February 6, 2019

# Fast Multipole Method Manual

## 1 Description

This is a reasonably well-optimized code for our differential algebraic multilevel fully adaptive 3D fast multipole method for the Laplace kernel of point particles in Cartesian basis. The data structures are coded in standard, portable C++ to allow dynamic memory management and bit-level operations, in addition to speed and easier parallelization. The expansions and translations are done in a compact COSYScript code to take advantage of the following:

- DA data types that allow automatic high-order truncated Taylor expansions with machine precision.
- Efficient function composition.
- Fast polynomial evaluation.

## 2 Instructions

The file "fmm_mpi.fox" is the COSY script for running FMM. It may be ran in serial using
`cosy fmm_mpi`
or in parallel using COSY compiled for OpenMPI using
`mpirun -np 4 mpi_cosy.gnu.large fmm_mpi`
Here, it is assumed that the Michigan State University "COSY" program has been built or installed beforehand, and that a file "COSY.bin" has been saved from an earlier execution of the "cosy.fox" script. If not, please go to the COSY Website
(http://www.bt.pa.msu.edu/index_cosy.htm).
The current COSY implementation of FMM requires a C++ program that creates the appropriate data for its (FMM) execution in COSY. The C++ source files for this program can be found inside 3D_fmm.zip within the fmmcpp directory. To compile, first extract the files listed below to a folder and use the makefile provided to compile the program.

> main.cpp, octree.h, particles.h, timers.h, getopt.h, particles.cpp, octree.cpp, timers.cpp, getopt.c

You will need the GNU g++ or Intel C++ compiler installed. Using the included makefile, the program is compiled with GNU g++ or Intel icpc and named "fmmcpp.gnu" or "fmmcpp.intel", respectively. See the header of the makefile for instructions on its options. Because the script "fmm.fox" executes the C++ program, there must be a way to supply the C++ program with the parameters required for its execution. This is done by preparing a file named "fmm.input" that contains the following information:

Line 1: name of output directory for temporary files (must include path delimiter)

Line 2: name of source file

Line 3: name of charge file

Line 4: number of source particles

Line 5: name of target file

Line 6: number of target particles

Line 7: particle limit

Line 8: name of fmmcpp executable file

Line 9: fmm order

Line 10: binary input 0 for asci, 1 for binary, 2 for cosy binary

Line 11: load balancing 1 for on 0 for off

A value of 0 in lines 4 and/or 6 implies taking all particles included in the files from lines 3 and 5.

If the charge values are different from protons, electrons, positrons or antiprotons, the line 3 must be appropriately replaced with the specific charge file. e.g data\charges_1k.dat On successful execution, the results are saved in the output files "fields" and "potentials". The fmm_mpi.summary file includes input parameters and the timing information for each of the main sections of fmm_mpi.fox. Additional run-time information is also saved in several log files.

| | |
|---|---|
| proc_1_1.log | information about the processor number 1 |
| fmmcpp_1.status | indicates whether the data structuring in C++ is successful. For instance, if the value is 0, then the fmm.fox will not be executed. |
| fmmcpp_1.output | contains contents of fmmcpp.input with updated particle limit from each process on line 12. |

# 3 Instructions for the C++ program

The minimum input required for standalone C++ program is position and charge information for charged-particle sources. The current version expects this information to be found in text files: for positions each line contains the Cartesian coordinates of a source/target separated by spaces; for charges one integer per line equal to the source charge value in units of the absolute value of the elementary charge. The syntax for executing the C++ program is as follows: fmmcpp.gnu -s <source_file> [ -t <target_file> ] [ options ]

The other command-line options for the C++ program are as follows:

-p <output_directory, default = tmp/>

-S <number_of_source_particles, default = all>

-T <number_of_source_particles, default = all>

-q <particle_limit, default = 5% of number_of_source_particles>

-o <COSY_order, default = 2>

-N <number_of_processors, default = 1>

-b <option, a=asci, b=binary, c=cosy_binary>

-s <source data file, default=NULL>

-t <target data file, default=NULL>

The output directory is where the data generated by the C++ program are written, and where COSY FMM expects them to be found. The "-S" and "-T" options are useful in processing subsets of source and target particles that are actually available in the named files.

# 4 Particle limit

The particle_limit is calculated based on the particle_limit_fraction in three ways. Hence, it can be set to any desired value as explained below.

1. If the particle_limit_fraction is greater than or equal to 1, the particle limit is the integer part of the particle_limit_fraction. If this particle limit is, however, greater than the number of source particles you will get an error message saying ERROR: particle_limit exceeds the number of sources.

   For example, assume that the number of source particles is 1000. If the particle_limit_fraction is 4.5, then the particle_limit will be 4. If the particle_limit_fraction is 1001.12, then the particle limit, 1001, will exceed the number of source particles and the error message will be displayed.

2. If the particle_limit_fraction is greater than zero (but less than 1), the particle limit is calculated as the product between the number of source particles and the particle_limit_fraction and assigned the integer part of that value to the particle limit. e.g If the particle_limit_fraction is 0.0251 and the number of source particles is 1000 then the particle_limit will be 25.

3. If the particle_limit_fraction is zero (default setting) or negative, the particle limit is calculated as the product between the number of source particles and the DEFAULT_PARTICLE_LIMIT (default value is 0.05) and assigned the integer part of that value to the particle limit. e.g If the particle_limit_fraction is -1 and the number of source particles is 1000 then the particle_limit will be 50.

# 5 Examples

Example 1: This first example is to show the different types of input the fmm may accept. The fmm.input file for this first example is as described in 2. Its contents follow.

```
tmp/
input/positions2000.ssv
input/charges2000.ssv
2000
input/positions2000.ssv
2000
40
./fmmcpp.gnu
2
0
0
```

In this case, the input data is assumed to be in ASCI format. Input data is also provided in binary and cosy binary formats. The fmm.input file must be modified for binary data (left) and cosy binary (right) are below as follows.

```
tmp/
input/positions2000.bin
input/charges2000.bin
2000
input/positions2000.bin
2000
40
./fmmcpp.gnu
2
1
0
```

```
tmp/
input/positions2000_cosy.bin
input/charges2000_cosy.bin
2000
input/positions2000_cosy.bin
2000
40
./fmmcpp.gnu
2
2
0
```

The output in all of these cases should match that of fields_1.txt and potentials_1.txt.

If using windows (as in the next two examples), each forward slash (/) must be replaced by two backslashes (\\) within fmm.input.

Example 2: Use N_1k.ssv for both source particle data and target particle data. This data are normally distributed. The fmm.input file should contain the following parameters.

```
tmp\\
N_1k.ssv
charges_1k.dat
1000
N_1k.ssv
1000
100
fmmcpp.gnu
9
0
0
```

The output should match that of fields_2.txt and potentials_2.txt.

Example 3: Use sourcedata.ssv and targetdata.ssv for source particle and target particle data, respectively. The former data are normally distributed whereas the latter is uniformly distributed. The fmm.input file should contain the following parameters.

```
1  tmp\\
2  sourcedata.ssv
3  charges_2.dat
4  2000
5  targetdata.ssv
6  2401
7  100
8  fmmcpp.gnu
9  9
10 0
11 0
```

The output should match that of fields_3.txt and potentials_3.txt.