Advances in Astronomy and Space Physics, **3**, 66-70 (2013)

The discrete dipole approximation code DDscat.C++: features, limitations and plans

*V. Ya. Choliy**

Taras Shevchenko National University of Kyiv, Glushkova ave., 4, 03127, Kyiv, Ukraine

We present a new freely available open-source C++ software for numerical solution of the electromagnetic waves absorption and scattering problems within the Discrete Dipole Approximation paradigm. The code is based upon the famous and free Fortan-90 code DDSCAT by B. Draine and P. Flatau. Started as a teaching project, the presented code DDscat.C++ differs from the parent code DDSCAT with a number of features, essential for C++ but quite seldom in Fortran. This article introduces the new code, explains its features, presents timing information and some plans for further development.

Key words: discrete dipole approximation, light scattering simulations, computer software

INTRODUCTION

Electromagnetic field is scattered or absorbed by targets. It is an isolated grain (of arbitrary geometry and possibly with complex refractive index) or 1-d or 2-d periodic structure of unit cells. According to discrete dipole approximation (DDA) paradigm, the target is approximated with an array of polarizable particles (dipoles). The theory of DDA was proposed by Purcell and Pennypacker [11] and developed by Draine and Flatau [3, 4]. Extension of the theory to periodic structures was made in [5]. Calculations of the electric and magnetic field near the target was introduced in [7].

All mentioned algorithms were implemented in DDSCAT [1] and explained for users in [6]. Current version of DDSCAT is 7.3.0 and here we refer this code as the parent one. Its *User guide* [6] is a necessary and excellent book to start using the code.

The presented code DDscat.C++ [2] is the DDSCAT rewritten in C++. Current version of DDscat.C++ is a clone of the parent code but it contains some C++ specific features to make it easily modifiable and portable. At the beginning the idea was to have a good software for students to study the photonics and IT in a single package. Step-by-step the code has been changed, and now we have the code with another design and architecture, but mostly with the same functionality. That is why we left behind the pages the DDA itself and concentrated on the features and limitations of the new code. *User and programmer guide* [2] of the presented code explains mostly differences between the codes and concentrates on the programming features. The DDscat.C++ users are recommended to read both manuals before start.

Here it is the place to mention another DDA codes. Good but little outdated review is presented in [10]. Among them the Amsterdam DDA code ADD-A [13]: written in C and claimed [9] as an extremely advanced C code for the DDA, SIRRI [8], and ZDD [14]. Only ADD-A and DDSCAT are open source and free. There is yet another OpenDDA framework[3], with an open code accessible via registration on the web-site. Please, refer to Scatterlib[4] for the list of another scattering codes.

FEATURES

The DDscat.C++ uses plug-in paradigm as a main architectural principle. This means that the code is build up with a blocks allowing the lightweight replacement, modification and refactoring. DDscat.C++ is a set of dynamically linked libraries with strictly defined and fixed communication interfaces.

Examples and testing capabilities are an essential part of the code. As running of all tests consumes a lot of time, we do not use CppUnit but our own code and scripts library to be run on request. All DDSCAT tests work fine in DDscat.C++.

Overall view of the architecture and its main

*Choliy.Vasyl@gmail.com

[1] http://code.google.com/p/ddscat/

[2] http://code.google.com/p/ddscatcpp/

[3] http://opendda.org

[4] http://code.google.com/p/scatterlib/

blocks are presented in Fig. 1.

The users familiar with the parent code may easily identify known code blocks. Asterisk as an upper index marks new code parts, introduced in C++ version. Every code portion is controlled with and is communicated via the specially designed manager components. These code snippets are singletons.

Input manager loads parameter file, checks parameters compatibility and prepares the software to run. There are two possibilities to store the parameters. DDscat.C++ may load and understand the parent code *.par files without any changes, but it might be controlled by xml files. Xml is a file format widely used in IT [1] and easily understandable by humans and by computers. Please, refer to [2] for explanation and discussion of the topic. We use open-source third party xml software libXml2[5].

The input manager works together with the restart manager – a code useful in restarting of the calculations, for example, after sudden power failure. It is clearly new feature as the DDscat.C++ restart manager handles the attempts of the code to crash. Strictly speaking, DDscat.C++ never crashes, it is able only to finish gently. This new feature is in the testing phase now.

The target manager manipulates the targets. The target explains the grains geometry or represents elementary cell to build 1-d or 2-d infinite periodic arrays of targets. The parent code contains a lot of different geometries already implemented. These are ellipsoids (spheroids), prisms, cylinders, disks, slabs, tetrahedra, possibly with holes and their simple joints. Some of the targets are just a combination or multiplications of existing ones. There is a CallTarget component in the parent code which helps users to create new targets by specifying the positions and compositions of the target dipoles in the file. The new code CallTarget2 written in wxPython will allow users to create targets interactively.

The solver manager is an interface for linear system solvers. The parent code uses conjugated gradient (CG) method. A lot of CG codes are collected by P. Flatau in Complex Conjugate Gradient Methods library CCGPACK. Current version of the CCGPACK library is 2.0[6]. Very little but useful User guide may be found in the same place. As a parent code the C++ version uses only a little subset of the routines from CCGPACK. Reimplementing of all CCGPACK routines is in our to-do list.

The FFT manager is an interface for Fast Fourier Transform routines. Only two of the parent code possibilities are implemented in C++ version. These are FFTW[7] and Gpfaft, C++ version of the famous Fortran Temperton[8] FFT code [12]. Other possibilities, based upon Intel MKL or Inter Performance Primitives are implemented only as interfaces due to licensing shortcomings.

New linear system solvers (not mandatory CG), FFT routines and targets may be added to the code without rebuilding it. During the initialization of the code, the managers check whether the additional libraries are present and load them if they are. By default target manager and all targets reside in targetlib. New targets and additional target manager code may be put into targetlibpp library for testing. During the code debug it is necessary to rebuild only targetlibpp. When the testing of the new target is finished, the code may be moved to targetlib, and again, to run the code we need to rebuild only targetlib.

The dielectric manager is a little database managing system (DBMS) assigned to manipulate the dielectric data. Those data are normally stored in files with read-only access. It might be a real DBMS, for example, based on Sqlite, with all necessary access codes. It is also possible to keep dielectric data into xml files and use universal accessors like in the Input manager. The data stored in those databases might be accessible from any other applications. The manager can also handle magnetic data.

The parent code was implemented in a parallel mode to be used with MPI and OpenMP. It is able to process different target orientations in parallel but inside the orientation the program works as a sequential one. OpenMP and MPI codes are temporary disabled in C++ version. The feature of the DDscat.C++ is the usage of the CUDA[9] to achieve parallelism inside the elementary task. For that purpose CUDA-based FFT (CUFFT) and linear solver (CUBLAS) are included in DDscat.C++.

DDscat.C++ contains two Readnf executables. The first, Readnf1 should be used to prepare the near-field results for visualization. It is controlled with Readnf1.par (or xml) which consists of the $1^{st}, 2^{nd}$ and 3^{rd} lines of Readnf.par of the parent code. The second one, Readnf2, is used only to make the cross section of the field along the line. User may specify a lot of lines in the Readnf2.par (or xml) file. The code will create as many files as the lines given: one cross line per the output file. DDpostprocess software is identical to the parent code in functionality. All those codes use the same library Processlib.

The usage of CallTarget and CallTarget2 are explained in [6] and [2], correspondingly.

There is only one Windows specific code included into the delivery package, namely the Profiler. The

[5] http://www.xmlsoft.org

[6] http://code.google.com/p/conjugate-gradient-lib/

[7] http://www.fftw.org

[8] possibly our code is the first C/C++ clone of Gpfaft

[9] http://nvidia.com/CUDA/

open source code from the Code Project[10] was principal for code refactoring. The `Profiler` helped us to recognise the code bottlenecks and to direct the refactoring efforts.

QUICK START

`DDscat.C++` code is freely downloadable from the Google code site[11]. The package contains all necessary `*.h` and `*.cpp` files (and does not contain any binary files) to build the software under `Windows XP, Mac OS X`, or `Linux` (checked at `Debian` and `Ubuntu`) operating systems. We succeeded in compiling the code under Raspbian at Raspberry Pi model B[12], but we have very little expertise of the usage of the code there.

The development of `DDscat.C++` was done with `Qt Creator` and `Qt 4.7.4` under `Windows` and `Linux Ubuntu` (two OSes used the same code). The final code then was recombined into `MSVC 7.1` projects with `Intel C++ compiler`[13] and into `Xcode 3.0` projects under `Mac OS X 10.5.8`. All project files for `Qt Designer, MSVC, Xcode` and normal `makefiles` are included into the delivery package. Opening project files and rebuilding the code will result in binary distribution in `BinQt` (for `Qt`), `Bin` (for `MSVC`), `BinX` (for `Xcode`), `BinA` (makefiles). All intermediate files are left in Debug or Release sub-directories inside sub-project directories. The binary distribution contains `DDscat.C++`, `Readnf`, `CallTarget`, `DDpostprocess`, `VTRConvert`, and a list of libraries.

Any third party binary files (for example, `xml` libraries) should reside in an appropriate bin directory to run the code. It is on the user responsibility to download and install them.

The `DDscat.C++` may be controlled with parameter file of the parent code, but some additional freedom in the parameter file is allowed. All string parameters may be presented without putting into apostrophes. Thus, 'GPFAFT', like in the parent code, and just GPFAFT are identical and allowed. Target name may be a single word of any length and free capitalization with all underscore symbols ignored by `DDscat.C++`. That is why `SPH_ANI_N` and `SphaniN` or even `S__p_HAn___iN__` are identical for the Target manager and are allowed.

`DDscat.C++` makes memory allocation only once during target loading. That is why 8^{th} and 9^{th} lines of parameter file are ignored but should be present in the file.

In the definition of composition files after the 13-th line there might be a lot of file names given in the parameter file. `DDscat.C++` allows using the equality sign after some amount of composition files given. It means that all already given file names will be cyclically copied until their amount become equal to `NCOMP`. There are two special file name stubs: `Water` and `Ice` (case insensitive). They represent water and ice dielectric properties, built into the code.

RESULTS

Two new targets `TarNel` and `AniElN` were added to `DDscat.C++` and used here for demonstration purposes. The targets consist of N ellipsoids of identical sizes aligned along the X-axis. The ellipsoids can be anisotropic and touch each other. Anyway, every ellipsoid can have its own composition. In the 'heaviest' case, there should be `3*N` different or equal composition files listed in the parameter file.

The targets have 5 parameters. These are `x-length/d`, `y-length/d`, `z-length/d`, number of ellipsoids, distance between their surfaces along the X-axis. The first three parameters are identical to `ELLIPSO_2`.

Fig. 2 presents the electric field $|\vec{E}|/|\vec{E}_0|$ on two planes, both passing through the centres of three ellipsoids with $a = 0.398\ \mu m$ $24 \times 36 \times 30$. The fifth parameter is equal to 6. The incident wave is propagating with $\vec{k}_0 \parallel \vec{x}_{TF}$ and $\vec{E}_0 \parallel \vec{y}_{TF}$. The geometry is identical to the `ELLIPSOID_NEARFIELD` example of the parent code. Large arrow shows the direction of the X-axis and the incident wave. This figure was generated with `MayaVi2` software. A lot of other figures for different particles are given in the Appendix of the electronic version of the article.

To test the CUDA-based algorithms we used quite old-fashioned self made computer with AMD Phenom 9850, 8 Gb memory, 8 Tb HDD and four GeForce GTX-260 installed on Platinum K9A2 motherboard under `WinXP 64` with `DDscat.C++` 32-bit code compiled with `MSVC 7.1`. General testing of the code was done on the same computer with CUDA disabled.

Table 1: The timing results of the `DDscat.C++` code (seconds) for the `ELLIPSON` example.

Size	Solve	Scat	Nearfield	CUDA Nearfield
2	16.3 - 16.6	9.5	23.3	9.2
3	28.6 - 30.7	14.1	27.7	9.2
5	47.1 - 52.7	23.7	51.0	9.6
8	80.0 - 89.4	38.1	97.2	10.0
12	113.7 - 116.2	56.8	146.2	10.8

The run times of the different `DDscat.C++` code portions for different examples are given in Table 1.

[10] N. Soman, `http://codeproject.com/Simple Profiler using the Visual Studio C C++ Compiler and DIA SDK - CodeProject.html`

[11] `http://code.google.com/p/ddscatcpp/`

[12] `http://www.raspberrypi.org`

[13] thanks a lot to the sponsor who want to stay anonymous

The results are given here only to review the influence of CUDA on routine `Nearfield`. The discussion of the results and deep analysis of the parallelisation technology will be explained with `DDscat.C++` 7.3.1 and in the future papers.

DISCUSSION

There was no idea to make the comparison of the `DDscat.C++` against `DDSCAT`. Anyway if we recall the `DDSCAT` pros and cons from [10] and shortly comment them here, some comparison happens by itself, despite of identical functionality.

The `DDscat.C++` is the most accurate. With the same parameters it converges to a little better precision (20%) with the same or even less amount of iterations, than `DDSCAT`. It is definitively due to different languages, more advanced IT technologies used, and modified architecture.

The `DDscat.C++` is the fastest code, as it is several times faster than the parent code (this statement is quite preliminary, it holds only for our examples with CUDA enabled and only for Nearfield). Fully 'CUDA-fied' `DDscat.C++` should be 50-150 times faster than the sequential parent code for single precision. But this comparison is not quite honest as the `DDscat.C++` and the `DDSCAT` make use of different parallelization schemes. Only different orientations are parallelized in the `DDSCAT` with `MPI`. These are like different tasks. They are absolutely autonomous. In contrary to that the `DDscat.C++` with CUDA parallelizes every numerical algorithm used to its deep.

The `DDscat.C++` is very effective in memory management. Existing code of `DDscat.C++` never keeps unused memory allocated and can use GPU (graphical cards) memory to store the target data. That is why the `DDscat.C++` can manage the targets with greater sizes. We have succeeded with `ELLIPSON` target of 120 ellipsoids, which is impossible with the `DDSCAT`.

The `DDscat.C++` code is written in `C++` and does no need recompiling for different size geometries anymore. Strictly speaking, it does not need recompiling at all as it does not contain any static arrays inside. The code is written not in maniac-style `C++`, but in `C++` with a lot of `C` code in it. That should help users to start using the code. In some places the code really needs a refactoring and polishing the style, and these are our tasks for the next steps.

The `DDscat.C++` is still free with full code available including parallelized parts. `DDscat.C++` will follow the releases of the parent `DDSCAT` code and will always provide the same functionality with additional features clearly stated.

The `DDscat.C++` uses fixed parameter file names but may use any parameter file or even `xml` parameter files. `Xml` is a famous file format readable by humans. There is a lot of software using `xml` as the main input or output format. In any case it does not contradict with the possibility to run a lot of `DDscat.C++` instances at the same time. But from our point of view it is much productive to run them in the sequence: one after one and then post-process the results with the specially designed post-process codes.

The `DDscat.C++` never crashes. In practice it is stopped by restart manager with accident flag `on`. But if it happens, it might be restarted without loss of the results.

The `DDscat.C++` internal design is specially assigned for easy extension and adding some features. We have a lot of plans to make the code very interesting for scientific community. Magnetic dipoles and magnetic properties, surfaces of different geometry near the target, fractal targets, huge targets and non-cubic lattices are the nearest future steps.

In our opinion, the `DDscat.C++` is a good platform to start implementing new scientific tasks on it.

ACKNOWLEDGEMENTS

We would like to thank the `DDSCAT` parent code owners B. Draine and P. Flatau for warm attitude to newborn code, their answers and useful critics of our efforts. Another thanks we would like to express to the students of the Department of Experimental Physics of the Taras Shevchenko National University of Kyiv, especially to S. Gorbyk, who were involved into the project.

REFERENCES

[1] Choliy V. 2011, in *Journees'2011 Proceedings*, 160
[2] Choliy V. 2013 (in preparation)
[3] Draine B. T. 1988, ApJ, 333, 848
[4] Draine B. T. & Flatau P. J. 1994, Journal of the Optical Society of America A, 11, 1491
[5] Draine B. T. & Flatau P. J. 2008, Journal of the Optical Society of America A, 25, 2693
[6] Draine B. T. & Flatau P. J. 2012, [arXiv:1202.3424]
[7] Flatau P. J. & Draine B. T. 2012, Optics Express, 20, 1247
[8] Lumme K. & Rahola J. 1998, J. Quant. Spec. Radiat. Transf., 60, 439
[9] Mc Donald J., Golden A. & Jennings S. G. 2009, [arXiv:0908.0863v1]
[10] Penttilä A., Zubko E., Lumme K. et al. 2007, J. Quant. Spec. Radiat. Transf., 106, 417
[11] Purcell E. M. & Pennypacker C. R. 1973, ApJ, 186, 705
[12] Temperton C. 1992, SIAM Journal on Scientific Computing, 13, 676
[13] Yurkin M. A., Maltsev V. P. & Hoekstra A. G. 2007, J. Quant. Spec. Radiat. Transf., 106, 546
[14] Zubko E., Shkuratov Y., Hart M., Eversole J. & Videen G. 2003, Optics Letters, 28, 1504

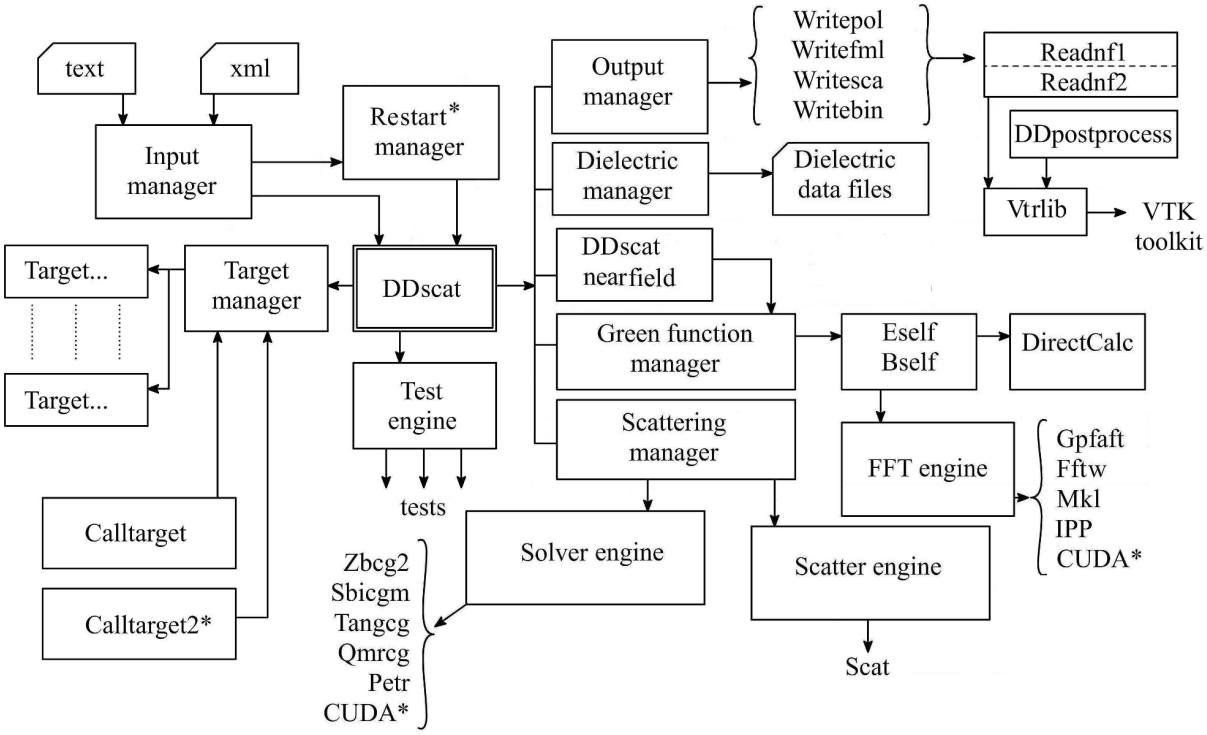

Fig. 1: General view of the code architecture.

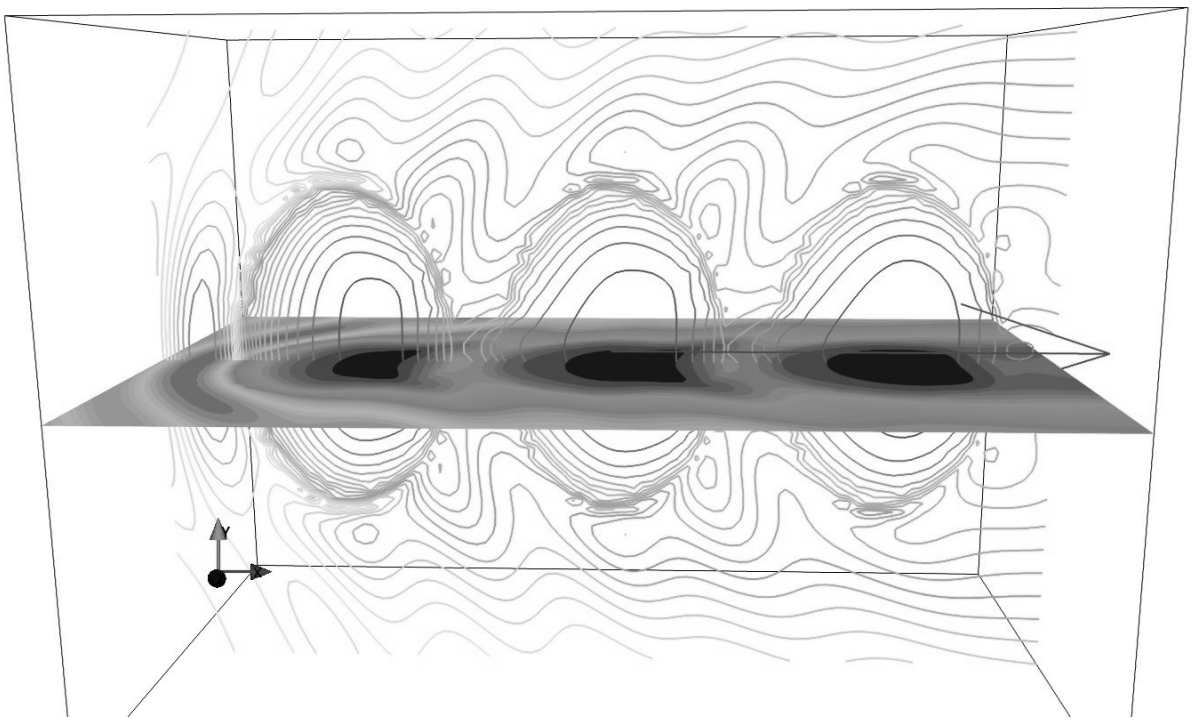

Fig. 2: The electric field around the chain of three ellipsoids.