

A Fast Multipole Code for the 2-D Coulomb Problem

The following program `fastm.cpp` implements the Fast Multipole Algorithm for a 2-D system of charges in a square region.

Headers, global variables, and function declarations

```
#include <cmath>
#include <complex>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <list>

using namespace std;

// global variables
int N = 1000;           // number of charges
double L = 10;          // side of square region containing charges
complex<double> *z;      // positions of particles
double *q;              // charges of particles
complex<double> Epot;    // total potential energy of system

int levels;             // number of levels of subdivision
int M = 1;              // maximum moment is dipole in this program
list<int> **boxList;     // 2-D array of lists of particle indices
```

Multipole moments and local coefficients will be represented by 4-dimensional arrays: the first index labels

the level in the quadtree, the second and third indices label the box, and the fourth index labels the order of the moment or local coefficient.

```
complex<double> ****a;    // multipole moments at all levels
complex<double> ****b;    // local expansion coefficients at all levels

// function declarations
int boxesEachDirection(int level);
void computeEpot();
void computeMomentsFinestLevel();
void convertMomentsToLocal(int level);
void initialize();
void mergeMoments(int level);
void shiftLocalToChildren(int level);
void sortBodies(int level);
```

Initialization

This function allocates memory for arrays, and places N random charges at random locations within the system volume.

```
void initialize() {

    // allocate arrays for charges and positions
    q = new double [N];
    z = new complex<double> [N];
```

```
// place particles with random charges at random locations
for (int i = 0; i < N; i++) {
    q[i] = 2 * rand() / double(RAND_MAX) - 1;
    double x = L * rand() / double(RAND_MAX);
    double y = L * rand() / double(RAND_MAX);
    z[i] = complex<double>(x, y);
}

// compute number of levels
levels = 0;
int boxes = 1;
while (boxes < N) {
    boxes *= 4;
    ++levels;
}

// allocate arrays for interaction lists, moments and local coefficients
a = new complex<double>*** [levels+1];
b = new complex<double>*** [levels+1];
for (int level = 0; level <= levels; level++) {
    int B = boxesEachDirection(level);
    if (level == levels)
        boxList = new list<int>* [B];
    a[level] = new complex<double>** [B];
    b[level] = new complex<double>** [B];
    for (int i = 0; i < B; i++) {
        if (level == levels)
            boxList[i] = new list<int> [B];
        a[level][i] = new complex<double>* [B];
    }
}
```

```

        b[level][i] = new complex<double>* [B];
        for (int j = 0; j < B; j++) {
            a[level][i][j] = new complex<double> [M+1];
            b[level][i][j] = new complex<double> [M+1];
        }
    }
}

```

Computation of the potential energy

The following function pulls together the various steps of the fast multipole algorithm. The multipole moments are computed at the finest level and then *merged* upward into the coarser levels. Then the interaction lists are processed at each level to determine the local expansion coefficients. Finally, the local coefficients are shifted down from the coarser levels and accumulated at the finest level.

```

void computeEpot() {

    // sort charges into box lists at the finest level
    sortBodies(levels);

    // compute multipole moments
    computeMomentsFinestLevel();
    for (int level = levels - 1; level > 1; level--)
        mergeMoments(level);

    // convert interaction list moments to local expansions
    for (int level = 2; level <= levels; ++level)

```

```
convertMomentsToLocal(level);

// shift expansion coefficients from parents to children
for (int level = 2; level < levels; level++)
    shiftLocalToChildren(level);
```

Now that all multipole fields at all levels have been converted to local coefficients at the finest level, the local interactions can be evaluated.

```
// initialize accumulator
Epot = 0;

// compute local expansion contributions to Epot
int B = boxesEachDirection(levels);
for (int i = 0; i < B; i++)
    for (int j = 0; j < B; j++) {

        // find center of the box
        complex<double> zB(i + 0.5, j + 0.5);
        zB *= L / B;

        // loop over charges in box
        list<int>::const_iterator pos;
        for (pos = boxList[i][j].begin(); pos != boxList[i][j].end(); pos++) {
            int n = *pos;
            Epot += q[n] * (b[levels][i][j][0] +
                           b[levels][i][j][1] * (z[n] - zB));
        }
    }
```

```
}
```

As in the tree-code algorithm, the interactions between charges in near neighbor boxes at the finest level are evaluated exactly.

```
// compute near neighbor contributions to Epot
for (int i = 0; i < B; i++)
for (int j = 0; j < B; j++) {
    for (int k = i - 1; k <= i + 1; k++)
    for (int l = j - 1; l <= j + 1; l++) {
        if (k < 0 || k >= B || l < 0 || l >= B) // outside system volume
            continue;
        list<int>::const_iterator p1, p2;
        for (p1 = boxList[i][j].begin(); p1 != boxList[i][j].end(); p1++)
        for (p2 = boxList[k][l].begin(); p2 != boxList[k][l].end(); p2++) {
            int n1 = *p1;
            int n2 = *p2;
            if (n1 != n2)
                Epot += q[n1] * q[n2] * log(z[n1] - z[n2]);
        }
    }
}
```

Remember to correct for double counting of pairs of particles.

```
Epot /= 2;
}
```

Computation of moments

The total charge and dipole moment are evaluated explicitly only at the finest level. Moments at coarser levels are computed successively by *merging* the moments of the four children to obtain the moments of the parent box:

$$a_0 = \sum_{k=1}^4 a_{0_k} , \quad a_1 = \sum_{k=1}^4 [a_{1_k} + a_{0_k} c_k] .$$

```
void mergeMoments(int level) {  
  
    // loop over all boxes at this parent level  
    int B = boxesEachDirection(level);  
    for (int i = 0; i < B; i++)  
        for (int j = 0; j < B; j++) {  
  
            // zero moments  
            for (int k = 0; k <= M; k++)  
                a[level][i][j][k] = 0;  
  
            // loop over four children  
            for (int k = 0; k < 2; k++)  
                for (int l = 0; l < 2; l++) {  
                    int ic = 2 * i + k;  
                    int jc = 2 * j + l;  
                    complex<double> ck(2 * k - 1, 2 * l - 1);  
                    ck *= L / (4 * B);  
                }  
        }  
}
```

```
        a[level][i][j][0] += a[level + 1][ic][jc][0];
        a[level][i][j][1] += a[level + 1][ic][jc][1]
            + a[level + 1][ic][jc][0] * ck;
    }
}
```

Conversion of moments to local expansion coefficients

In the tree-code algorithm, the charges in each box at a given level interacts with moments of the interaction list of boxes. In this fast multipole algorithm, the moments of the interaction list boxes are converted into local coefficients.

```
void convertMomentsToLocal(int level) {

    // loop over all boxes at this level
    int B = boxesEachDirection(level);
    for (int i = 0; i < B; i++)
        for (int j = 0; j < B; j++) {

            // zero local coefficients
            for (int k = 0; k <= M; k++)
                b[level][i][j][k] = 0;

            // find interaction list
            int i0 = (i / 2) * 2 - 2;
            int j0 = (j / 2) * 2 - 2;
            for (int i1 = i0; i1 < i0 + 6; i1++) {
```

```

if (i1 < 0 || i1 >= B)                // outside system volume
    continue;
for (int j1 = j0; j1 < j0 + 6; j1++) {
    if (j1 < 0 || j1 >= B)            // outside system volume
        continue;
    // exclude near neighbors
    if ( (i - i1) * (i - i1) < 2 && (j - j1) * (j - j1) < 2 )
        continue;

```

The local field expansion coefficients are now computed using

$$b_0 = a_0 \log(z_B) - \frac{a_1}{z_B}, \quad b_1 = \left(\frac{a_0}{z_B} + \frac{a_1}{z_B^2} \right).$$

```

// displacement between box centers
complex<double> zB(i - i1, j - j1);
zB *= L / B;

// accumulate local coefficients
b[level][i][j][0] += a[level][i1][j1][0] * log(zB)
                  - a[level][i1][j1][1] / zB;
b[level][i][j][1] += a[level][i1][j1][0] / zB
                  + a[level][i1][j1][1] / (zB * zB);
    }
}
}

```

Shifting of local expansions from parents to children

This function *shifts* the local expansion coefficients in a parent box to its four children using the relations

$$b_{0_k} = b_0 + b_1 c_k, \quad b_{1_k} = b_1,$$

where c_k is the displacement of the center of child k from the center of its parent box.

```
void shiftLocalToChildren(int level) {

    // loop over all boxes at this parent level
    int B = boxesEachDirection(level);
    for (int i = 0; i < B; i++)
        for (int j = 0; j < B; j++) {

            // loop over four children
            for (int k = 0; k < 2; k++)
                for (int l = 0; l < 2; l++) {
                    int ic = 2 * i + k;
                    int jc = 2 * j + l;
                    complex<double> ck(2 * k - 1, 2 * l - 1);
                    ck *= L / (4 * B);
                    b[level+1][ic][jc][0] += b[level][i][j][0]
                                            + b[level][i][j][1] * ck;
                    b[level+1][ic][jc][1] += b[level][i][j][1];
                }
        }
}
```

The computation of moments at the finest level uses the same procedures as the function `computeMoments` in `treecode.cpp`:

```
void computeMomentsFinestLevel() {  
  
    // loop over all boxes at finest level  
    int B = boxesEachDirection(levels);  
    for (int i = 0; i < B; i++)  
        for (int j = 0; j < B; j++) {  
  
            // find center of the box  
            complex<double> zCenter(i + 0.5, j + 0.5);  
            zCenter *= L / B;  
  
            // zero moments  
            for (int k = 0; k <= M; k++)  
                a[levels][i][j][k] = 0;  
  
            // loop over charges in box  
            list<int>::const_iterator pos;  
            for (pos = boxList[i][j].begin(); pos != boxList[i][j].end(); pos++) {  
                int n = *pos;  
                a[levels][i][j][0] += q[n];  
                a[levels][i][j][1] += q[n] * (z[n] - zCenter);  
            }  
        }  
}
```

Functions repeated from treecode.cpp

```
int boxesEachDirection(int level) {  
  
    // find number of boxes in each direction  
    int B = 1;  
    for (int i = 0; i < level; i++)  
        B *= 2;  
  
    return B;  
}  
  
void sortBodies(int level) {  
  
    int B = boxesEachDirection(level);  
  
    // clear any lists previously constructed  
    for (int i = 0; i < B; i++)  
        for (int j = 0; j < B; j++)  
            boxList[i][j].clear();  
  
    // add particle indexes to list  
    for (int n = 0; n < N; n++) {  
        int i = int(z[n].real() / L * B);  
        int j = int(z[n].imag() / L * B);  
        boxList[i][j].push_front(n);  
    }
```

```
}
```

The main function

```
int main(int argc, char *argv[]) {  
  
    if (argc > 1)  
        N = atoi(argv[1]);  
    if (argc > 2)  
        L = atof(argv[2]);  
    cout << "Fast-Multipole Algorithm for Charges in 2-D" << endl;  
    cout << "Number of charges   = " << N << endl;  
    cout << "Side of square box = " << L << endl;  
  
    initialize();  
    clock_t t0 = clock();  
    computeEpot();  
    clock_t t1 = clock();  
    cout.precision(16);  
    cout << "   Potential energy = " << Epot << endl;  
    cout << "           CPU time = " << double(t1 - t0) / CLOCKS_PER_SEC  
        << " sec" << endl;  
  
}
```