor $p \in [1.0, 15.0]$. The base number of increments is defined for $p = 1.0$ and increasing $p$ by 1.0 doubles the number of increments.

A relevant dimension is every layer of integration that is not negligible in calculations, or more simply $n_i > 1$. The total number of increments for $k$ relevant dimensions is then

$$n_t = \prod_{i=1}^{k} n_i \approx m^k \cdot 2^{p-1}. \tag{42}$$

The base number of increments per integration layer is denoted as $m$, and we decided to set it to 10. In the case of field calculations, there are at most 2 relevant dimensions: angular increments and thickness increments. The relation is then

$$n_t = n_a n_\phi \approx 10^2 \cdot 2^{p-1}. \tag{43}$$

Increments are assigned by a balancing algorithm which will be discussed in the follow-up paper.

## IV. Performance results

### A. Used Computers

Multiple computer and operating system configurations are considered. In our case, performance is measured in two meaningful ways: real world performance, expressed in number of field computations performed every second (comps/s), and algorithmic performance, which measures how efficiently the algorithm has been implemented. Algorithmic efficiency is expressed in terms of iterations (increments) per second (Inc/s). These two metrics are linked by the precision factor which determines the total number of increments. When calculating with thick and flat coils, algorithms consist of two *for* loops. More and more time is spent in the inner loop as the precision factor becomes larger, increasing the efficiency of the algorithm for higher order quadrature. Differences between operating systems are expected due to different compilers and thread schedulers. Five computers are used:

- Computer A: AMD Ryzen 9 5900HX 8C/16T @4.2GHz, Nvidia RTX 3080 Laptop 16GB 115W, 32GB DDR4-3200 RAM, Windows 10 19044 and Pop!OS 22.04
- Computer B: AMD Threadripper 1950X 16C/32T @4.0GHz, Nvidia RTX 2080 Ti 11GB, 32GB DDR4-3200 RAM, Windows 10 19044
- Computer C: Intel Core i7 7700HQ 4C/8T @3.4GHz, Nvidia GTX 1050 4GB, 16GB DDR4-2400 RAM, Windows 10 build 19044
- Computer D: Intel Core i7 8700K 6C/12T @4.4GHz, Nvidia RTX 2080 Ti 11GB, 32GB DDR4-3000 RAM, Windows 11 22000.739 and Pop!OS 22.04
- Computer E: Intel Core i7-8750H 6C/12T @3.6GHz, Nvidia RTX 2070 Mobile 8GB, 16GB DDR4-2667 RAM, Windows 11 22000.856 and Pop!OS 22.04

Computer A will be tested most extensively as that is a modern laptop (from 2021) and represents performance attainable by most contemporary systems. Computer B is an

older workstation computer with 16 processor cores and a last generation graphics card. Computers C, D, and E all have processors with a nearly identical architecture but with different clock speeds and core counts. GPU performance will only be showcased for Computer A.

### *B. Field performance*

While our testing suggests that using a basic GPU, such as the Nvidia GTX 1050 in Computer C, can greatly improve performance, using double precision removes that advantage on most systems. Hence, we decided to focus on processor performance. All showcased performance graphs can be obtained from functions implemented in module Benchmark, from Python or C++. Python support was implemented with pybind11 [22] to achieve performance similar to native C++ code. The Python module can also be used from MATLAB.

The code works best on Linux with the required GCC compiler. Windows is also well supported but the performance of Microsoft Visual C++ (MSVC) compiler tends to be lower overall, especially when using Python (about 3 times lower in some instances). MacOS support is planned.

Since performance on Linux (more specifically Pop!OS) has been much more consistent, it is used in most of the benchmarks. The computer and operating system are always denoted. Performance benchmarks for slow magnetic field and slow gradient methods are omitted because they were observed to be very similar to slow potential. Performance of different computers and operating systems is shown for one case (4 threads and fast magnetic field). Performance depends on the number of calculations, and 1 000 000 points are chosen for testing. The tested coils have $R = 0.1$ m and $a = 0.4$ m.

From Figures 5-9, we observe that our approach delivers very high performance, especially on Linux. Performance scaling with respect to threads on Windows was unpredictable, fluctuating by as much as 20% between consecutive runs. Scaling on Linux is almost linear, until the number of threads matches the number of cores. When the workload is substantial, using the number of threads equal to the number of hardware threads was observed to yield the best performance. Figure 10 shows that performance on older computers is also good, especially on Linux. Precision factors between 3.0 and 5.0 offer a good balance between precision and performance.

Finally, performance scaling with respect to the number of points is shown in Figure 11 for 1 coil, and in Figure 12 for a large system of 100 coils. Precision factor is set to 3.0. Coil dimensions are $R = 0.1$ m and $a = 0.1$ m.

Multithreading can significantly boost performance, roughly by a factor equal to the number of cores. Points are distributed into similarly sized blocks, and there are as many blocks as there are threads. Efficiency is further improved by using the thread pool [23] approach. For multi-coil systems, coarse-grained multithreading is employed and it significantly increases performance if a large number of cores are utilised.

GPU performance is 10 times higher than with multithreading when using 1 coil, and 60 times higher when using 100 coils, exceeding half a billion computations per second. For precision factor 1.0, performance between 1 and 2.5 billion computations per second was observed for all methods.

Fig. 5. Slow potential performance (Computer A Win10)

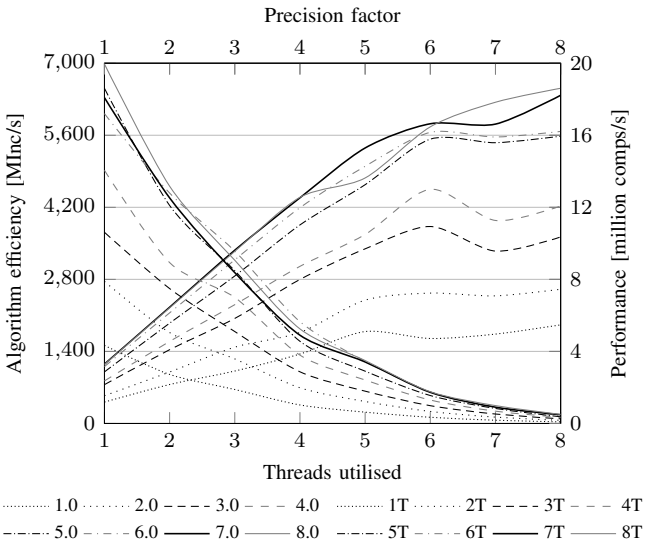



Fig. 6. Slow potential performance (Computer A Pop22.04)

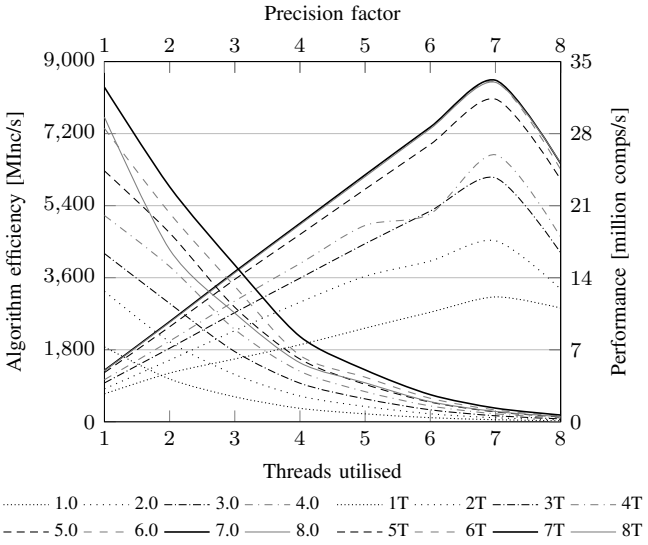



Fig. 7. Fast potential performance (Computer A Pop22.04)

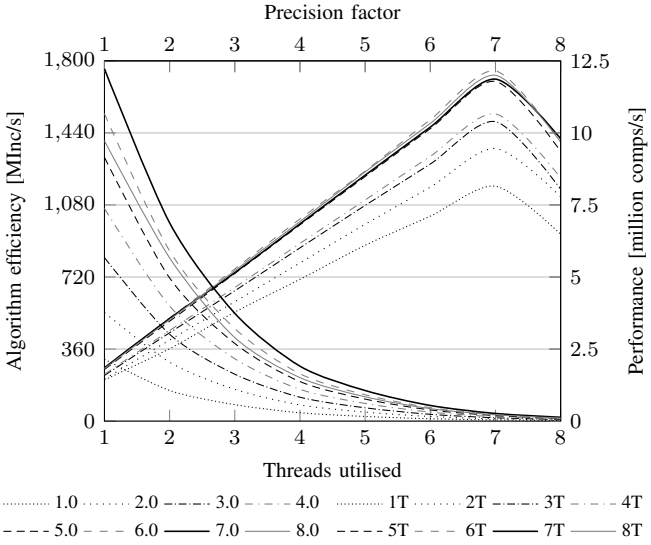


Fig. 8. Fast field performance (Computer A Pop22.04)

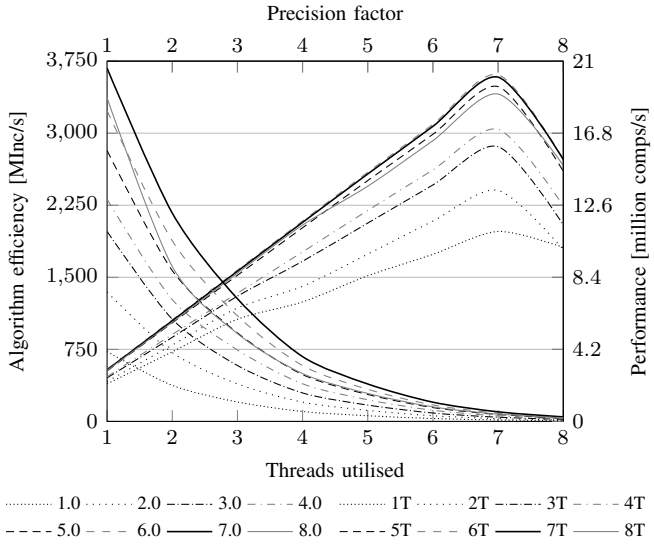



Fig. 9. Fast gradient performance (Computer A Pop22.04)

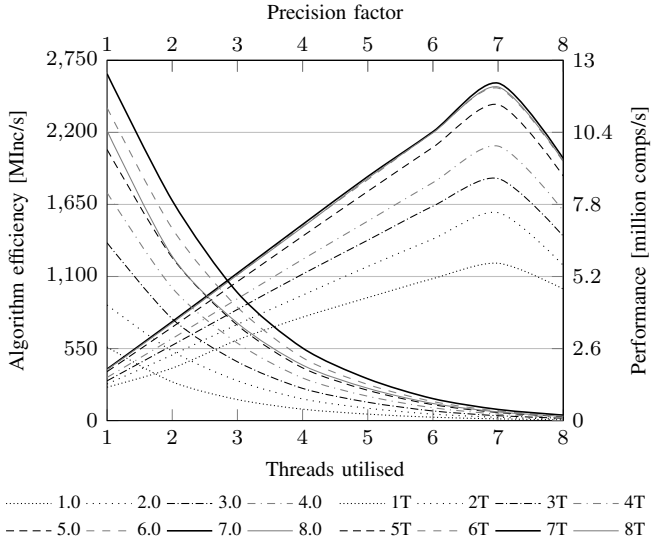



Fig. 10. Comparison of fast field performance (4 threads)

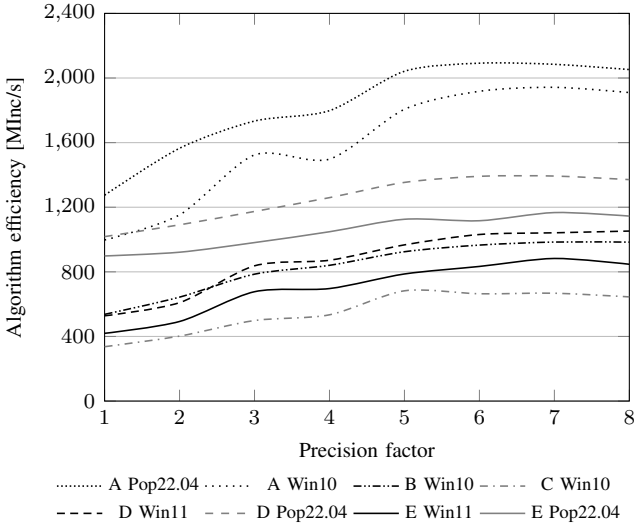



Fig. 11. Performance scaling for fast field, 1 coil (Computer A Win10)

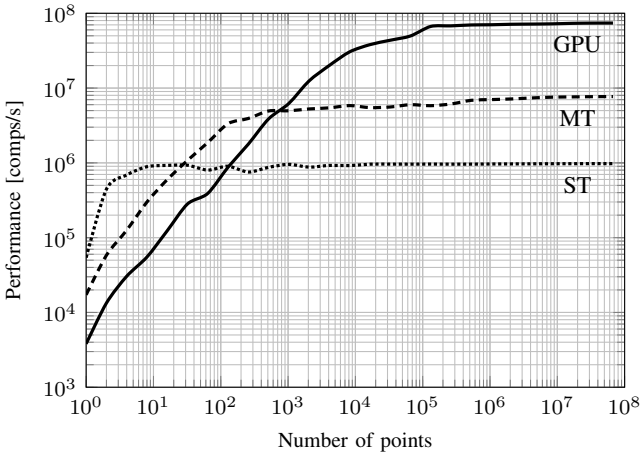
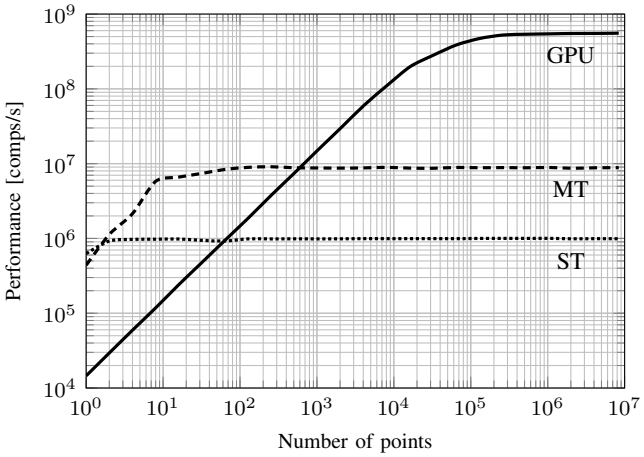



Fig. 12. Performance scaling for fast field, 100 coils (Computer A Win10)




To conclude our testing, Figure 13. shows an image of the magnetic field, generated using [24]. It was compared to an image obtained with Simcenter MAGNET. The field outside of the coil is exact, but artefacts, which look like a series of deliberately arranged thin coils, are found inside.

## V. Conclusion

A performant new approach for calculating the magnetic vector potential, magnetic flux density and magnetic gradient was presented. In contrast to other approaches, it is based on simple expressions which are numerically integrated and efficiently implemented in a compiled programming language. The Gauss-Legendre quadrature was used for numerical integration and its parameters were precomputed. Support for multithreading and hardware accelerated computation are also included and implemented separately for a single coil and a system of coils. Multithreading performance was extensively tested and found to be excellent, especially on Linux. Hardware acceleration significantly increases performance over multithreading, especially in multi-coil systems. Precision and use cases will be evaluated in a follow-up paper. The framework, named C-Coil, is implemented in C++ and CUDA, with a supported Python module accessible from MATLAB.

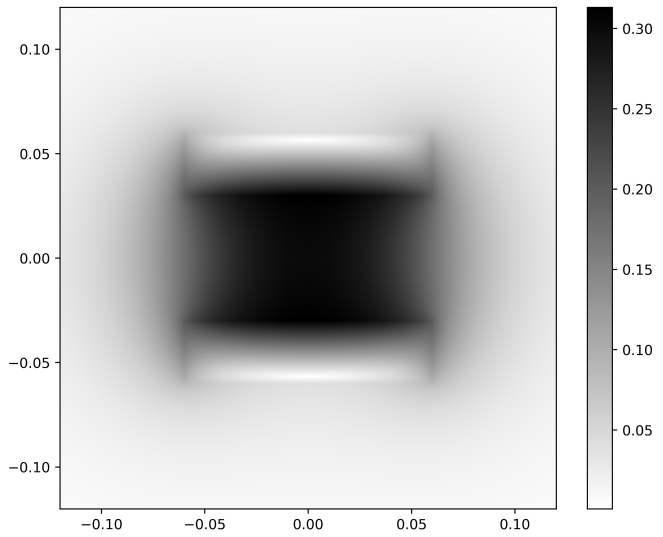



Fig. 13. Absolute value of the magnetic field [T]. Coil specifications are: $R = 0.03$ m, $a = 0.03$ m, $b = 0.12$ m, $N = 3600$, and $I = 10$A.

## ACKNOWLEDGEMENT

The authors would like to thank Prof. Mario Cifrek and Asst. Prof. Juraj Petrović for initial help that enabled us to better set our objectives, Prof. Martin Dadić for helpful consultations and access to Simcenter MAGNET software, and Dr. Branimir Pervan for valuable advice and paper review.

## REFERENCES

[1] S. Babic and C. Akyel, "Improvement in calculation of the self- and mutual inductance of thin-wall solenoids and disk coils," *IEEE Trans. Magn.*, vol. 36, no. 4, pp. 1970-1975, July 2000.

[2] S. Babic, C. Akyel and S. J. Salon, "New procedures for calculating the mutual inductance of the system: filamentary circular coil-massive circular solenoid," *IEEE Trans. Magn.*, vol. 39, no. 3, pp. 1131-1134, May 2003.

[3] S. Babic, C. Akyel and S. J. Salon, "New procedures for calculating the mutual inductance of the system: filamentary circular coil-massive circular solenoid," in *IEEE Trans. Magn.*, vol. 39, no. 3, pp. 1131-1134, May 2003.

[4] J. T. Conway, "Inductance Calculations for Circular Coils of Rectangular Cross Section and Parallel Axes Using Bessel and Struve Functions," *IEEE Trans. Magn.*, vol. 46, no. 1, pp. 75-81, Jan. 2010.

[5] S. Babic, F. Sirois, C. Akyel, G. Lemarquand, V. Lemarquand and R. Ravaud, "New Formulas for Mutual Inductance and Axial Magnetic Force Between a Thin Wall Solenoid and a Thick Circular Coil of Rectangular Cross-Section," *IEEE Trans. Magn.*, vol. 47, no. 8, pp. 2034-2044, Aug. 2011.

[6] S. Babic and C. Akyel, "New Formulas for Mutual Inductance and Axial Magnetic Force Between Magnetically Coupled Coils: Thick Circular Coil of the Rectangular Cross-Section-Thin Disk Coil (Pancake)," *IEEE Trans. Magn.*, vol. 49, no. 2, pp. 860-868, Feb. 2013.

[7] T. Župan, Ž. Štih and B. Trkulja, "Fast and Precise Method for Inductance Calculation of Coaxial Circular Coils With Rectangular Cross Section Using the One-Dimensional Integration of Elementary Functions Applicable to Superconducting Magnets," *IEEE Trans. Appl. Supercond.*, vol. 24, no. 2, pp. 81-89, April 2014, Art no. 4901309.

[8] Y. Luo, X. Wang and X. Zhou, "Inductance Calculations for Circular Coils With Rectangular Cross Section and Parallel Axes Using Inverse Mellin Transform and Generalized Hypergeometric Functions," *IEEE Trans. Power Electron.*, vol. 32, no. 2, pp. 1367-1374, Feb. 2017.

[9] J. T. Conway, "Mutual inductance of thick coils for arbitrary relative orientation and position," *2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL)*, 2017, pp. 1388-1395.

[10] K. Song, J. Feng, R. Zhao, X. Wu, "A general mutual inductance formula for parallel non-coaxial circular coils," in *ACES Journal*, vol. 34, no. 9, pp. 1385-1390, September 2019.

[11] Y. Wang, X. Xie, Y. Zhou and W. Huan, "Calculation and Modeling Analysis of Mutual Inductance Between Coreless Circular Coils With Rectangular Cross Section in Arbitrary Spatial Position," *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 1258-1267

[12] S. I. Babic and C. Akyel, "Magnetic Force Calculation Between Thin Coaxial Circular Coils in Air," *IEEE Trans. Magn.*, vol. 44, no. 4, pp. 445-452, April 2008.

[13] Y. Ren, "Magnetic Force Calculation Between Misaligned Coils for a Superconducting Magnet," *IEEE Trans. Appl. Supercond.*, vol. 20, no. 6, pp. 2350-2353, Dec. 2010.

[14] S. Babic and C. Akyel, "Magnetic Force Between Inclined Circular Filaments Placed in Any Desired Position," *IEEE Trans. Magn.*, vol. 48, no. 1, pp. 69-80, Jan. 2012.

[15] Z. J. Wang and Y. Ren, "Magnetic Force and Torque Calculation Between Circular Coils With Nonparallel Axes," in *IEEE Trans. Appl. Supercond.*, vol. 24, no. 4, pp. 1-5, Aug. 2014, Art no. 4901505.

[16] Feynman, Richard P., Robert B. Leighton, and Matthew L. Sands, "The Feynman Lectures on Physics", Reading, Mass: Addison-Wesley Pub. Co, 1963.

[17] Rainer Kress, "Numerical Analysis," *Springer New York*, 1998, doi: 10.1007/978-1-4612-0599-9

[18] Nodes and Weights of Gauss-Legendre Calculator, accessed on: August 26, 2022, Available: https://keisan.casio.com/exec/system/1280624821

[19] Intel Advanced Vector Extensions Programming Reference, 2011, accessed on: August 26, 2022, Available: https://www.intel.com/content/dam/develop/external/us/en/documents/36945

[20] Nvidia Cuda Toolkit Documentation, accessed on: August 26, 2022, Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[21] R. Dolbeau, "Theoretical peak FLOPS per instruction set: a tutorial", J Supercomput, vol. 74, pp. 1341–1377, 2018, Available: https://doi.org/10.1007/s11227-017-2177-5

[22] Wenzel Jakob, (2016) pybind11 (Version 2.9.2), Source code: https://github.com/pybind/pybind11 (accessed on: August 26, 2022)

[23] Vitaliy Vitsentiy, (2004) CTPL (Version 0.0.2), Source code: https://github.com/vit-vit/CTPL (accessed on: August 26, 2022)

[24] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", *Computing in Science And Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

[25] Davor Dobrota, Nikola Sočec, and Lara Vrabac (2022) C-Coil (Version 1.0.0), Source code: https://github.com/DavorDobrota/C-Coil (accessed on: October 7, 2022)

**Davor Dobrota** was born in Zagreb, Croatia, in 2001.

He is currently an undergraduate student of Electrical Engineering and Information Technology at the University of Zagreb, Zagreb, Croatia. His research interest is the use of modern hardware to accelerate computational electromagnetism, especially magnetostatics.

**Nikola Sočec** was born in Zagreb, Croatia, in 2001.

He is currently an undergraduate student of Computing at the University of Zagreb, Zagreb, Croatia. His interests include software development, high performance computing and artificial intelligence. He works as a part-time software developer at Mindsmiths d.o.o., Zavrtnica 17, Zagreb, Croatia.

**Lara Vrabac** was born in Zagreb, Croatia, in 2001.

She is currently an undergraduate student of Electrical Engineering and Information Technology at the University of Zagreb, Zagreb, Croatia. Her research interests include applied physics and electromagnetism.

**Dario Bojanjac** (M'11) Dario Bojanjac was born in Zagreb, Croatia in 1986. He received dipl.ing. and Ph.D. degree in electrical engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, in 2009 and 2015 respectively and B. S. degree in mathematics from the same university.

From 2009 to 2019 he was a Research and Teaching Assistant at the Department of Wireless Communications, Faculty of Electrical Engineering and Computing, University of Zagreb. Since 2019 he is Assistant Professor at the same department. His research interests include numerical methods for Maxwell's equations, mathematical modelling of electromagnetic scattering problems and homogenization techniques.