INTRODUCTION TO FAST MULTIPOLE METHODS

LONG CHEN

ABSTRACT. This is a simple introduction to fast multipole methods for the N-body summation problems. Low rank approximation plus hierarchical decomposition leads to fast $\mathcal{O}(N)$ or $\mathcal{O}(N\log N)$ algorithms for the summation problem or equivalently the computation of a matrix-vector product.

We consider a summation problem in the form:

$$u_i = \sum_{j=1}^{N} \phi_{ij} q_j, \quad i = 1, \cdots, N. \tag{1}$$

In the matrix form, if we denote

$$\boldsymbol{u} = (u_i) \in \mathbb{R}^N,\ \boldsymbol{q} = (q_j) \in \mathbb{R}^N,\ \boldsymbol{\Phi} = (\phi_{ij}) \in \mathbb{R}^{N\times N},$$

the sums in (1) are equivalent to the multiplication of matrix $\boldsymbol{\Phi}$ with vector $\boldsymbol{q}$

$$\boldsymbol{u} = \boldsymbol{\Phi}\boldsymbol{q}. \tag{2}$$

A direct evaluation obviously requires $\mathcal{O}(N^2)$ operations. It is unavoidable for general dense matrices. But if the matrix has certain structure, we may calculate the sums in $\mathcal{O}(N\log N)$ or even $\mathcal{O}(N)$ operations.

To see the huge saving of an $\mathcal{O}(N)$ algorithm comparing with an $\mathcal{O}(N^2)$ one when N is large, let us do the following calculation. Suppose $N = 10^6$ and a standard PC can do the summation of 10^6 numbers in 1 minute. Then an $\mathcal{O}(N)$ algorithm will finish the sums (1) in few minutes while an $\mathcal{O}(N^2)$ algorithm will take nearly two years (10^6 minutes $\approx$ 694 days).

1. SEPARABLE AND LOW RANK MATRICES

Consider the following simple example of (1) with

$$\phi_{ij} = (x_j - y_i)^2 = y_i^2 - 2x_j y_i + x_j^2,$$

where $\boldsymbol{x} = (x_j) \in \mathbb{R}^N, \boldsymbol{y} = (y_i) \in \mathbb{R}^N$ are two given vectors. Then, for $i = 1, \cdots, N$

$$u_i = \left(\sum_{j=1}^{N} q_j\right) y_i^2 - 2\left(\sum_{j=1}^{N} q_j x_j\right) y_i + \left(\sum_{j=1}^{N} q_j x_j^2\right) = \alpha y_i^2 - 2\beta y_i + \gamma. \tag{3}$$

The coefficients α, β, and γ do not depend on i and can be computed in $\mathcal{O}(N)$ operations for one pass. Then another loop can compute the summation in $\mathcal{O}(N)$ operations. We list the $\mathcal{O}(N^2)$ naive summation algorithm in `summation1` and an $\mathcal{O}(N)$ algorithm in `summation2`. In `summation2`, the nested for loops are separated. This is a simple example of separable matrices.

```
u = summation1(phi, q)
u = zeros(N,1);
for i = 1:N
    for j = 1:N
        u(i) = u(i) + phi(i,j)*q(j);
    end
end
```

```
u = summation2(phi, q)
u = zeros(N,1);
% phi(i,j) = (x(i)-y(j))^2;
for j = 1:N
    alpha = alpha + q(j);
    beta = beta + q(j)*x(j);
    gamma = gamma + q(j)*x(j)^2;
end
for i = 1:N
    u(i) = alpha*y(i)^2 -2*beta*y(i) + gamma;
end
```

We then introduce the matrix formulation of the summation and consider the generalization. Let $\boldsymbol{x}^2 = (x_1^2, x_2^2, \dots, x_N^2)^T$, $\boldsymbol{y}^2 = (y_1^2, y_2^2, \dots, y_N^2)^T$, and $\boldsymbol{e} = (1, 1, \dots, 1)^T$. We can write the summation (3) in the matrix form

$$\boldsymbol{\Phi} = \boldsymbol{y}^2\boldsymbol{e}^T + \boldsymbol{e}(\boldsymbol{x}^2)^T - 2\boldsymbol{y}\boldsymbol{x}^T. \tag{4}$$

Here for two column vectors $\boldsymbol{a}, \boldsymbol{b}$ of the same length, $\boldsymbol{a}^T\boldsymbol{b}$ is the standard inner product while $\boldsymbol{a}\boldsymbol{b}^T$, called outer product, is a rank-1 matrix. The matrix-vector product can be computed as

$$\boldsymbol{\Phi q} = \boldsymbol{y}^2(\boldsymbol{e}^T\boldsymbol{q}) - 2\boldsymbol{y}(\boldsymbol{x}^T\boldsymbol{q}) + \boldsymbol{e}((\boldsymbol{x}^2)^T\boldsymbol{q}).$$

In general, if the matrix can be written as

$$\boldsymbol{\Phi} = \sum_{k=1}^{p} \boldsymbol{a}_k\boldsymbol{b}_k^T = A_{N\times p}B_{p\times N}, \tag{5}$$

we say $\boldsymbol{\Phi}$ is *separable of order* p. Note that the rank of $\boldsymbol{\Phi}$ is at most p. Every matrix can be written in the form $\sum \phi_{ij}\boldsymbol{e}_i\boldsymbol{e}_j^T$, i.e., separable with order $p = N^2$. We are interested in the case $p \ll N$ and in the order $\mathcal{O}(1)$ or $\mathcal{O}(\log N)$. In this case $\boldsymbol{\Phi}$ is also called a low-rank matrix.

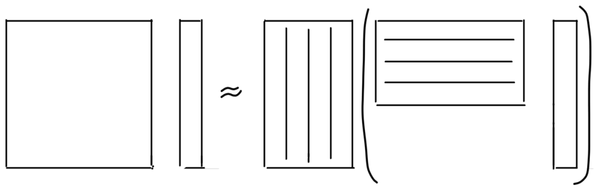

FIGURE 1. Low rank approximation of a matrix can reduce the $\mathcal{O}(N^2)$ matrix-vector multiplication to $\mathcal{O}(pN)$ operations.

For separable matrices with order p, the matrix-vector product can be computed in $\mathcal{O}(pN)$ operations as $\boldsymbol{\Phi q} = A_{N\times p}(B_{p\times N}\boldsymbol{q}_{N\times 1})$. The storage of a separable matrix is also reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(pN)$; see Fig. 1 for an illustration. We even do not have to store the matrix $\mathbf{\Phi}$ when computing $\mathbf{\Phi q}$; in contrast only several vectors are stored, see `summation2` for example.

For a structured matrix depending on only $\mathcal{O}(N)$ parameters $((x_i),(y_j)$ in this example), it may not be always separable. But it might be well approximated by a separable and low rank matrix by exploring the structure of the matrix.

2. Multipole Expansion and Multilevel Aggregation

We chose the following entries in example (1)

$$\phi_{ij} = G(x_j - y_i) = \frac{1}{\|x_j - y_i\|^2}, \tag{6}$$

where $(x_j)_{j=1}^N, (y_i)_{i=1}^N$ are two given set of points.

The sum (1) with kernel (6) may find application in many physical problems. For example, the gravitational potential of N-body or electrostatics of N-point charges. Each x_j, y_i could be points in $\mathbb{R}^n$ for $n = 1, 2, 3$ in practical applications. The points x_j will be called *source* particles and y_i *target* points (or *evaluation* point). The quantity q_j can be thought as the charge of particle x_i in the application of electrostatics or mass of object x_i in gravitation. The kernel $G(x_j - y_i)$ is from the Coulomb's law of the force. The sum $u_i = \sum_j \phi_{ij} q_j$ is the potential at the target point y_i.

In general, $G(s)$ is a scalar or vector field. For a fixed target point y, depending the location of source points, the field $G(x - y)$ for x near y is called *near field* and for x far away from y, it is called *far field*. The 'near' and 'far' will be characterized more rigorously later on. The summation is the combination of near field and far field contribution.

The key idea of fast multipole methods is that *the far-field interaction can be approximated by separable low rank matrices*. The approximation error depends on the distance of sources and targets. Thus a multilevel decomposition and multiple centers are needed. Low rank approximation plus hierarchical decomposition leads to fast $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ algorithms for the summation problem or equivalently the computation of matrix-vector product.

2.1. **Multipole expansion and separable far-filed effect.**

If the source points $\{x_j, j \in T\}$ is far away from y, we may compute the aggregate effect of the charges, not every interaction.

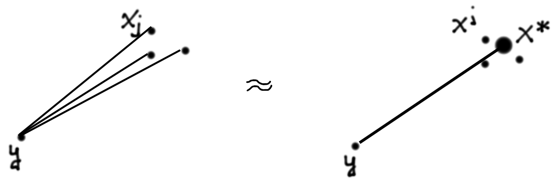

FIGURE 2. The far-field effect from several point charges can be approximated as a single function of the center.

Let x^* be a point near $\{x_j, j \in T\}$ serving as a center of the source in T and y is far away from $\{x_j, j \in T\}$. As Fig 2 shows, the distance $\|x_j - x^*\| \le \delta \|y - x^*\|$ with $\delta < 1$.

We want to approximate the aggregate effect of $\sum_{j\in T} q_j G(x_j - y)$ by a function of $x^* - y$. The basic tool is Taylor series:

$$\begin{aligned}
G(x_j - y) &= G(x_j - x^* + x^* - y) = G((x^* - y)(1 + \frac{x_j - x^*}{x^* - y})) \\
&= G(x^* - y)G(1 + \frac{x_j - x^*}{x^* - y}) \\
&= G(x^* - y)\sum_{m=0}^{p} \frac{G^{(m)}(1)}{m!}\frac{(x_j - x^*)^m}{(x^* - y)^m} + \mathcal{O}(\delta^{p+1}) \\
&= \sum_{m=0}^{p} a_m(x_j - x^*)S_m(x^* - y) + \mathcal{O}(\delta^{p+1}),
\end{aligned}$$

where

$$a_m(x_j - x^*) = \frac{G^{(m)}(1)}{m!}(x_j - x^*)^m, \quad S_m(x^* - y) = \frac{G(x^* - y)}{(x^* - y)^m}.$$

Therefore the points $x_j, j \in T$ and y are separated approximately by the center x^*.

explain the name ‘multipole’ and momentum a_m.

Remark 2.1. In the above Taylor expansion, we assume $G(s)$ is smooth in the neighborhood of $s = 1$ and use the fact $G(st) = G(s)G(t)$. For different fields, e.g. $G(s) = \log s$, we can replace by $G(st) = G(s) + G(t)$ and obtain a similar expansion.

The potential u_i can be approximated by

$$u_i = \sum_{j=1}^{N} G(x_j - y_i)q_j \approx \sum_{m=0}^{p}\left[\sum_{j=1}^{N} q_j a_m(x_j - x^*)\right] S_m(x^* - y_i).$$

As in `summation2`, once x^* is given, the m-th momentum $\sum_j q_j a_m(x_j - x^*)$ can be computed in one pass for $j = 1 : N$ and $m = 1 : p$ in $\mathcal{O}(pN)$ operations. The integer p is chosen such that $\delta^{p+1} \leq \epsilon$ for a given accuracy ϵ. Mathematically, to get $\delta^p \leq \epsilon$, we need $p \geq \log \epsilon / \log \delta \approx 1/(1-\delta)$. Therefore for $i = 1 : N$, the sum can be finished in $\mathcal{O}(pN)$ operations.

2.2. **Multilevel aggregation.** The accuracy of the low rank approximation depends on the choice of the center x^* through the parameter δ which is the ratio of the distance $\|x_j - x^*\|/\|y - x^*\|$. When δ is tiny, we can even chose $m = 0$, i.e., $\sum_j G(x_j - y) \approx G(x^* - y)$. For example, when we calculate the gravitation of a galaxy from the Earth, we can simply treat the galaxy as one single point although the galaxy may contain hundreds of millions of stars.

In our application, the target may not be well separated from the source. Indeed they could be mixed together. We should aggregate the source according to the distance to the target.

To fix the idea, let us assume that all source and target points are in the unit interval $[0, 1]$. We partition $[0, 1]$ into a uniform grid of size h such that each small interval contains only constant number (say, 1 to 5) of source points. Such partition is possible if the source is uniformly distributed in $[0, 1]$. If the distribution is non-uniform, an adaptive grids to equidistributing the number of source in each cell can be used.

Let us further assume the target y_i is in the first cell, i.e., $y_i \in [0, h]$. Obviously for source in the first and second cells, no need of the separation or in other words the low rank approximation is not accurate for near field. We can compute the sum $\sum_{x_j \in [0,2h]} G(x_j - y_i)q_j$ directly which can be done in $\mathcal{O}(1)$ operations since each cell contains constant number of source points.

How about aggregate all other source points into one? That is: define $T = [2h, 1]$ and chose $x^* = 1/2 + h$ as the center of T. Recall that $y_i \in [0, h]$. Therefore the ratio $\delta = (0.5 - h)/0.5 = 1 - 2h$ is very close to 1 and a larger $p \approx 1/h \approx N$ should be chosen and thus no saving on the complexity. We shall show a multilevel centers resulting a near optimal algorithm.

Take $J = \lceil \log_2 N \rceil$ and set $h = 1/2^J$. We choose $T_0 = (0, h], T_1 = (h, 2h], \ldots, T_k = (2^{k-1}h, 2^k h], \ldots T_J = (1/2, 1]$ and translate the sum of N terms to $J = \mathcal{O}(\log N)$ terms:

$$u_i = \sum_{j=1}^{N} q_j G(x_j - y_i) = \sum_{k=0}^{J} \sum_{j \in T_k} q_j G(x_j - y_i) = \sum_{k=1}^{J} R_{T_k}.$$

For each $T_k, k = 0, \ldots, J$, we chose x_k^* as its center; see Fig 3 for an illustration.

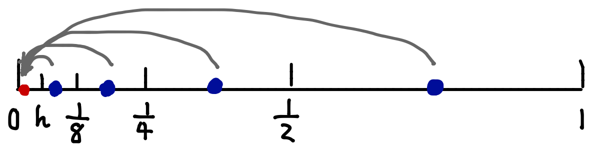

FIGURE 3. According to the distance to the target, the cell size can growth geometrically.

For nearby sources, the near-field effect can be only computed by summation of every interaction, i.e., for $k = 0, 1$

$$R_{T_k} = \sum_{x_j \in T_k} q_j G(x_j - y_i).$$

The far-field function will be approximated by low rank approximation: for $k \geq 2$

$$R_{T_k} = \sum_{x_j \in T_k} q_j G(x_j - y_i) = \sum_{m=0}^{p} \left[\sum_{x_j \in T_k} q_j a_m(x_j - x_k^*) \right] S_m(x_k^* - y_i) + \mathcal{O}(\delta_k^{p+1}).$$

A simple calculation shows

$$\delta_k \leq \frac{1}{3 - (1/2)^{k-2}} \leq \frac{1}{2} \quad \text{for } k \geq 2.$$

The integer p is chosen such that $p = C|\log \varepsilon|$ for a given tolerance ε and thus the tail $\mathcal{O}(\delta^{p+1})$ can be safely skipped in the computation.

We need to compute the coefficients for each group T_k and the cost is $\mathcal{O}(N)$ since every source x_j is visited once. We also need to compute $S_m(x_k^* - y_i)$ for $m = 1 : p$ and $k = 1 : J$. Thus the total cost to compute one u_i is $\mathcal{O}(pJN) = \mathcal{O}(pN \log N)$.

For a different target y_i in a different cell, a different sequence of T_k should be chosen. The length of T_k is geometrically increasing away from y_i such that at most J centers are used in the summation. We can fix several centers and use different far-field approximation for target in different regions. For the p-th order approximation, the sum will take $\mathcal{O}(pJ)$ operations provided the coefficients can be reused. We need an algorithm to compute the coefficients in one pass like the first loop in `summation2`.

3. TREECODE: NEARLY OPTIMAL COMPLEXITY

In this section we present the tree algorithm developed by Barnes and Hut [1]. The complexity is linearithimic, i.e., of order $\mathcal{O}(N\log N)$ which is nearly optimal for a problem of size N.

3.1. **A tree algorithm.** We construct a binary tree of the unit interval as shown in Fig 4. Denoted the cells in level l as $T_{l,k}$ for index $k = 1 : 2^l$. We chose $x^*_{l,k}$ as the center of the cell $T_{l,k}$.

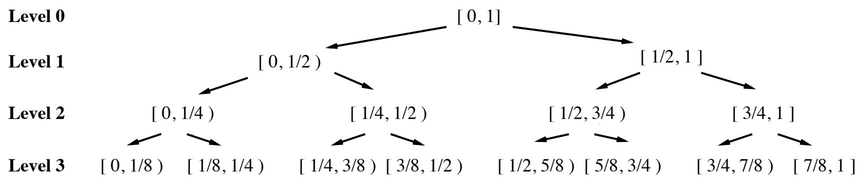

FIGURE 4. Binary tree structure induced by a uniform subdivision of the unit interval. From *A short course on fast multipole methods* by Beatson and Greengard [2].

For each cell $T_{l,k}$ in the tree, compute and store the weight

$$w_{l,k,m} = \sum_{x_j\in T_{l,k}} q_j a_m(x_j - x^*_{l,k}).$$

Each source x_j will be used once in every level and thus the computation of the weight $w_{l,k,m}$ is $\mathcal{O}(N\log N)$. This is like the first loop in `summation2` on computing coefficients.

With the weight computed, for each i, the summation

$$u_i = \sum_{j=1}^{N} G(x_j - y_i)q_j \approx \sum_{l=1}^{J}\sum_{m=0}^{p} w_{l,k(l,i),m} S_m(x^*_{l,k(l,i)} - y_i)$$

can be finished in $\mathcal{O}(Jp)$ operations. Here $k = k(l,i)$ indicates cells used in the summation depends on the target location y_i and the level. In other words, for a given $y_i \in T_{J,i}$, we need to figure out an index path in the tree such that the selected elements along the path will consists of a multilevel partition of $[0,1]$ centered at $T_{J,i}$. For example, for $y_i \in [0,1/8]$, the summation is over $T_{3,2} = [1/8,1/4], T_{2,2} = [1/4,1/2]$, and $T_{1,2} = [1/2,1]$. For $y_i \in [3/8,1/2]$, the summation list is $T_{3,3} = [1/4,3/8], T_{3,5} = [1/2,5/8], T_{3,6} = [5/8,3/4], T_{2,1} = [0,1/4], T_{2,4} = [3/4,1]$. We now discuss a way to find such a path.

3.2. **Interaction list.** For a cell $T_{l,k}$ in level l, its (at most two) neighbors are near-field cells. Other cells in the same level are far-field cells. For each target point in $T_{l,k}$, we need to compute the far-field interaction. But we do not sum over all these far-field cells in the same level. Depending on the distance to $T_{l,k}$, some far-field cells should be computed in the coarser level. (Recall that the panel away from the target can be large.) To this end we define the *interaction list* as the far-field cells in this level only. Mathematically the interaction list of a cell $T_{l,k}$ contains the far-field cells in the same level which are not contained in the far-field of the parent of $T_{l,k}$; or in other words, they are far-field cell of $T_{l,k}$ and near-field of the parent cell of $T_{l,k}$. The summation is only restricted to cells in the interaction list which is at most 4-cells in 1-D; see Fig 5. In one level, for a given cell, the far field consists of well separated cells in this level (collected in the interaction list) and those in coarser levels. The well separated cells in the interaction list can be computed in $\mathcal{O}(1)$ since further cells have been considered in the coarser levels.

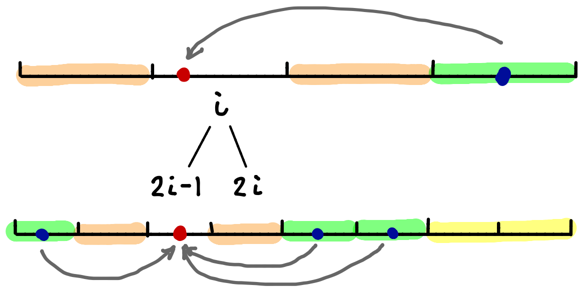

FIGURE 5. Interaction list in a coarse and fine grid. The orange one is the near field cells and the green cells are in interaction list. The yellow cells are far field cells whose parent is also the far-field of the target in the coarse grid.

We list code to find the interaction list for a given index i and explain briefly.

```
1  ic = ceil(i/2); % index in the coarse level
2  nearfieldc = max(ic-1,1):min(ic+1,2^(L-1));
3  activeidx = (2*nearfieldc(1)-1):2*nearfieldc(end);
4  interactionidx = activeidx(activeidx<i-1 | activeidx>i+1);
```

The index map between coarse cells and fine cell is: the i_c coarse cell will be divided into two cells with the index $2i_c - 1, 2i_c$ in the fine grid. Vice verse, the parent of the i-th cell in the fine grid is $i_c = \lceil i/2 \rceil$. Line 1 computes `ic` from a fine index `i`. The near field of i_c-cell will be $\{i_c - 1, i_c + 1\}$ with truncation of the boundary. This is line 2. Refine the near field cells in the coarse grid to get cells in the fine grid, which is called `activdidx` in line 3. In line 4, the near field of i is excluded from the `activeidx` which gives the index of interaction cells of i.

3.3. **Tree code.** We present a pseudo code of the tree algorithm developed by Barnes and Hut [1] below. The implementation details will be left as a project.

```
function fmmtree
J = floor(log2(N));
%% Compute the weight (bottom to top)
for L = J:-1:1
    for k = 1:2^L
        for m = 0:p
            compute w(L,k,m);
        end
    end
end
%% Evaluation (top to bottom)
for L = 1:J
    for k = 1:2^L
        for each y(i) in T(L,k)
```

```
            for m = 0:p % far-field
                for s in the interaction list of T(L,k)
                    u(i) = u(i) + w(L,k,m)*Sm(xstar(L,s) - y(i));
                end
            end
            if L == J % finest level: add near-field
                u(i) = u(i) + near-field(y(i),T(J,:));
            end
        end
    end
end
```

The tree algorithms requires $\mathcal{O}(N\log N)$ operations since every x_j, y_i is visited once in each level and there are $\mathcal{O}(\log N)$ levels. When evaluate the summation, instead of looping over $y_i, i = 1 : N$, we scan each cell in the tree and compute the contribution to each target in this cell. The map from the point index to cell index can by simply calculated by `ci = ceil(x(i)/h)` where h is the length of the cell in the current level. Although we write loops, the computation of weight and the evaluation in each cell does not depending on the ordering and can be implemented in parallel.

4. Fast Multipole Method: Optimal Complexity

In the $\mathcal{O}(N\log N)$ algorithm, we need $J = \log N$ in two places: computing the weight in the multipole expansion in each level and evaluating J far-field terms in the summation. We now discuss an algorithm developed by Greengard and Rokhlin [3] to remove the multiplicity of J and thus obtain an optimal complexity $\mathcal{O}(N)$ algorithm.

The computation of weight is easy to optimize since positive power $(x_j - x^*)^m$ is involved. The far-field evaluation is not easy due to the negative power $(y_i - x^*)^{-m}$. One key idea is to translate the far-field multiple expansion to local Taylor expansion which change the negative power to positive power. Then the local expansion can be easily shifted through the tree.

4.1. **Translation of multipole expansion.**

In the formulation of $a_m(x_j - x^*)$, only the positive power of $(x_j - x^*)^m$ is used which can be easily translated between levels.

Let us take $T_{l-1,1}$ and its two children cells $T_{l,1}$ and $T_{l,2}$. Denote h_l the length of cells in level l. For $j \in T_{l,1}$,

$$
\begin{aligned}
(x_j - x^*_{l-1,1})^m &= (x_j - x^*_{l,1} + x^*_{l,1} - x^*_{l-1,1})^m \\
&= \sum_{s=0}^{m} C_m^s (x^*_{l,1} - x^*_{l-1,1})^{m-s}(x_j - x^*_{l,1})^s \\
&= \sum_{s=0}^{m} C_m^s (-h_l/2)^{m-s}(x_j - x^*_{l,1})^s.
\end{aligned}
$$

A similar expansion for $j \in T_{l,2}$ can be also obtained. So if we store the momentum $a_{l,i,m} = \sum_{j\in T_{l,i}} q_j (x_j - x^*_{l,i})^m$ for $i = 1 : 2^l, m = 0 : p$, we can compute the power in level $l-1$ as

$$\begin{aligned} a_{l-1,1,m} &= \sum_{j\in T_{l-1,1}} q_j (x_j - x^*_{l-1,1})^m \\ &= \sum_{j\in T_{l,1}} q_j (x_j - x^*_{l-1,1})^m + \sum_{j\in T_{l,2}} q_j (x_j - x^*_{l-1,1})^m \\ &= \sum_{s=0}^{m} C_m^s (h_l/2)^{m-s} \left[(-1)^{m-s} a_{l,1,s} + a_{l,2,s}\right]. \end{aligned}$$

That is we can merge the weight of two children to their parent. For readers familiar with multigrid methods, it is like a restriction operator mapping a longer vector $(a_{l,k,m})$ of length pN_l to a shorter vector $(a_{l-1,k,m})$ of length pN_{l-1}.

Now the N points source x_j is only visited in the finest level and in each level the computation cost is $\mathcal{O}(N_l = 2^l)$. Since the sequence N_l is geometrically decay as l decreases, the computation of a_m in all levels will take $\mathcal{O}(p\sum_{l=J}^{1} N_l) = \mathcal{O}(pN)$ operations.

As an explanatory example, we look at the monopole expansion, i.e., only $a_{l,k,0} = \sum_{j\in T_{l,k}} q_j$ is used. Then obviously

$$a_{l-1,1,0} = a_{l,1,0} + a_{l,2,0}.$$

The translation can be implemented by the restriction operator of the piecewise constant function $\boldsymbol{a}_{l-1} = R\boldsymbol{a}_l$ in the terminology of multigrid methods. In general the translation for the multipole expansion is like the restriction operator of high order (p-th order) discontinuous elements. For example for $p = 2$, the local restriction operator R is

$$R = \begin{pmatrix} 1 & 0 & 0 & | & 1 & 0 & 0 \\ -h & 1 & 0 & | & h & 1 & 0 \\ h^2 & -2h & 1 & | & h^2 & 2h & 1 \end{pmatrix}.$$

The $\mathcal{O}(N)$ bottom-to-top algorithm for computing the coefficients $a_{l,k,m}$ is illustrated in the following figure.

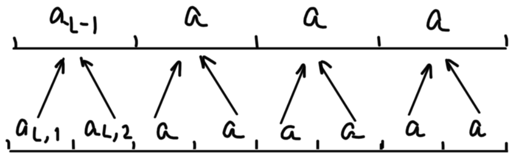

FIGURE 6. An $\mathcal{O}(N)$ bottom-to-top algorithm for computing the coefficients $a_{l,k,m}$

4.2. **Multipole expansion to local expansion.** The $\mathcal{O}(N\log N)$ tree algorithm is achieved by a multilevel partition of all source points. To get an $\mathcal{O}(N)$ algorithm, such technique should be applied to the target points. Again for simplicity, we assume $\{y_i\}$ is also uniform distributed in $[0,1]$ and the same binary tree can be used to partition the targets into multilevel cells.

One $\log N$ complexity is from the evaluation the far-field approximation $S_m(y_i - x^*_{l,k})$ for each y_i at every level. To remove this $\log N$ term, we will defer the evaluation to the finest level only. For a given cell $T_{l,k}$ and a cell $T_{l,s}$ in its interaction list, the evaluation will be replaced by the translation of the multipole expansion at $T_{l,s}$ to a local expansion at the center $y^*_{l,k}$. For one cell, the algorithm will not loop over all y_i in the cell, instead only work with one center; see Fig. 7.

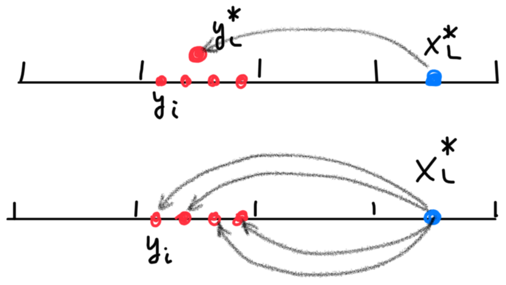

FIGURE 7. Multipole expansion to local expansion

For a cell $T_{l,k}$, we define the multipole potential

$$\Phi_{l,k}(y; x^*_{l,k}, \boldsymbol{a}_{l,k}) = \sum_{m=0}^{p} a_{l,k,m} S_m(x^*_{l,k} - y), \tag{7}$$

and the local expansion

$$\Psi_{l,k}(y; y^*_{l,k}, \boldsymbol{b}_{l,k}) = \sum_{m=0}^{p} b_{l,k,m}(y - y^*_{l,k})^m. \tag{8}$$

Note that these two functions of y are uniquely determined by the coefficients $a_{l,k,m}, b_{l,k,m}$. To translate $\Phi_{l,s}$ to $\Psi_{l,k}$ is to compute the coefficients $\boldsymbol{b}_{l,k}$ using $\boldsymbol{a}_{l,s}$. The computational cost will be thus proportional to the length of $\boldsymbol{a}_{l,k}$ and in each level will be $\mathcal{O}(N_l)$ and the overall cost is $\mathcal{O}(N)$. Once we know $\Psi_{J,k}$ for $k = 1 : 2^J$ in the finest level, the evaluation will be simply

$$u_i \approx \text{ near filed } + \sum_{l=1}^{J} \sum_{(l,k)\in I(l,i)} \Phi_{l,k}(y_i; x^*_{l,k}, \boldsymbol{a}_{l,k}) \approx \text{ near filed } + \Psi_{J,k}(y_i; y^*_{J,k}, \boldsymbol{b}_{J,k}),$$

which can be finished in $\mathcal{O}(N)$ operations.

We now discuss how to translate the multipole potential Φ to the local expansion Ψ. Again the tool is Taylor expansion. Given two well separated cells $T_{l,k}$ and $T_{l,s}$ in the same level, we expand $S_m(x^*_{l,s} - y)$ as a function of $y - y^*_{l,k}$:

$$\begin{aligned} S_m(x^*_{l,s} - y) &= S_m(x^*_{l,s} - y^*_{l,k} + y^*_{l,k} - y) = S_m((x^*_{l,s} - y^*_{l,k})(1 + \frac{y^*_{l,k} - y}{x^*_{l,s} - y^*_{l,k}})) \\ &= S_m(x^*_{l,s} - y^*_{l,k}) S_m(1 - \frac{y - y^*_{l,k}}{x^*_{l,s} - y^*_{l,k}}) \\ &= \sum_{n=0}^{p} \frac{S_m^{(n)}(1)}{n!} \frac{S_m(x^*_{l,s} - y^*_{l,k})}{(x^*_{l,s} - y^*_{l,k})^n} (y - y^*_{l,k})^n + \mathcal{O}(\delta^{p+1}). \end{aligned}$$

Note that in this step an additional approximation error is introduced, i.e.,

$$\Phi_{l,s} \approx \Psi_{l,k} + \mathcal{O}(\delta^{p+1}).$$

From the above formula, we can easily see the relation of $\boldsymbol{a}_{l,s}$ and $\boldsymbol{b}_{l,k}$

$$b_{l,k,m} = \sum_{t=m}^{p} a_{l,s,t} \frac{S_t^{(m)}(1)}{m!} \frac{S_t(x_{l,s}^* - y_{l,k}^*)}{(x_{l,s}^* - y_{l,k}^*)^m}.$$

4.3. **Translation of local expansions.** Unlike the tree code, now the far-field interaction is not accumulated to `u(i)`. Instead the far-field in the interaction list will be translated to the local expansion. To bring the local expansion from the coarse to the fine level, we need a translation of the local expansion. Or equivalently find relation between children's coefficients $\boldsymbol{b}_{l+1,1}, \boldsymbol{b}_{l+1,2}$ from the coefficient $\boldsymbol{b}_{l,1}$ of their parent.

Mathematically we want to express the local expansion

$$\Psi_{l,1}(y; y_{l,1}^*, \boldsymbol{b}_{l,1}) = \Psi_{l+1,i}(y; y_{l+1,i}^*, \boldsymbol{b}_{l+1,i})$$

for $y \in T_{l+1,i}$. The formula is an easy consequence by expanding

$$(y - y_{l,1}^*)^m = (y - y_{l+1,1}^* + y_{l+1,1}^* - y_{l,1}^*)^m = \sum_{s=0}^{m} C_m^s (-h_l/2)^{m-s} (y - y_{l+1,1}^*)^s$$

to get

$$b_{l+1,1,s} = \sum_{m=s}^{p} b_{l,1,m} C_m^s (-h_l/2)^{m-s}, \quad b_{l+1,2,s} = \sum_{m=s}^{p} b_{l,1,m} C_m^s (h_l/2)^{m-s}$$

This step is the prolongation of function $\Psi_{l,k}$ from a coarse level l to a fine level $l+1$ in terms of functions $\Psi_{l+1,k}$. The prolongation operator can be written as $\boldsymbol{b}_{l+1} = R^T \boldsymbol{b}_l$ where R is the restriction operator; see Section 3.1.

As an explanatory example, we look at the monopole expansion, i.e., only $b_{l,k}$ is used in a piecewise constant expansion. Then the translation is simply

$$b_{l+1,1} = b_{l,1}, \quad b_{l+1,2} = b_{l,1},$$

i.e., the prolongation of piecewise constant functions. For $p = 2$, the prolongation operator R^T is

$$R^T = \begin{pmatrix} 1 & -h & h^2 \\ 0 & 1 & -2h \\ 0 & 0 & 1 \\ - & - & - \\ 1 & h & h^2 \\ 0 & 1 & 2h \\ 0 & 0 & 1 \end{pmatrix}.$$

Note that the update of coefficients $b_{l,k}$ includes both the translation of b_{l-1} in the coarse level and the multipole expansion to the local expansion in the same level (for cells in the interaction list); see Fig. 8.

4.4. **Algorithm of optimal complexity.** We present the following $\mathcal{O}(N)$ algorithm.

```
function fmm
J = floor(log2(N));
%% Compute the weight (bottom to top)
for k = 1:2^J  % finest level
    for each x(j) in T(L,k)
        compute a(l,k,m);
    end
end
```

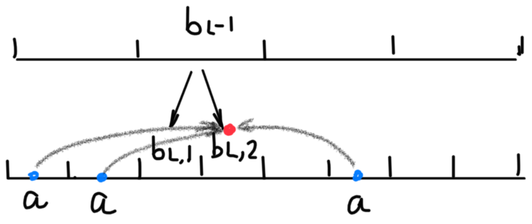

FIGURE 8. An $\mathcal{O}(N)$ bottom-to-top algorithm for computing the coefficients $b_{l,k,m}$

```
for L = J-1:-1:1
    for k = 1:2^L % multipole to multipole translation (restriction)
        a(L,k) = M2M(a(L+1,kc(1)),a(L+1,kc(2)));
    end
end
%% Evaluation (top to bottom)
for L = 2:J-1
    for k = 1:2^L
        for s in the interaction list of T(L,k)
            b(L,k) = b(L,k) + M2L(a(L,s));  % multipole to local  translation
        end
        b(L+1,kc(1:2)) =  L2L(b(L,k)); % local to local translation (prolongation)
    end
end
for k=1:2^J % evaluation in the finest level
    for each y(i) in T(J,k)
        u(i) = evaluate(y(i),b(J,k)) + near field;
    end
end
```

In each level, the computational cost is $\mathcal{O}(pN_l)$. So the total complexity is $\mathcal{O}(p\sum_{l=J}^{1} N_l) = \mathcal{O}(pN)$. Now the ordering of levels does matter but for a given level, the computation of cells is independent of each other.

REFERENCES

[1] J. Barnes and P. Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, 324:446–449, 1986.
[2] R. Beatson and L. Greengard. A short course on fast multipole methods.
[3] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.