

fmmgen: Automatic Code Generation of Operators for Cartesian Fast Multipole and Barnes-Hut Methods

Ryan Pepper

Faculty of Engineering and Physical Sciences, University of Southampton, University Road, Southampton, SO17 1BJ, United Kingdom

Hans Fangohr

Faculty of Engineering and Physical Sciences, University of Southampton, University Road, Southampton, SO17 1BJ, United Kingdom

European XFEL, Holzkoppel 4, 22869 Schenefeld, Germany

arXiv:2005.12351v1 [physics.comp-ph] 25 May 2020

1. Introduction

Implementations of the Fast Multipole Method (FMM) for the Poisson equation usually utilise a spherical harmonic expansion of the Green's function, which leads to an irreducible representation. Doing this in a computationally efficient way is non-trivial because calculating the spherical harmonic functions necessary for such an expansion effectively requires making use of recursion relations in order to avoid numerical issues. Cartesian Taylor expansions also form a straightforward basis in which to expand the potential, though the multipole expansions are reducible and so are less computationally efficient than spherical harmonic expansions for the high accuracy regime [1]. The expansion of the kernel used in different software packages varies widely across different areas of study; in astrophysics a Cartesian basis is primarily used, especially in the Barnes-Hut (BH) method [2, 3], but in chemical molecular dynamics studies it is more usual to make use of the Spherical Harmonic expansion. Similarly, the method utilised varies enormously, and application specific performance optimisations can be made for, which preclude against code-reuse when attempts are made to apply the methods to other problems.

In the past, an approach based on template metaprogramming in C++ was utilised by Visscher and Apalkov [4] to provide efficient recursive implementations of the Cartesian operator functions for point dipole and micromagnetic cell sources. A similar templating approach was used by Wang, et. al. in order to implement Cartesian operators for the FMM as applied to the Boundary Element Method [5]. This approach, however, is not straightforwardly generalisable to languages which do not support code generation at compile time, such as C and Fortran, and it makes it difficult to apply broad optimisations to the templated code. In most FMM and BH implementations based on the Cartesian expansion, hand-written code is therefore used

to implement the various FMM operators; in Dehnen's FalcON code this goes as far as hand implementing vectorisation of the various operators [6]. This is difficult, because beyond several expansion orders it is tedious to ensure code correctness, and the number of terms in the multipole expansion grows with n as $n(n+1)/2$. Implementing operators by hand in this way also makes it difficult to enable code reuse; for example, implementing both the multipolar BH and FMM in the same code base requires a lot of duplication of code.

Generalising to provide efficient operators for point sources of different orders (i.e. point monopoles, point dipoles, point quadrupoles) is also important for adoption of the method. It is important to note that for many problems which are computationally intractable, reduced order models of systems can be constructed using multipoles, by treating objects with complex internal structures as points with a multipole expansion up to some given order, and these sources can often have non-negligible quadrupole terms. [7]

Symbolic code generation is a technique which has, in recent years, been applied to the generation of functions for the computational solution of ordinary and partial differential equations. The FFC library [8] constructs functions for the evaluation of variational forms for assembling finite-element matrices, and is used as part of both the FEniCs and Firedrake projects [9, 10]. The OpenSBLI project [11, 12, 13] generates finite-difference stencils in the language of the high-performance OPS library [14] from symbolic representations of differential equations, while the Devito [15] project achieves similar goals uses symbolic code generation functionality in the SymPy library to generate efficient finite-difference kernels written in C. In the context of the fast multipole method, code generation has previously been utilised by Coles and Masella in order to provide an implementation of the Cartesian basis Fast Multipole Method in the closed source PolarisMD code, for the calculation of the electric potential and field from polarisable atoms in molecular dynamics, [16] with this work then being extended to support the use of more efficient operators through detracing techniques introduced by Applequist. [1, 17, 18, 6]

In this paper, we describe the implementation and details of

Email addresses: ryan.pepper@soton.ac.uk (Ryan Pepper), hans.fangohr@xfel.eu (Hans Fangohr)

an open-source code generation library, fmmgen, [19] which produces a set of operators for the Cartesian BH and FMM methods, and provides OpenMP parallelised example implementations of the methods. We draw attention to how optimisations and simplifications can be enabled at different stages in the code generation to improve performance, and comment on the effectiveness of optimisation strategies. We also discuss the inclusion of the software library in the atomistic spin dynamics software Fidimag [20] to calculate dipolar fields.

2. Mathematical Basis

We begin by showing the mathematical details necessary to construct the FMM and Multipolar BH methods to compute the potential and field the Laplace Equation in a Cartesian basis for source points of arbitrary order (i.e. Monopoles, Dipoles, Quadrupoles, ..., 2^n -poles. We denote the minimum ‘order’ of point sources in a system source as s , such that a monopole has order $s = 0$, a dipole $s = 1$, etc.

We here use the mathematical notation of monomials, which is widely used in the Fast Multipole literature. Here:

$$\begin{aligned} \mathbf{n} &= (n_x, n_y, n_z) \\ \mathbf{n} + \mathbf{m} &= (n_x + m_x, n_y + m_y, n_z + m_z) \\ \mathbf{n}! &= n_x! n_y! n_z! \\ \mathbf{r}^{\mathbf{n}} &= x^{n_x} y^{n_y} z^{n_z} \\ |\mathbf{n}| &= n_x + n_y + n_z \\ \binom{\mathbf{n}}{\mathbf{k}} &= \binom{n_x}{k_x} \binom{n_y}{k_y} \binom{n_z}{k_z} \end{aligned}$$

Consider the expansion of the Coulomb Potential from two well-separated cells A and B , with centres \mathbf{z}_a and \mathbf{z}_b , and containing points \mathbf{x}_a and \mathbf{x}_b respectively. We define vectors $\mathbf{r}_a = \mathbf{x}_a - \mathbf{z}_a$ and $\mathbf{r}_b = \mathbf{x}_b - \mathbf{z}_b$. When a charge q_a is located at \mathbf{x}_a , the potential at \mathbf{x}_b can be evaluated as:

$$\phi(\mathbf{r}) = \frac{q}{|\mathbf{x}_b - \mathbf{x}_a|} \quad (1)$$

Taylor expanding this around the point \mathbf{x}_a and truncating at order p gives an approximate function for the evaluation of the potential:

$$\phi(\mathbf{x}_b - \mathbf{x}_a) \approx q_a \sum_{|\mathbf{n}|=0}^p \frac{(-1)^{\mathbf{n}}}{\mathbf{n}!} (\mathbf{x}_a - \mathbf{z}_a)^{\mathbf{n}} \nabla^{\mathbf{n}} \phi(\mathbf{x}_b - \mathbf{z}_a) \quad (2)$$

By grouping terms, a multipole term defined around the centre \mathbf{z}_A can be written:¹

$$\mathcal{M}_{\mathbf{n}}(\mathbf{z}_A) = \frac{(-1)^{|\mathbf{n}|}}{\mathbf{n}!} q_a (\mathbf{x}_a - \mathbf{z}_a)^{\mathbf{n}} \quad (3)$$

¹This definition varies between fields and authors. Notably, the factor of $(-1)^{\mathbf{n}}/\mathbf{n}!$ is often absorbed into the local expansion definition. It is also worth

For a given $\mathcal{M}_{\mathbf{n}}$ term centred at \mathbf{z}_a , the shifted multipole expansion at a centre \mathbf{z}'_a can be derived through the substitution of $(\mathbf{x}_a - \mathbf{z}_a) = ((\mathbf{x}_a - \mathbf{z}'_a) + (\mathbf{z}'_a - \mathbf{z}_a))$, expanding out in powers and substituting multipole terms where recognised.

$$\mathcal{M}_{\mathbf{n}}(\mathbf{z}'_a) = \sum_{|\mathbf{k}|=0}^{p-|\mathbf{n}|} \frac{(\mathbf{z}_a - \mathbf{z}'_a)^{\mathbf{k}}}{\mathbf{k}!} \mathcal{M}_{\mathbf{n}-\mathbf{k}}(\mathbf{z}_a) \quad (4)$$

Using (4), expressions for calculating the multipole expansion of arbitrary order source particles can be written by considering a ‘source’ multipole $\mathcal{S}_{\mathbf{n}}$. For a Coulomb charge, such that $\mathcal{S}_{(0,0,0)} = q$, and all other terms would be zero. For a dipole, $\mathcal{S}_{(1,0,0)} = \mu_x$, $\mathcal{S}_{(0,1,0)} = \mu_y$ and $\mathcal{S}_{(0,0,1)} = \mu_z$, with all other terms zero. Mixed systems can also be considered. Thus, in an arbitrary system where the lowest order of source is s , the expansion can be written:

$$\mathcal{M}_{\mathbf{n}}(\mathbf{z}) = \sum_{|\mathbf{k}|=0}^{p-|\mathbf{n}|} \frac{(\mathbf{z}_a - \mathbf{x}_a)^{\mathbf{k}}}{\mathbf{k}!} \mathcal{S}_{\mathbf{n}-\mathbf{k}} \quad (5)$$

For the charge only case, we see that we can straightforwardly recover through Eq 3 through the knowledge that all terms except $\mathcal{S}_{(0,0,0)}$ are zero.

The potential can then be rewritten in terms of these Multipole terms. This expression forms the basis of the multipolar Barnes-Hut method.

$$\phi(\mathbf{x}_B - \mathbf{x}_A) \approx \sum_{\mathbf{n}=s}^p \frac{(-1)^{\mathbf{n}}}{\mathbf{n}!} \mathbf{r}_a^{\mathbf{n}} \mathcal{M}_{\mathbf{n}} \nabla^{\mathbf{n}} \phi(\mathbf{x}_B - \mathbf{z}_A) \quad (6)$$

Taking a further expansion, this time around \mathbf{z}_B , and truncating such that the maximum order of terms is the same gives:

$$\phi(\mathbf{x}_B - \mathbf{x}_A) \approx \sum_{\mathbf{n}=s}^p \sum_{\mathbf{m}=0}^{p-|\mathbf{n}|-s} \frac{(-1)^{\mathbf{n}}}{\mathbf{n}! \mathbf{m}!} \mathbf{r}_b^{\mathbf{m}} \mathcal{M}_{\mathbf{n}} \nabla^{\mathbf{n}+\mathbf{m}} \phi(\mathbf{x}_B - \mathbf{z}_A) \quad (7)$$

Grouping terms again in Eq. 7, a local expansion can be evaluated centred around \mathbf{z}_B .

$$\mathcal{L}_{\mathbf{n}}(\mathbf{z}_B) = \sum_{|\mathbf{m}|=0}^{p-|\mathbf{n}|-s} \frac{(-1)^{\mathbf{n}}}{\mathbf{m}!} \mathcal{M}_{\mathbf{m}}(\mathbf{z}_A) \nabla^{\mathbf{n}+\mathbf{m}} \phi(\mathbf{z}_B - \mathbf{z}_A) \quad (8)$$

Then, the potential can be evaluated in terms of the local expansion.

$$\phi(\mathbf{x}_B) \approx \sum_{|\mathbf{n}|=s}^p \frac{1}{\mathbf{n}!} (\mathbf{x}_b - \mathbf{z}_b)^{\mathbf{n}} \mathcal{L}_{\mathbf{n}}(\mathbf{z}_B) \quad (9)$$

Derivatives of the potential then be calculated by differentiating this expression with respect to the component axis:

$$\frac{\partial^{\mathbf{k}} \phi}{\partial \mathbf{r}^{\mathbf{k}}} \approx \sum_{|\mathbf{n}|=s+|\mathbf{k}|}^p \frac{1}{(\mathbf{n}-\mathbf{k})!} (\mathbf{x}_b - \mathbf{z}_b)^{\mathbf{n}-\mathbf{k}} \mathcal{L}_{\mathbf{n}}(\mathbf{z}_B) \quad (10)$$

If the order of the derivative is greater than $p-s$, this expression is not sufficient. In this case, a finite-difference approximation must be used.

3. Implementation

3.1. Operator Generation

Here, we attempt to give a description of the open source code generation framework, `fmmgen`, [19] is implemented and how it can be used. The framework is built in Python, using the symbolic algebra package `SymPy` [21], and generates source code output in C and C++, with the reasoning that code generated in these languages by the framework can be straightforwardly incorporated into other projects without great difficulty or the requirement of large dependencies.

The code generation of each of the multipole operator equations can be broken up into different stages, each of which can be used independently. The user must specify the minimum source order s , the maximum expansion order p , and the output they desire (potential, field, or both). From these parameters, a mapping between n values and one-dimensional array indices is created. By default this mapping is lexicographic, i.e. $((0,0,0), (1,0,0), (0,1,0), (0,0,1), (2,0,0), (1,1,0), \dots)$, such that the total monomial order of a given term is strictly increasing. Nonetheless, if another ordering is preferred (for e.g. in some fields the quadrupole moments are ordered differently), it is possible to use change this by simply using a different array mapping. If it is known in advance that certain terms will always be zero, terms can be removed from the mapping in order to create simpler symbolic representations of the multipole and local expansion operators. We also make use of the source order parameter given by the user to reduce the memory needed to store the multipole and local expansions; this is possible because it is not possible to construct a multipole with a net n^{th} -moment from sources of order $s > n$.

A set of expansion functions are implemented for the Fast Multipole Method, which are used to construct symbolic representations of \mathcal{M}_n , the Particle-to-Multipole (P2M) operator, and \mathcal{L}_n , the Multipole-to-Local operator, at a given n , as well as the shifting operators for these, the Multipole-to-Multipole (M2M) and Local-to-Local (L2L) operators. These functions must make reference to the mapping, in order to return the correct array indices. Generator functions use the set of expansion functions and iterate through the full list of n values needed for a particular problem, and an array representation of each operator is formed. This is repeated for each expansion order, and a least-recently-used (LRU) cache is used in the generation stage to reduce the code generation time. We finally generate a symbolic representation of the operator functions for both the Barnes-Hut and Fast Multipole Method which can calculate the required quantities from a multipole (M2P) or local expansion (L2P), or from another source (P2P).

Once the full set of symbolic operators is generated, a code writing class is used to turn the symbolic representation of the operators into C or C++ code. While the `SymPy` library can provide some basic code-generation functionality, by default it generates unoptimised code which leaves much room for im-

provement in performance terms. To this end, we implemented a set of optimisations which can be enabled and disabled at the code generation stage by the user of the library. We leave these as options rather than enabling by default, because it is then easy to test that the optimisations affect only the performance, and because the optimised code is often more difficult to read and hence debug.

Coles et. al. previously discussed how in code generation of multipole operators, [16] they reduce the number of mathematical operations in the code through Common Subexpression Elimination (CSE), which analyses the code for repeated calculations across multiple lines, and pulls these out as factors. Prior to using CSE, we preprocess the operators to increase the chance of finding common subexpressions. These preprocessing stages rationalise powers (for e.g. replacing $(x^2)^2$ with x^4 , factor terms, and remove extraneous multiplications which sometimes appear in the code generation stage (e.g. $(1.0)x$). In Figure 3, we show the effect that CSE has on the Multipole-to-Local operator.

The optimisations have the greatest effect on performance in the calculation of the Multipole-to-Local operator for the FMM and Multipole-to-Particle operators for the Barnes-Hut method, which make use of the calculation of the derivatives of $1/r$ up to a given order.

In traditional codes, the computations of derivatives of these derivatives up to an order p are usually performed incrementally, such as by using the $O(p^6)$ formula of Cipriani and Silvi [22] or using an $O(p^4)$ recursive formula as described by Challacombe et. al. [23]. With the code-generation, we were able to implement straightforwardly an optimisation noted by Dehnen [6] by making use of the harmonicity of the Poisson Green's function, which allows us to calculate derivatives as:

$$\nabla^{n+(0,0,2)}\phi = -\nabla^{n+(2,0,0)}\phi - \nabla^{n+(0,2,0)}\phi \quad (11)$$

This reduces the number of mathematical operations for higher order calculations. We do note however, that while the `SymPy` library provides some metrics for the number of mathematical operations in given expressions, these are not an effective way of deducing the computational cost of generated code, because the choice of compiler and the enablement of compiler optimisations drastically affects the FLOP count, and because some operations take more clock cycles than others. This means that for accurate FLOP counts, tools such as Intel VTune must be used at runtime.

The code also supports the replacement of evaluations of `pow(x, n)` (or `std::pow(x, n)` in C++), where n is a positive or negative integer value, with multiplication. It is well known that this can be an effective optimisation in numerical codes, but in practice it can be tedious to implement, and beyond a certain point round off errors begin to accumulate. [24] In the code generation stage, these operations can be replaced up to some maximum n_{max} , the optimum which can be determined through profiling for a given architecture, precision and compiler combination.

noting that the definition of the dipole and quadrupole moments can vary; for e.g. in Chemistry the dipole moment vector for a two charge system is normally given as directed from positive to negative charge; in Physics this is reversed.

```

1 void M2L_1(double x, double y, double z, double * M, double * L) {
2     double R = sqrt(x*x + y*y + z*z);
3     double D[4];
4     D[0] = (1 / (R));
5     D[1] = -1.0*x/(R*R*R);
6     D[2] = -1.0*y/(R*R*R);
7     D[3] = -1.0*z/(R*R*R);
8     L[0] += D[0]*M[0] + D[1]*M[1] + D[2]*M[2] + D[3]*M[3];
9     L[1] += D[1]*M[0];
10    L[2] += D[2]*M[0];
11    L[3] += D[3]*M[0];
12 }
13

```

(a) M2L Operator without CSE

```

1 void M2L_1(double x, double y, double z, double * M, double * L) {
2     double Rinv = pow(x*x + y*y + z*z, -0.5);
3     double D[4];
4     double Dtmp0 = (Rinv*Rinv*Rinv);
5     D[0] = Rinv;
6     D[1] = -Dtmp0*x;
7     D[2] = -Dtmp0*y;
8     D[3] = -Dtmp0*z;
9     L[0] += D[0]*M[0] + D[1]*M[1] + D[2]*M[2] + D[3]*M[3];
10    L[1] += D[1]*M[0];
11    L[2] += D[2]*M[0];
12    L[3] += D[3]*M[0];
13 }
14

```

(b) M2L Operator with CSE

Figure 1: Here, we see the affect of enabling common-subexpression elimination for the 1st Order Multipole-to-Local operator in the FMM method when $s = 0$. Prior to enabling this subexpressions such as $1/R^3$ are repeated multiple times across multiple lines of code as in (a), while with it enabled, these are factored out into temporary stack variables as in (b).

3.2. Methods

We implemented both the Barnes-Hut and Fast Multipole Method in a single code base using the generated operator functions. In this, we make use of an octree data structure, whereby the simulation domain is recursively subdivided into octants depending on the particle density, controlled by a parameter n_{crit} , which controls the maximum number of particles in an octant before it is split. The implementation of our octree structure is such that the memory comprising of the multipole and local expansion arrays is contiguous, to allow better cache coherency. Unlike in some codes, we use the cell centre as the expansion centre; while for gravitational systems the centre of mass is an obvious choice as the dipole term in a cell will vanish, for higher order sources and mixed systems, the choice is not so obvious. For the BH method, we evaluate the multipole expansion on cells at the lowest level of the tree, and then pass this upwards using the M2M operators. Then, for each particle, located at \mathbf{x}_p the tree is traversed from the top level downwards. A cell is considered to be near to a particle if it meets the Barnes-Hut multipole acceptance criterion:

$$\frac{r_{\text{cell}}}{|x_p - x_c|} < \theta_{\text{BH}} \quad (12)$$

which relates the cell size to the distance, and an opening angle parameter θ , which is a user supplied parameter which controls the accuracy.

If a cell has no child cells, and the cell does not meet the acceptance criterion, then the cell's particles are looped through, and the interaction is calculated directly using the Particle-to-Particle (P2P) operator. If the criterion is met, then the interaction between the cell and the particle is instead computed using the Multipole-to-Particle (M2P) operator. Finally, if the cell has child cells, then the procedure is repeated on these.

For the FMM, we implemented the dual-tree traversal algorithm which has seen widespread adoption, rather than the classic FMM introduced by Greengard and Rokhlin in which cell-cell interactions only occur between neighbouring cells and their children, [25] as this has much in common with the Barnes-Hut approach. The initial procedure here is the same as the Barnes-Hut method; multipoles are computed for cells on the lowest level of the tree and then shifted upwards. Then, the tree is traversed from top to bottom. Cells which fulfill the multipole acceptance criterion:

$$\frac{r_{c_A} + r_{c_B}}{R} < \theta_{\text{FMM}} \quad (13)$$

interact via the Multipole-to-Local (M2L) operator, while cells which do not are recursed into until either their children fulfill the criteria, or a leaf cell is reached, at which point the cells interact directly. For more straightforward parallelisation, as opposed to the task-based parallelism favoured by some authors, we traverse the tree at initiation in our test implement-

ation, and store the sorted interaction lists which can then be iterated through with loop-based parallelism.

4. Testing

We provide a test application with the library which can be configured to allow the evaluation of the potential and/or field from a set of source particles of arbitrary order, using either the Barnes-Hut or FMM approach, which allows for a straightforward comparison between the two methods and their performance. We ran tests with this text executable on a machine with a 4-core 3.4GHz Intel i7 6700 machine. We note that this processor is affected by the Spectre and Meltdown vulnerabilities, and testing was performed with the Linux kernel version 4.15.0-55-generic, which includes mitigations for this, which have been reported to affect the performance of some HPC applications. [26] The executables were compiled with both the Intel and GNU g++ compilers to allow comparison between the performance.² Tests were performed with OpenMP enabled, and with options set to prevent thread migration between cores and idle threads from sleeping, and with hyperthreading disabled. All of the timing results shown below are averaged over three runs in order to reduce the effect of system calls and background processes on the runtime measurement.

Initially, we tested how the performance optimisations described in the previous section affected the performance of the potential and field calculation via the Fast Multipole Method, for a system of 10^5 randomly distributed charged particles in $[-10^{-9}, 10^{-9}]^3$, with fixed values of $\theta = 0.3$ and $n_{crit} = 128$. We compiled executables for both compilers with generated operators with CSE and the computation of derivatives through the reuse of results and the harmonicity property enabled and disabled, the results of which are shown in Figure 1. We found that in general, the timing results were relatively consistent, with the runtime increasing progressively with the expansion order. With the GNU compiler, enabling the harmonic derivatives optimisation led to a decrease in performance at 9th order of around 50% while enabling CSE led to around a 75% decrease. For the Intel compiler, the corresponding decreases were around 1% and 2%. At lower expansion orders, we see very little performance increase, and this is because there are fewer opportunities for eliminating common factors in expressions. The reason behind the difference between the GNU and Intel compilers was investigated. Analysis with Intel VTune showed that substantial numbers of the repeated operations at high optimisation levels were cached in compilation of the non-CSE enabled code with the Intel Compiler, but not with the GNU compiler. In both cases, there was only a marginal difference in performance when both optimisations were enabled. All subsequent tests to this were run with the Intel executable with both CSE and Harmonic derivatives enabled.

In Figure 2, we show the scaling of the FMM and BH methods with regards to the number of particles, where the number

of particles is chosen such that they are equidistant in log-space. We can see that for numbers of particles up to 10^6 , the BH method outperforms the FMM. In both cases, the exact break-even over the direct method depends on the expansion order, but is less than 1000 charges. We can see that increasing the expansion order gives a clear delineation of the runtime of the Barnes-Hut method while in the FMM, there is less of an impact; this is because the M2P kernel is evaluated many more times in the BH method than the equivalent M2L kernel is in the FMM method, and it is why the method scales more poorly ($O(n \log n)$ for BH vs $O(n)$ for the FMM) at very large numbers of particles.

We note that the two multipole acceptance criterion are not directly equivalent, despite having a similar controlling effect on accuracy, because in the BH it directly relates the particle distance to a cell and its size, while in the FMM it is a cell-cell parameter. As a result of this, to achieve similar error characteristics with the two methods, θ_{FMM} should be around twice θ_{BH} . This can be seen in Figure 3, where we show the error distributions for the two methods at different expansion order at $\theta = 0.25, 0.5$ for a system of 50000 particles with $n_{crit} = 128$. Here, we can see that the

As a real-world test case, we integrated the dipole field FMM calculation generated by the library into the atomistic spin dynamics component of the computational nanomagnetism software Fidimag [27, 20]. To check the implementation, we compared it against the standard technique in this field, which is to use a Fast Fourier Transform (FFT) accelerated convolution in order to sum the field contributions from dipoles placed on a lattice [28, 29, 30]. We constructed a test case comprising of a system of atomic dipoles in a cubic arrangement with n spins on each axis resulting in n^3 spins, and varied n between 5 and 70. We computed the field with the convolution method using the Fast Fourier Transforms computed from the library FFTW with OpenMP parallelism enabled. In order that the comparison was fair, we neglected any start up time which is one-off, and so do not include the pre-computation of the demagnetising tensor for the FFT technique or the tree construction for the FMM technique.

We show the performance results in Fig. 5. In all tests, we found that the FMM method was around an order of magnitude worse in performance terms compared to the FFT convolution technique. We also note that for some θ values ($\theta > 0.7$), in realistic test simulations in which the Landau-Lifshitz-Gilbert equation was used to relax the system, we found that simulations either failed to converge using the FMM, or took more integration steps to do so, as a result of the loss of accuracy in the method. This suggests that, at least on parallel shared memory architectures, using the FMM for dipolar field calculations, at least as implemented here, is not an effective method for lattice simulations.

Despite this, the inclusion of the FMM method into our code Fidimag is designed such that it enables the study of systems where particles do not lie on a lattice, enabling the computation of the dipolar field in problems where it was not previously possible. We note that the atomistic spin dynamics codes Vampire [31] and Vinamax [32] make use of approximations

²The executables were compiled with the Intel Compiler v.19.0.3.199 from Parallel Studio 2019 Update 3 and g++ v.7.5.0.

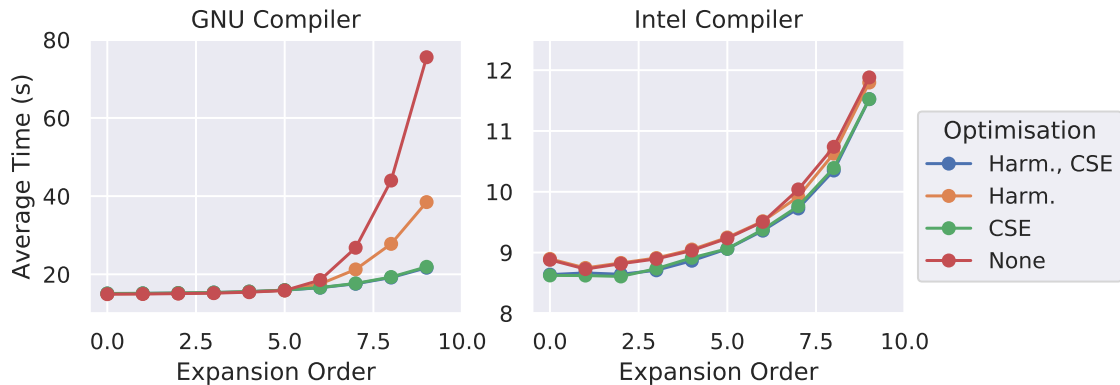


Figure 2: Performance between compilers and with CSE and Harmonic Derivative evaluation disabled and enabled. We see a much greater impact of the code generation optimisations using the GNU g++ compiler than with the Intel Compiler.

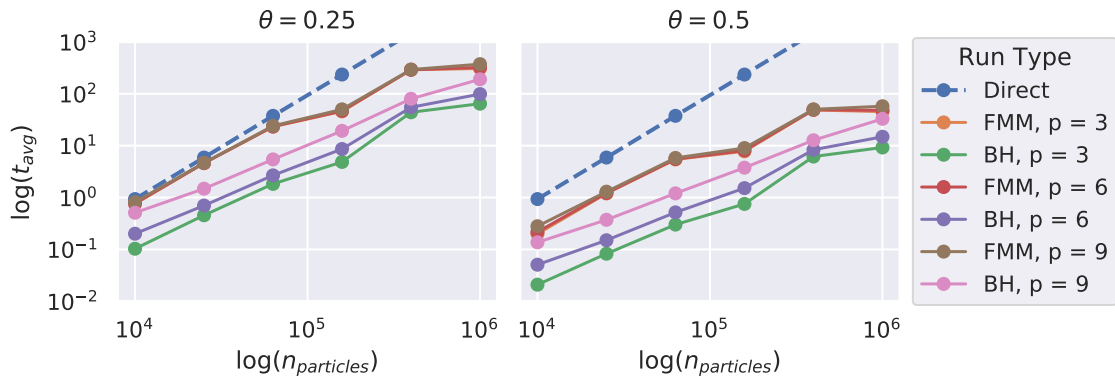


Figure 3: We show how performance varies when varying the opening angle parameter θ and the expansion order p for both the Barnes-Hut and FMM methods. We note that the run time is much less affected by increasing expansion order for the FMM method compared to the Barnes-Hut method.

for computing the dipole field that are similar to the Barnes-Hut method with $p = 1$ and $s = 1$. From our own tests, we found that approximation at this level of expansion order is not sufficient to maintain an acceptable level of accuracy in simulations in general, because it can lead to an error on the dipolar field of over 100% on individual particles - we note that for example, in Fig. 4, we can see that the error distributions have a substantial tail, necessitating the use of high expansion orders to put a reasonable constraint on the error of the field at individual points where the potential is calculated. The effect of such large errors may or may not manifest itself in simulations, and is strongly dependent on other parameters and the relative strength of the dipolar field against other energy terms.

Our results contradict prior performance studies on the fast multipole method in atomistic lattice systems, where the method showed speed-ups over the FFT convolution method for the numbers of particles commonly used in atomistic simulations. We note that the method shown in one paper promising speed-ups from the Cartesian FMM used the scalar non-parallelised FFT routine from Numerical Recipes [33], which was likely to be significantly slower than the FFT methods in FFTW (originally released in 1999, and with the much improved version 3 re-

leased in 2003 which added vectorised forms of the FFT) even at the time of publication [34]. Though we have chosen here to show the results by way of comparison with the FFT in FFTW due to this being freely available across architectures and operating systems, we note that performance of the FFT through the FFTW interface supplied in Intel’s Math Kernel Library was found to be around 2.5x that of FFTW on the same hardware used in this study, and so the FMM fared worse by comparison under these circumstances.

5. Discussion

In this work, we have implemented and shown the efficacy of code generation for the Multipolar Barnes-Hut and Fast Multipole Methods, and have described the implementation of this into a publicly available framework. While we have achieved substantial increases in performance over the direct method, there are several areas in which further progress can be made. Notably, the use of an irreducible representation of the operator functions through the use of a detracing operator can reduce the storage space needed for the Cartesian FMM. [1] In addition, while we have not yet attempted to apply explicit vectorisation

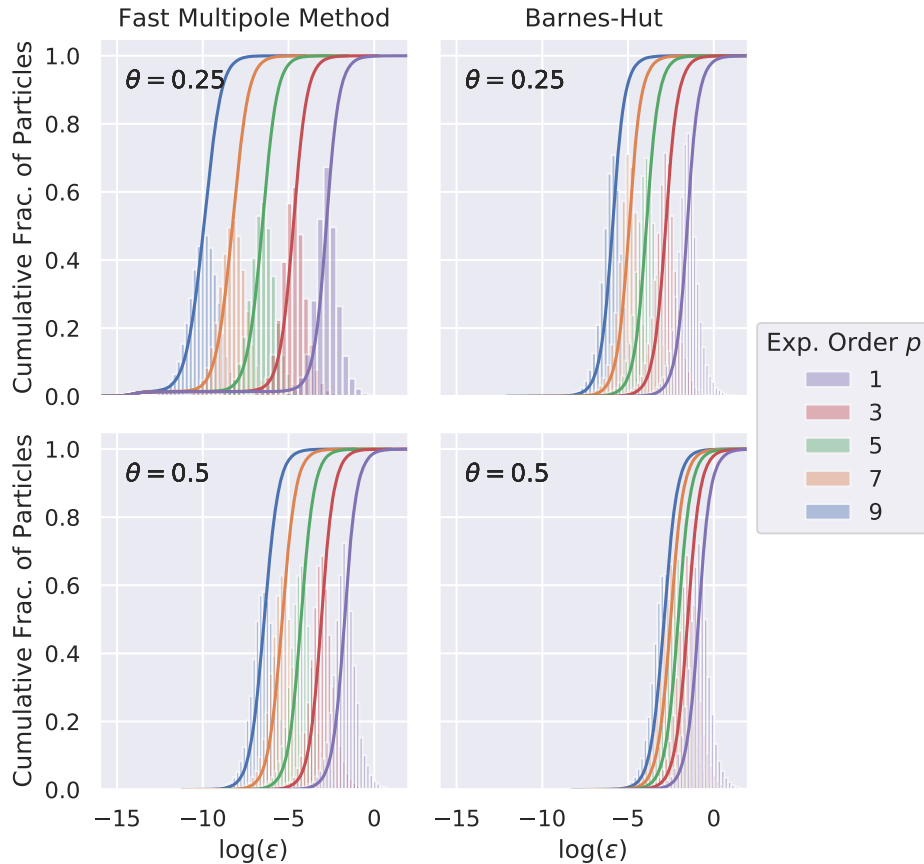


Figure 4: Histogram and cumulative distribution of error for Coloumb potential calculation for the Fast Multipole and Barnes-Hut methods. With increasing expansion order, we see that the median error decreases.

in the code generation stage, the formulation of the code generation system means that doing so across the whole code base should be a more straightforward exercise than a hand written version.

It is important to note that while the algebraic complexity of the spherical harmonic expansion is lower (at $\mathcal{O}(p^4)$ for a naive implementation or $\mathcal{O}(p^3)$ when rotations are used to reduce the cost of the local expansion translation), at low orders it has been shown by various authors that the computational cost of using the Cartesian method is often still lower. It has, however, been shown by the proliferation of consumer-grade GPU hardware in computational research that in many cases, accuracy of less than 10^{-7} is sufficient in many numerical applications. It is with this in mind that there is still much to recommend about the Cartesian approach over the Spherical Harmonics technique.

It is of our opinion that the specialised nature of many fast multipole libraries towards specific problems means that heirarchical methods have not been as successfully adopted as other numerical techniques, and indeed, part of our own motivation for this work was in the difficulty of applying existing packages to our own problems of interest, namely nanomagnetic dipoles. We note that, for example, in gravitational systems where the

domain origin is chosen as the centre of mass, the dipole moment will always vanish. [3, 1]. A specific and widely used optimisation for the fast multipole method in this case, therefore, is to neglect entirely the calculation of the dipole moments in a system, which precludes the reuse of a hand-written gravitational FMM code for other applications where the dipole moments are non-vanishing, without some modification.

6. Data Access

The fmmgen software is hosted at <https://github.com/rpep/fmmgen>, and an archive copy of the version used to perform this study is stored on Zenodo at <https://doi.org/10.5281/zenodo.3842591>. The code, data and scripts are available to reproduce the study and figures and are hosted publicly on Zenodo at <http://doi.org/10.5281/zenodo.3842584>.

7. Acknowledgements

This work was financially supported by EPSRC Doctoral Training Centre Grant EP/L015382/1. We thank J. Coles from

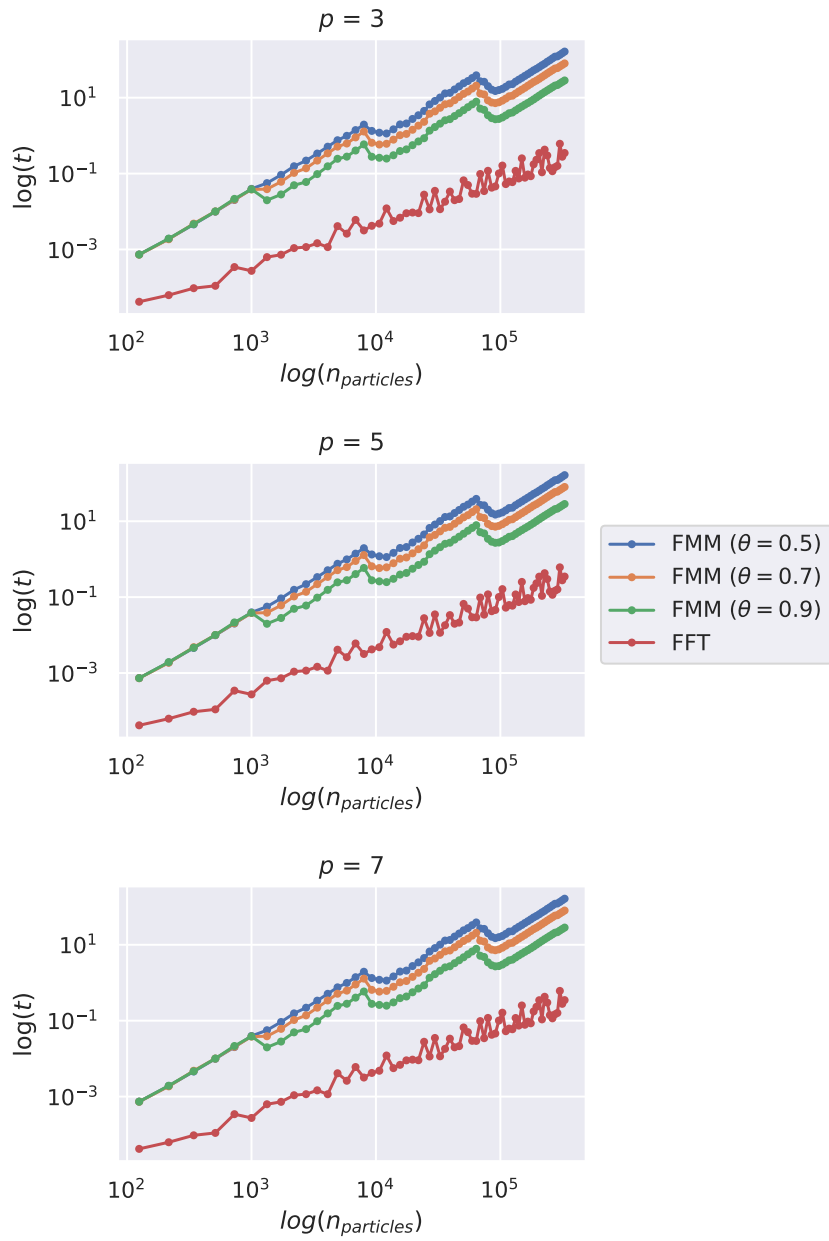


Figure 5: Here we show the performance of the FMM against the FFT for dipolar field calculation, by comparing runtime as the number of spins in a cube is increased at different expansion orders. We note that between expansion orders, we see a negligibly small difference in performance with variation in θ having much more of an impact on runtime.

Technische Universität München for his helpful suggestions on testing the correctness of the implemented operator functions.

8. References

References

- [1] J. P. Coles, R. Bieri, A fully traceless cartesian multipole formulation for the distributed fast multipole method, 2018. URL: <https://arxiv.org/abs/1811.06332>.
- [2] W. Dehnen, A very fast and momentum-conserving tree code, *The Astrophysical Journal* 536 (2000) L39–L42.
- [3] W. Dehnen, A hierarchical (n) force calculation algorithm, *Journal of Computational Physics* 179 (2002) 27–42.
- [4] P. B. Visscher, D. M. Apalkov, Simple recursive implementation of fast multipole method, *Journal of Magnetism and Magnetic Materials* 322 (2010) 275–281.
- [5] T. Wang, S. K. Layton, L. A. Barba, Inexact krylov iterations and relaxation strategies with fast-multipole boundary element method, 2015. URL: <https://arxiv.org/abs/1506.05957>.
- [6] W. Dehnen, A fast multipole method for stellar dynamics, *Computational Astrophysics and Cosmology* 1 (2014).
- [7] Z. V. Gareeva, K. Y. Guslienko, Collective magnetic skyrmion gyrotropic modes in a dot chain, *Journal of Physics Communications* 2 (2018) 035009.
- [8] M. S. Alnæs, A. Logg, K.-A. Mardal, O. Skavhaug, H. P. Langtangen, Unified framework for finite element assembly, *International Journal of Computational Science and Engineering* 4 (2009) 231–244.
- [9] A. Logg, G. N. Wells, DOLFIN, *ACM Transactions on Mathematical Software* 37 (2010) 1–28.
- [10] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, P. H. J. Kelly, Firedrake, *ACM Transactions on Mathematical Software* 43 (2016) 1–27.
- [11] D. J. Lusher, S. P. Jammy, N. D. Sandham, Shock-wave/boundary-layer interactions in the automatic source-code generation framework OpenSBLI, *Computers & Fluids* 173 (2018) 17–21.
- [12] G. Mudalige, I. Reguly, S. Jammy, C. Jacobs, M. Giles, N. Sandham, Large-scale performance of a DSL-based multi-block structured-mesh application for direct numerical simulation, *Journal of Parallel and Distributed Computing* 131 (2019) 130–146.
- [13] C. T. Jacobs, S. P. Jammy, N. D. Sandham, OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures, *Journal of Computational Science* 18 (2017) 12–23.
- [14] I. Z. Reguly, G. R. Mudalige, M. B. Giles, Loop tiling in large-scale stencil codes at run-time with OPS, *IEEE Transactions on Parallel and Distributed Systems* 29 (2018) 873–886.
- [15] F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hüchelheim, C. Yount, P. Witte, P. H. J. Kelly, F. J. Herrmann, G. J. Gorman, Architecture and performance of devito, a system for automated stencil computation, *CoRR* abs/1807.03032 (2018).
- [16] J. P. Coles, M. Masella, The fast multipole method and point dipole moment polarizable force fields, *The Journal of Chemical Physics* 142 (2015) 024109.
- [17] J. Applequist, Fundamental relationships in the theory of electric multipole moments and multipole polarizabilities in static fields, *Chemical Physics* 85 (1984) 279–290.
- [18] J. Applequist, Traceless cartesian tensor forms for spherical harmonic functions: new theorems and applications to electrostatics of dielectric media, *Journal of Physics A: Mathematical and General* 22 (1989) 4303–4330.
- [19] R. A. Pepper, F. H. F., fmmgen, <https://github.com/rpep/fmmgen>, 2019.
- [20] R. Pepper, Weiwei Wang, Marc-Antonio Bisotti, D. I. Cortes, H. Fangohr, T. Kluyver, M. Vousden, M. Beg, R. Carey, O. Laslett, computationalmodelling/fidimag: v3.0, 2020. URL: <https://zenodo.org/record/3841935>. doi:10.5281/ZENODO.3841935.
- [21] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, Sympy: symbolic computing in python, *PeerJ Computer Science* 3 (2017) e103.
- [22] J. Cipriani, B. Silvi, Cartesian expressions for electric multipole moment operators, *Molecular Physics* 45 (1982) 259–272.
- [23] M. Challacombe, E. Schwegler, J. Almlöf, Recurrence relations for calculation of the cartesian multipole tensor, *Chemical Physics Letters* 241 (1995) 67–72.
- [24] B. Wicht, C 11 performance tip: When to use std::pow ?, 2017. URL: <https://baptiste-wicht.com/posts/2017/09/cpp11-performance-tip-when-to-use-std-pow.html>.
- [25] R. Yokota, An FMM based on dual tree traversal for many-core architectures, *Journal of Algorithms & Computational Technology* 7 (2013) 301–324.
- [26] N. A. Simakov, M. D. Innus, M. D. Jones, J. P. White, S. M. Gallo, R. L. DeLeon, T. R. Furlani, Effect of meltdown and spectre patches on the performance of hpc applications, 2018. URL: <https://arxiv.org/abs/arXiv:1801.04329>.
- [27] M.-A. Bisotti, D. Cortés-Ortuño, R. Pepper, W. Wang, M. Beg, T. Kluyver, H. Fangohr, Fidimag – a finite difference atomistic and micromagnetic simulation package, *Journal of Open Research Software* 6 (2018).
- [28] M. Mansuripur, R. Giles, Demagnetizing field computation for dynamic simulation of the magnetization reversal process, *IEEE Transactions on Magnetics* 24 (1988) 2326–2328.
- [29] S. W. Yuan, H. N. Bertram, Fast adaptive algorithms for micromagnetics, *IEEE Transactions on Magnetics* 28 (1992) 2031–2036.
- [30] U. Nowak, Classical Spin Models, in: *Handbook of magnetism and advanced magnetic materials*, Wiley, 2007, pp. 858–876.
- [31] R. F. L. Evans, W. J. Fan, P. Chureemart, T. A. Ostler, M. O. A. Ellis, R. W. Chantrell, Atomistic spin model simulations of magnetic nanomaterials, *Journal of Physics: Condensed Matter* 26 (2014) 103202.
- [32] J. Leliaert, A. Vansteenkiste, A. Coene, L. Dupré, B. V. Waeyenberge, Vinamax: a macrospin simulation tool for magnetic nanoparticles, *Medical & Biological Engineering & Computing* 53 (2015) 309–317.
- [33] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 1992.
- [34] W. Zhang, S. Haas, Adaptation and performance of the Cartesian coordinates fast multipole method for nanomagnetic simulations, *Journal of Magnetism and Magnetic Materials* 321 (2009) 3687–3692.