pykifmm2d: A 2D Kernel Independent FMM, implemented in python and numba

March 3, 2019

1 What does this package do?

1.1 What does this do now?

This package aims to implement a reasonably efficient, parallel (on a single node), Kernel Independent FMM, with minimal dependencies and a simple to use interface.

The package currently supports:

1. Source to source point FMM
2. Planned source to source point FMM

I have included an option to 'plan' an FMM. This can be useful if an FMM has to be called over and over again on the same geometry, as occurs when solving time-stepping problems on fixed domains, or when using GMRES to invert a BIE. The 'planned' FMM can use a lot of RAM, so it should be used with some amount of care. While it won't make a huge difference for cheap kernels (e.g. Laplace), it can make a significant performance difference for expensive kernels (e.g. Modified Helmholtz).

1.2 What will it do, eventually?

The goal, hopefully, is to support the following:

- Source to target, source to source+target point FMM
- Volume to volume FMMs
- Potentially source+volume to source+volume FMMs

2 Tree Algorithm (point FMM)

The tree algorithm is slightly different than for other FMM codes. In particular, the level restriction is different: in the standard FMM algorithm, level restriction is performed for a leaf if a neighbor node is two or more levels finer than that leaf. In this tree, level restriction is performed if you are a leaf and any of your colleagues have grandchildren. This results in a tree that may be 1 level more refined than the standard tree, but allows for some algorithmic simplifications (as I will explain later in the FMM algorithm - in effect, the downwards pass looks basically the same for every single box in the tree). A high level overview is given in Algorithm 1; more detailed algorithms for the substeps required will be presented in [].

I will go into some more detail about each of these steps.

3 FMM Algorithm

The FMM algorithm is somewhat different from other FMM codes- in particular, I have made a variety of choices to simplify the algorithm, eliminating some of the "lists" that cause complexity in other implementations. I will explain this later, but the primary difference is in the usage of "fake" leaves: refined leaf nodes on which we form multipole expansions in the upward pass, but

Algorithm 1: Construct level-restricted quadtree

Input: x_{input} and y_{input}, float arrays of lengh n, and n_{cutoff}, the cutoff size for leaves
Output: Level-restricted quadtree

copy $x,\ y \leftarrow x_{\text{input}},\ y_{\text{input}}$
$x_{\min},\ x_{\max},\ y_{\min},\ y_{\max} \leftarrow \min x,\ \max x,\ \min y,\ \max y$
$ordv \leftarrow \text{arange}(n)$
`/* this vector will keep track of how we reorder the point vectors` x `and` y `*/`
create initial Level with single box $[x_{\min},\ x_{\max}] \times [y_{\min},\ y_{\max}]$
`/* for future: allow initial level to be multiple boxes to better support anisotropic domains (more important for volume FMM!) */`
while $\max_{\text{node in currLevel}}\{\#node\} > n_{\text{cutoff}}$ **do**
    currLevel ← newLevel from currLevel
    `/* this reorders` x`,` y`, and` $ordv$`! */`
maxDepth ← number of levels
for *level in Levels* **do**
    Tag Colleagues for all nodes in level
    Compute depth of all leaves
Level Restriction
`/* this reorders` x`,` y`, and` $ordv$`! */`
Tag X-list nodes and split into fake nodes
`/* this reorders` x`,` y`, and` $ordv$`! */`
Mark nodes as leaf or not

do not form local expansions in the downward pass. This simplifies the algorithm at the cost of some extra flops, but the simplifications allow for more optimized code (or perhaps I should say, the simplified code is easier to optimize for someone like me who is not really very good at writing code). In particular, this change eliminates the so-called *X-list*.

Another simplifcation that I have made is that I don't attempt to compress the multipole to local translations. Again, this will certainly require some extra flops, but it also allows for simpler code that is easier to optimize for someone who doesn't code so well (like me). I may return to this at some point.

4 Comparisons to other FMM codes