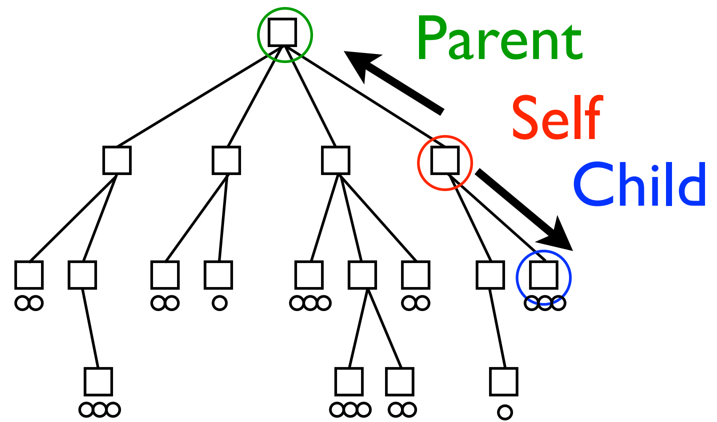
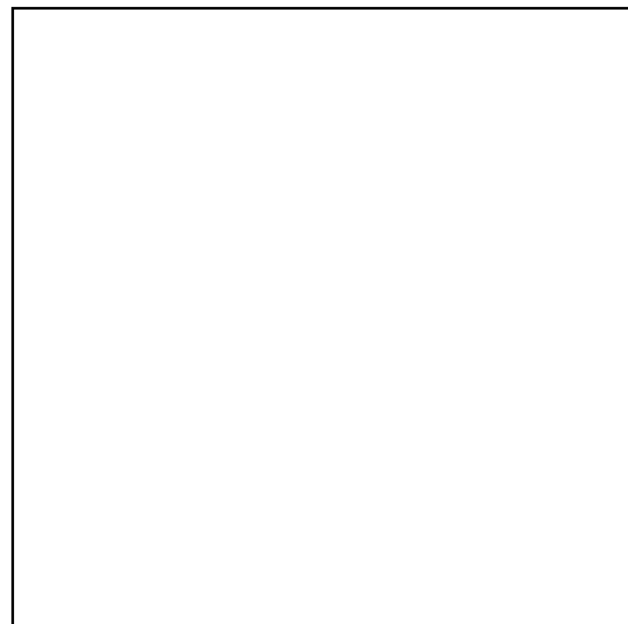
12 Steps to a Fast Multipole Method on GPUs

Rio Yokota
Boston University

Step04. tree construction





NCRIT = 10;

Properties of a cell

```
struct cell {
  int nleaf               number of leafs
  int nchild              octant of children
  int leaf[NCRIT];        pointer to leafs
  float xc,yc,zc,r;       center & size of cell
  cell *parent            pointer to parent cell
  cell *child[8];         pointer to child cells
};
```

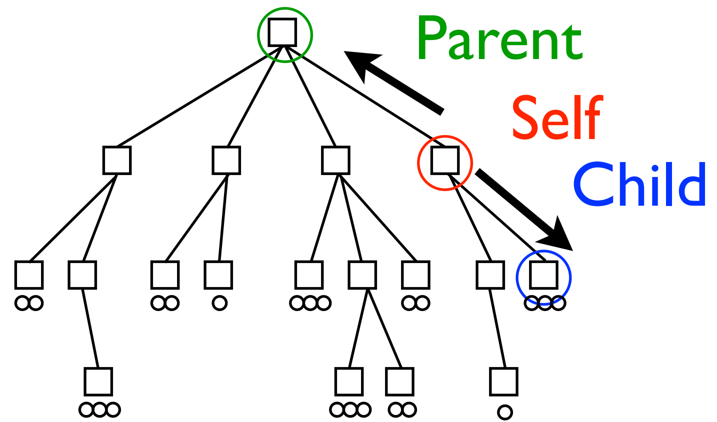
Flow of tree construction

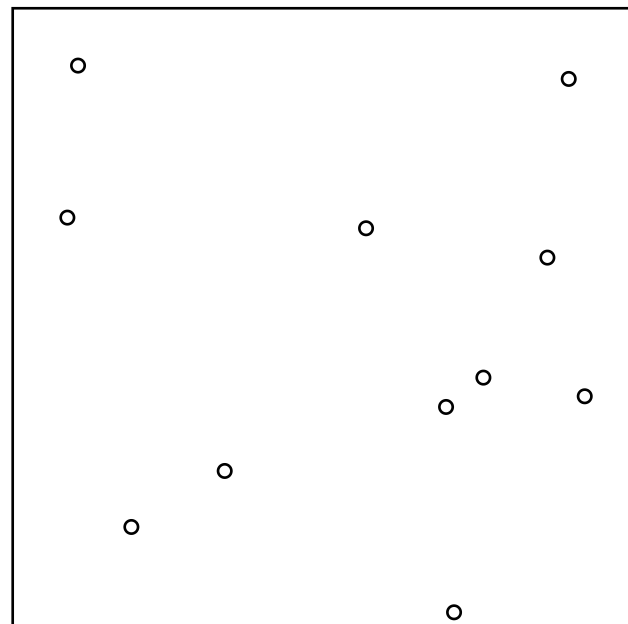
```
for( i=0; i<N; i++ ) {
  add_particle(i):
  C->nleaf++;
  if(C->nleaf >= NCRIT) {
    split_cell(C);
  }
}
```

PASI

Step04. tree construction

PASI





NCRIT = 10;

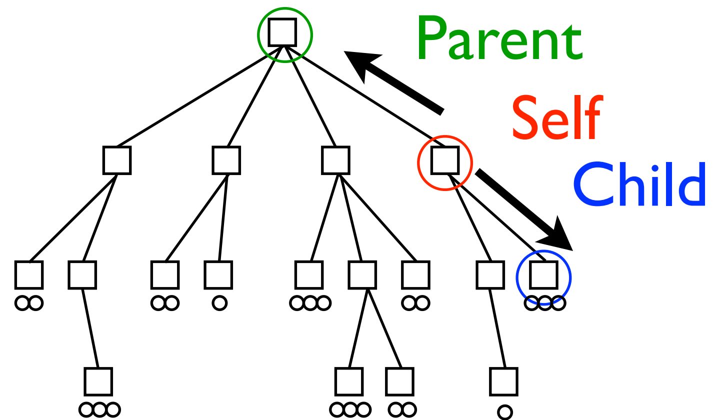
Properties of a cell

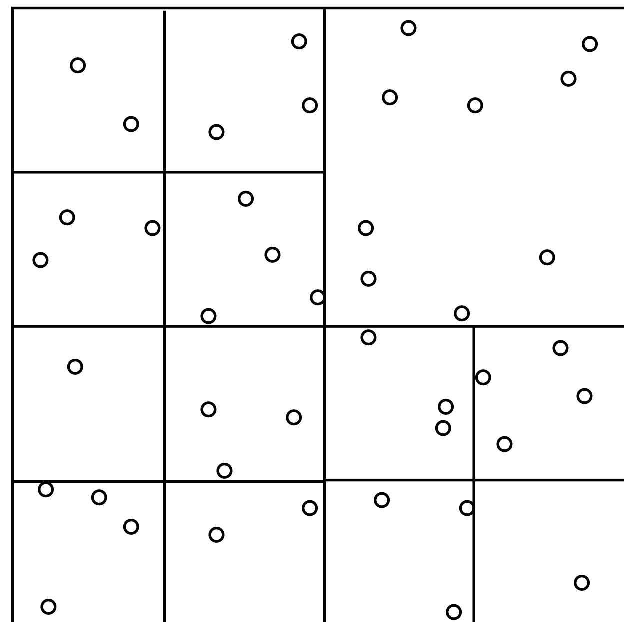
```
struct cell {
  int nleaf                    number of leafs
  int nchild                   octant of children
  int leaf[NCRIT];             pointer to leafs
  float xc,yc,zc,r;            center & size of cell
  cell *parent                 pointer to parent cell
  cell *child[8];              pointer to child cells
};
```

Flow of tree construction

```
for( i=0; i<N; i++ ) {
  add_particle(i):
  C->nleaf++;
  if(C->nleaf >= NCRIT) {
    split_cell(C);
  }
}
```

Step04. tree construction





NCRIT = 10;

Properties of a cell

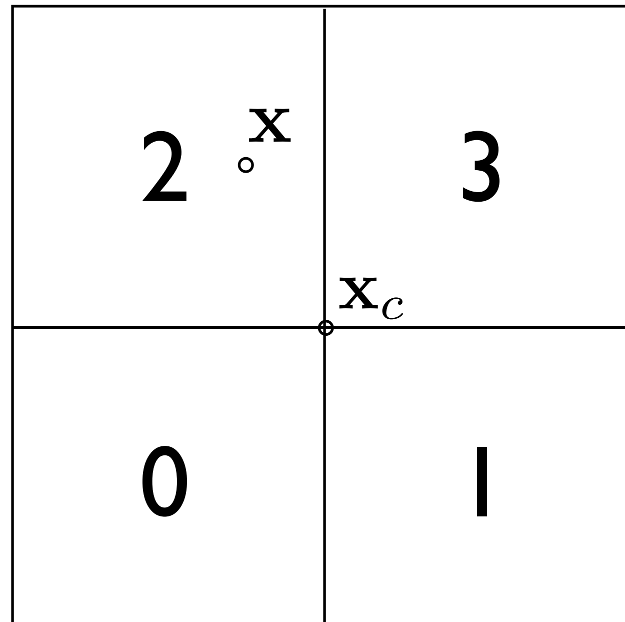
```
struct cell {
  int nleaf              number of leafs
  int nchild             octant of children
  int leaf[NCRIT];       pointer to leafs
  float xc,yc,zc,r;      center & size of cell
  cell *parent           pointer to parent cell
  cell *child[8];        pointer to child cells
};
```

Flow of tree construction

```
for( i=0; i<N; i++ ) {
  add_particle(i):
  C->nleaf++;
  if(C->nleaf >= NCRIT) {
    split_cell(C);
  }
}
```

PASI

Step04. tree construction :: adding particles



Which cell to add particle

octant = (x > xc)+((y > yc) << 1)+((z > zc) << 2);

0 to 7 | 0 or 1 | 0 or 2 | 0 or 4

Keep track of empty cells

nchild |= 1 << octant;

For example if **octant=3**

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

flip 4th bit

non-empty

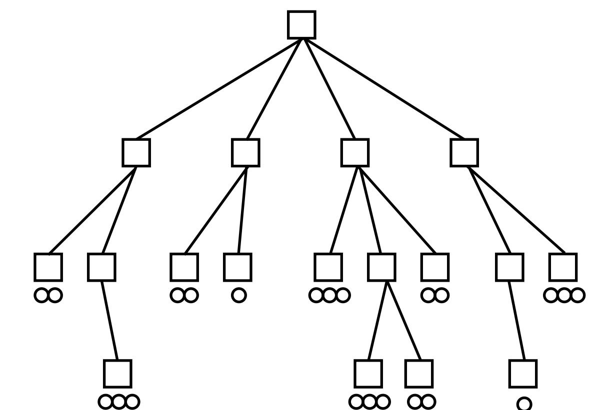
Keep track of leafs in cells

leaf[octant]=i; ← index of the particle

Do this recursively

```
while( C->nleaf >= NCRIT ) {
  C->nleaf++;
  if( !(C->nchild & (1 << octant)) )
    add_child();
  C = C->child[octant];
}
```

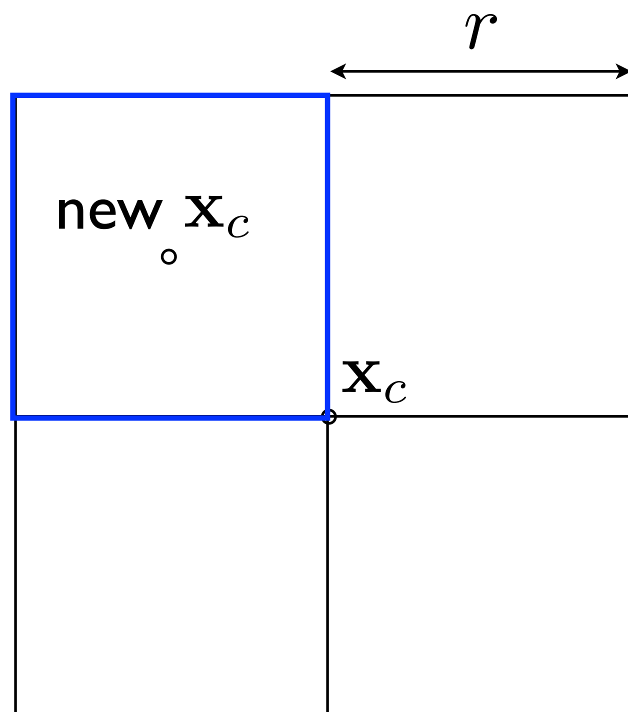
```
struct cell {
  int nleaf
  int nchild
  int leaf[NCRIT];
  float xc,yc,zc,r;
  cell *parent
  cell *child[8];
};
```



NCRIT = 3;

PASI

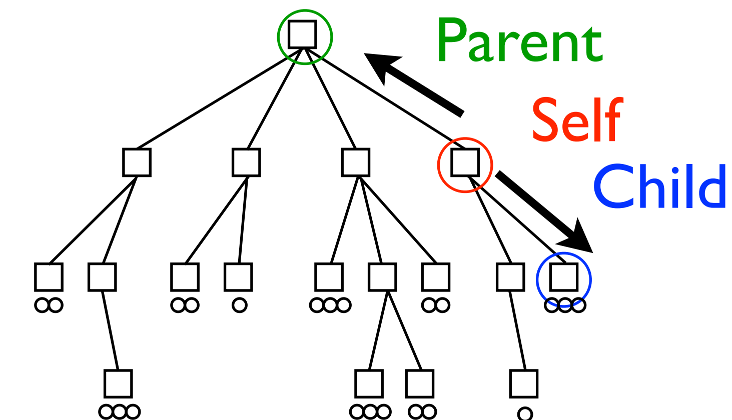
Step04. tree construction :: adding subcells



octant=2;

```
struct cell {
  int nleaf
  int nchild
  int leaf[NCRIT];
  float xc,yc,zc,r;
  cell *parent
  cell *child[8];
};
```

```
void add_child(int octant, cell *C, cell *&CN) {
  ++CN;
  initialize(CN);
  CN->r  = C->r/2;
  CN->xc = C->xc+CN->r*((octant&1)*2-1);
  CN->yc = C->yc+CN->r*((octant&2)-1);
  CN->zc = C->zc+CN->r*((octant&4)/2-1);
  CN->parent = C;
  C->child[octant] = CN;
  C->nchild |= (1 << octant);
}
```

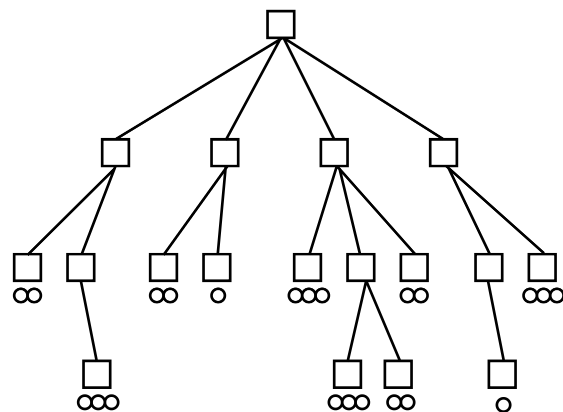


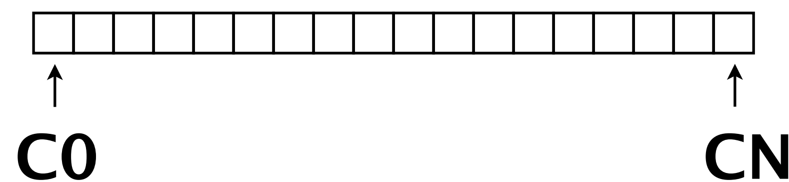
PASI

Step04. tree construction :: actual code

PASI

```
cell C0[N];
cell *CN = C0;
for( int i=0; i<N; i++ ) {
  cell *C = C0;
  while( C->nleaf >= NCRIT ) {
    C->nleaf++;
    int octant = (x[i] > C->xc) + ((y[i] > C->yc) << 1) + ((z[i] > C->zc) << 2);
    if( !(C->nchild & (1 << octant)) ) add_child(octant,C,CN);
    C = C->child[octant];
  }
  C->leaf[C->nleaf++] = i;
  if( C->nleaf >= NCRIT ) split_cell(x,y,z,C,CN);
}
```

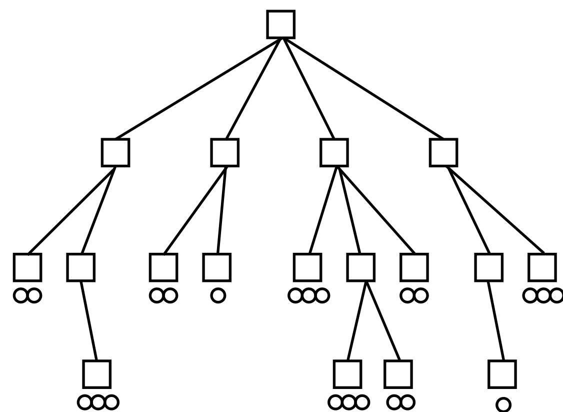


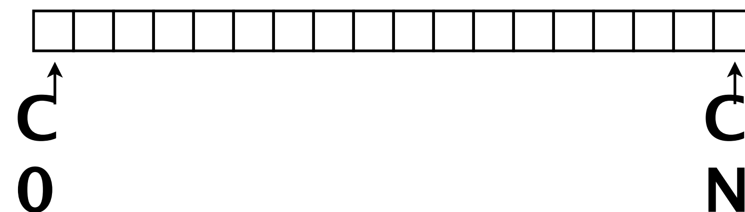


Step04. tree construction :: splitting cells

PASI

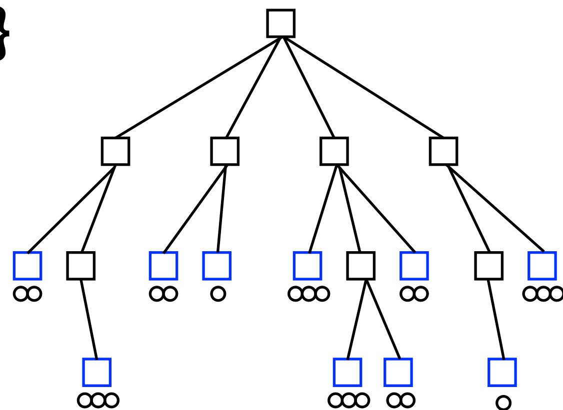
```
void split_cell(float *x, float *y, float *z, cell *C, cell *&CN) {
  for( int i=0; i<NCRIT; i++ ) {
    int l = C->leaf[i];
    int octant = (x[l] > C->xc) + ((y[l] > C->yc) << 1) + ((z[l] > C->zc) << 2);
    if( !(C->nchild & (1 << octant)) ) add_child(octant,C,CN);
    cell *CC = C->child[octant];
    CC->leaf[CC->nleaf++] = l;
    if( CC->nleaf >= NCRIT ) split_cell(x,y,z,CC,CN);
  }
}
```



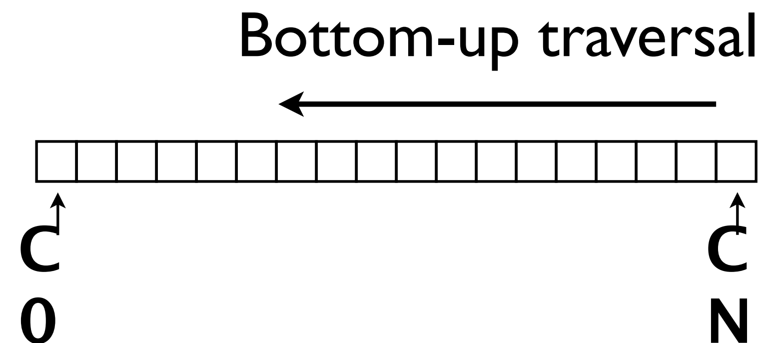


Step04. tree construction :: traversal

```
void traverse(cell *C, int &cells, int &leafs) {
  if( C->nleaf >= NCRIT ) {
    for( int c=0; c<8; c++ )
      if( C->nchild & (1 << c) ) {
        // depth-first traversal
        traverse(C->child[c],cells,leafs);
      }
  } else {
    for( int l=0; l<C->nleaf; l++ ) {
      // these are leafs
      leafs++;
    }
    // these are twigs
    cells++;
  }
}
```



Bottom-up traversal

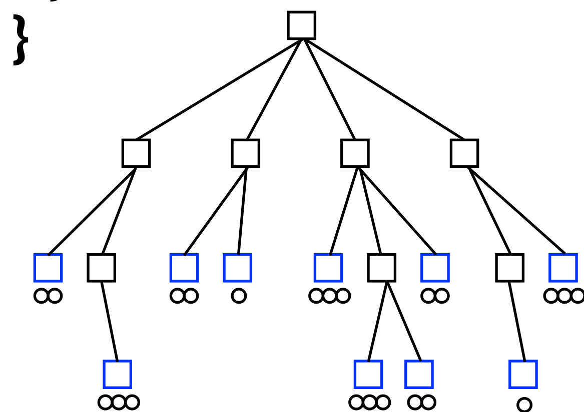


PASI

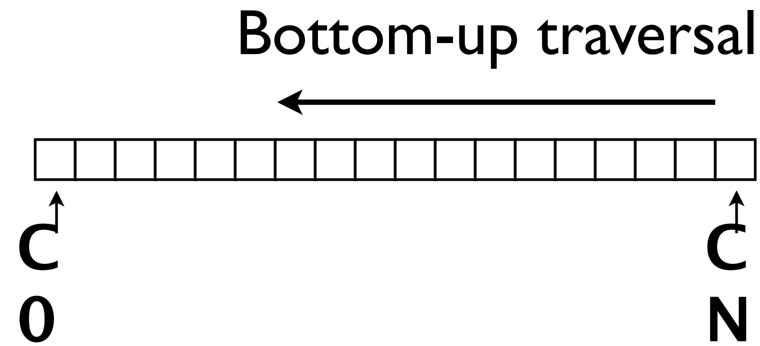
Step05. upward sweep :: multipole expansion

PASI

```
void getMultipole(cell *C, float *x, float *y, float *z, float *m) {
  float dx,dy,dz;
  if( C->nleaf >= NCRIT ) {
    for( int c=0; c<8; c++ )
      if( C->nchild & (1 << c) ) getMultipole(C->child[c],x,y,z,m);
  } else {
    for( int l=0; l<C->nleaf; l++ ) {
      int j = C->leaf[l];
      dx = C->xc-x[j];
      dy = C->yc-y[j];
      dz = C->zc-z[j];
      C->multipole[0] += m[j];
      C->multipole[1] += m[j]*dx;
      C->multipole[2] += m[j]*dy;
      C->multipole[3] += m[j]*dz;
    }
  }
}
```



Step05. upward sweep :: multipole translation



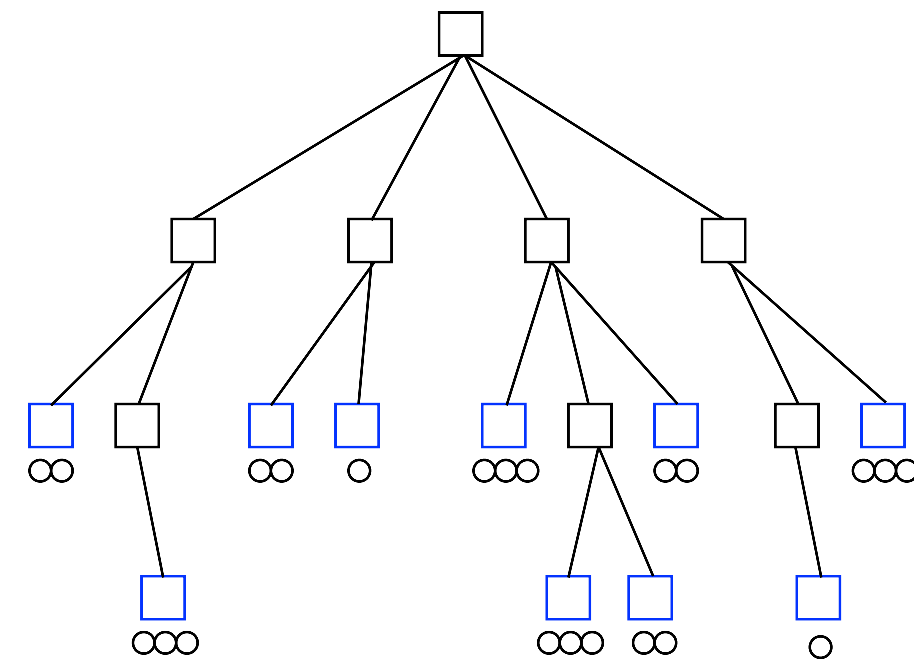

```
for( cell *C=CN; C!=C0; --C ) {
  cell *P = C->parent;
  upwardSweep(C,P);
}
```

```
void upwardSweep(cell *C, cell *P) {
  float dx,dy,dz;
  dx = P->xc-C->xc;
  dy = P->yc-C->yc;
  dz = P->zc-C->zc;
  P->multipole[0] += C->multipole[0];
  P->multipole[1] += C->multipole[1]+ dx*C->multipole[0];
  P->multipole[2] += C->multipole[2]+ dy*C->multipole[0];
  P->multipole[3] += C->multipole[3]+ dz*C->multipole[0];
}
```

PASI

Step06. treecode

```
void evaluate(cell *C, float *x, float *y, float *z, float *m, float &p, int i) {
  float dx,dy,dz,r,X,Y,Z,R,R3,R5;
  if( C->nleaf >= NCRIT ) {
    for( int c=0; c<8; c++ )
      if( C->nchild & (1 << c) ) {
        cell *CC = C->child[c];
        dx = x[i]-CC->xc;
        dy = y[i]-CC->yc;
        dz = z[i]-CC->zc;
        r = sqrtf(dx*dx+dy*dy+dz*dz);
        if( CC->r > THETA*r ) {
          evaluate(CC,x,y,z,m,p,i);
        } else {
          // multipole evaluation
        }
      }
  } else {
    for( int l=0; l<C->nleaf; l++ ) {
      // direct summation
    }
  }
}
```

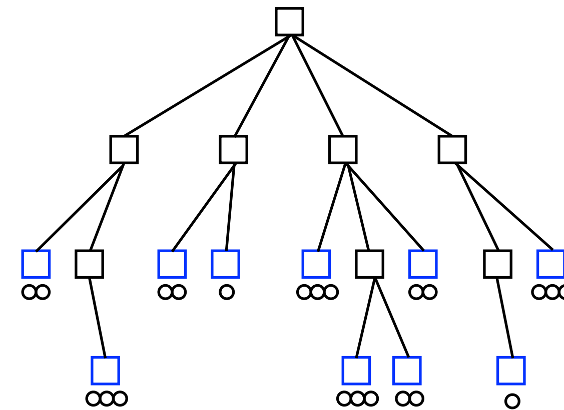


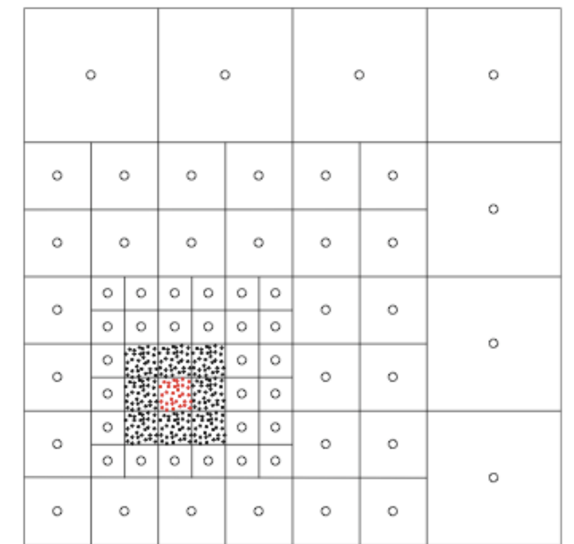
```
for( int i=0; i<N; i++ ) {
  cell *C = C0;
  evaluate(C,x,y,z,m,p,i);
}
```

PASI

Step07. vectorized treecode

```
void evaluate(cell *CI, cell *CJ, float *x, float *y, float *z, float *m, float *p) {
  float dx,dy,dz,r,X,Y,Z,R,R3,R5;
  if( CJ->nleaf >= NCRIT ) {
    for( int c=0; c<8; c++ ) {
      if( CJ->nchild & (1 << c) ) {
        cell *CC = CJ->child[c];
        dx = CI->xc-CC->xc;
        dy = CI->yc-CC->yc;
        dz = CI->zc-CC->zc;
        r = sqrtf(dx*dx+dy*dy+dz*dz);
        if( CI->r+CC->r > THETA*r ) {
          evaluate(CI,CC,x,y,z,m,p);
        } else {
          for( int l=0; l<CI->nleaf; l++ ) {
            // multipole evaluation
          }
        }
      }
    }
  } else {
    for( int li=0; li<CI->nleaf; li++ ) {
      int i = CI->leaf[li];
      for( int lj=0; lj<CJ->nleaf; lj++ ) {
        // direct summation
      }
    }
  }
}
```





```
for( int t=0; t<ntwig; t++ ) {
  cell *CI = twig[t];
  cell *CJ = C0;
  evaluate(CI,CJ,x,y,z,m,p);
}
```

PASI