A cost optimal parallel algorithm for computing force field in N-body simulations on a CREW PRAM

Guoliang Xue
Department of Computer Science, The University of Vermont, Burlington, VT 05405, USA

Received October 1998; revised July 1999
Communicated by D.-Z. Du

Abstract

We consider the following force field computation problem: given a cluster of \( n \) particles in three-dimensional space, compute the force exerted on each particle by the other particles. Depending on different applications, the pairwise interaction could be either gravitational or Lennard–Jones. In both cases, the force between two particles vanishes as the distance between them approaches to infinity. Since there are \( n(n-1)/2 \) pairs, direct method requires \( \Theta(n^2) \) time for force evaluation, which is very expensive for astronomical simulations. In 1985 and 1986, two famous \( \Theta(n \log n) \) time hierarchical tree algorithms were published by Appel (SIAM J. Sci. Statist. Comput. 6 (1985) 85–103) and by Barnes and Hut (Nature, 324 (1980) 446–449), respectively. In a recent paper, we presented a linear time algorithm which builds the oct tree bottom-up and showed that Appel’s algorithm can be implemented in \( \Theta(n) \) sequential time. In this paper, we present an algorithm which computes the force field in \( \Theta(n \log n) \) time on a \( \Theta(n/\log n) \) processor CREW PRAM. A key to this optimal parallel algorithm is replacing a recursive top-down force calculation procedure of Appel by an equivalent non-recursive bottom-up procedure. Our parallel algorithm also yields a new \( \Theta(n) \) time-sequential algorithm for force field computation. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Spatial tree algorithms; Force field evaluation; N-body simulations; PRAM; Cost optimal algorithms

1. Introduction and assumption

Fast algorithms for force field evaluation have important applications in molecular conformation, molecular dynamics, and astrophysical simulations. Given a cluster of \( n \) particles in three-dimensional space, we need to compute the force exerted on...
each particle by the other particles. Since there are \( n(n - 1)/2 \) pairs, direct method requires \( \Theta(n^2) \) time for force evaluation, which is very expensive for astronomical simulations.

In astrophysical simulations, the force exerted on one particle by another is given by the gravitational force. In molecular dynamics and molecular conformation, the Lennard–Jones potential is widely used. In both cases, the force exerted on one particle by another particle vanishes as the distance between them approaches to infinity. This observation leads to several fast approximation algorithms. In 1985 and 1986, two famous \( \Theta(n \log n) \) time-hierarchical tree algorithms were published by Appel [3] and by Barnes and Hut [4], respectively. In 1987, Greengard and Rokhlin [8] published the fast multipole algorithm which computes the force field in \( O(n) \) time. Recently, Aluru [1] showed that Greengard’s algorithm is not \( O(n) \). These algorithms have made great impacts on the computational study of molecular conformation/dynamics and astronomical simulations. Parallel implementations of these algorithms have been reported by many authors, including [7, 12–14, 17]. Due to the big constant in the fast multipole algorithm and the simplicity and efficiency of the tree algorithms, hierarchical tree algorithms received more attention in computational studies [2]. Therefore we concentrate on tree algorithms in this paper.

A central idea behind Appel’s algorithm and the Barnes–Hut algorithm is the monopole approximation [3]. Appel showed that when a group of \( n_1 \) particles and a group of \( n_2 \) particles are well separated from each other, we can approximate the \( n_1 \times n_2 \) pair interactions by a single-pair interaction between two big particles (one at the gravitational center of the first group and one at the gravitational center of the other). The tree algorithms consist of two phases. In the first phase, an oct-tree is constructed which hierarchically partitions the particles into many smaller clusters. In the second phase, the oct-tree is used to compute an approximation to the force field. In most simulations, the particles are almost homogeneously distributed. In this case, the oct-tree for an \( n \)-particle cluster has a height of \( \Theta(\log n) \). The oct-tree was built using the following top-down approach. The root node corresponds to a computation box (a cube) big enough to contain all the particles in the given cluster. The \( n \) particles are inserted to the root of the tree one by one. Whenever a node in the tree has two particles, the corresponding computation box is subdivided into 8 smaller computation boxes, which correspond to the 8 children of the current node. The particles in the current node are then inserted to the children nodes according to their spatial positions. It is clear that \( \Theta(n \log n) \) time is required to build the oct-tree this way if the height of the tree is bounded by \( \Theta(\log n) \). Both the Appel’s algorithm and the Barnes and Hut algorithm require \( \Theta(n \log n) \) time to build the tree for homogeneously distributed clusters.

In a recent paper [16], Xue presented an algorithm which builds the oct-tree bottom-up in \( \Theta(n) \) sequential time and showed that Appel’s algorithm can be implemented in \( \Theta(n) \) sequential time. That algorithm computes the force field top-down using a recursive procedure. It seems difficult to parallelize that linear time algorithm efficiently. In this paper, we replace the recursive top-down procedure of Appel by an equivalent
non-recursive bottom-up procedure and present a parallel algorithm which computes force field in $\Theta(\log n)$ time on a $\Theta(n/\log n)$ processor CREW PRAM.

The rest of this paper is organized as follows. In Section 2, we show that the oct-tree can be constructed bottom-up in $\Theta(\log n)$ time on a $\Theta(n/\log n)$ processor CREW PRAM. In Section 3, we show that the force field can be computed in $\Theta(\log n)$ time on a $\Theta(n/\log n)$ processor CREW PRAM. We conclude the paper in Section 4. Throughout this paper, we make the following assumption on the distribution of the particles:

**Assumption 1.1.** There exist two positive constants $c_1$ and $c_2$ such that the minimum inter-particle distance is at least $c_1$ and the maximum inter-particle distance is smaller than $c_2 n^{1/3}$.

Assumption 1.1 is highly believed to be true for most applications and is supported by many computer simulations. For the Lennard–Jones cluster, it is proved that the minimum inter-particle distance is greater than or equal to 0.5 (which is independent on the number of particles in the cluster) [15]. When the particles are homogeneously distributed, it is straightforward that the inter-particle distances are bounded by $O(n^{1/3})$.

## 2. Building the oct-tree bottom-up in $\Theta(\log n)$ time

In Section 2.1, we will describe the necessary data structure used in our algorithms. In Section 2.2, we will present a $\Theta(\log n)$ time algorithm for constructing the oct-tree using a $P = \Theta(n/\log n)$ processor CREW PRAM. The time complexity of our algorithm is analyzed in Section 2.3. To simplify the analysis, we assume that $P = n/ \log n$ and that both $n$ and $\log n$ are powers of 8. It is well-known that this assumption does not affect the asymptotic analysis of the algorithm.

### 2.1. Data structures

A *computation box* is defined by a point $\text{base}$ in three-dimensional space and a positive number $\text{size}$. Let $\text{base}X$, $\text{base}Y$, $\text{base}Z$ be the coordinates of $\text{base}$. The computation box defined by $\text{base}$ and $\text{size}$ is

$$[\text{base}X, \text{base}X + \text{size}) \times [\text{base}Y, \text{base}Y + \text{size}) \times [\text{base}Z, \text{base}Z + \text{size}). \quad (2.1)$$

Note that each interval in the Cartesian product (2.1) is closed on the left but open on the right. This convention makes the partition of a computation box much easier. A computation box is illustrated in Fig. 1(a). When a computation box is partitioned, we obtain 8 non-intersecting computation boxes of equal size whose union is the original computation box. An example is illustrated in Fig. 1(b).
We will make reference to the following data structure during our description of the algorithm:

```c
typedef struct _node{
    struct _node *parent; struct _node *child[8];
    int isLeaf; int weight; int pindex;
    double coordX; double coordY; double coordZ;
    double forceX; double forceY; double forceZ;
    double baseX; double baseY; double baseZ; double size;
}NODE;
```

Each node in the oct-tree is of type NODE. For every node in the tree, weight is the number of particles contained in the corresponding computation box. If weight is 0 or 1, we have a leaf node and the field isLeaf is 1. If weight is 2 or larger, we have an interior node and the field isLeaf is 0. For an interior node, the fields coordX, coordY, coordZ represent the coordinates of the gravitational center of the particles contained in the computation box corresponding to this node. The fields baseX, baseY, baseZ and size define the computation box corresponding to the node in the tree. The fields forceX, forceY and forceZ are used in the evaluation of force field whose use will be discussed later. For a leaf node whose weight is 1, pindex is the index of the unique particle that is contained in the computation box corresponding to the leaf node. The field parent contains a pointer to the parent node in the tree. For the root node, parent is NULL. For any interior node, child[j] is a pointer to the jth child of the current node (j = 0, 1, ..., 7). We make the assumption that every interior node has exactly 8 children whose computation boxes have the same size, which is half of the size of the computation box of the current node. If the computation box of the current node is given in (2.1), then the computation boxes for child[0], child[1], ..., child[7] are defined as follows:

\[
\begin{align*}
\text{child[0]} & : [\text{base X}, \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z}, \text{base Z} + \frac{1}{2} \text{size})], \\
\text{child[1]} & : [\text{base X}, \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z} + \frac{1}{2} \text{size}, \text{base Z} + \text{size})], \\
\text{child[2]} & : [\text{base X} + \frac{1}{2} \text{size}), \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z}, \text{base Z} + \text{size})], \\
\text{child[3]} & : [\text{base X}, \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y} + \frac{1}{2} \text{size}, \text{base Y} + \text{size}) \times [\text{base Z}, \text{base Z} + \frac{1}{2} \text{size})], \\
\text{child[4]} & : [\text{base X}, \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z} + \frac{1}{2} \text{size}, \text{base Z} + \text{size})], \\
\text{child[5]} & : [\text{base X} + \frac{1}{2} \text{size}), \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z}, \text{base Z} + \text{size})], \\
\text{child[6]} & : [\text{base X}, \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z} + \frac{1}{2} \text{size}, \text{base Z} + \text{size})], \\
\text{child[7]} & : [\text{base X} + \frac{1}{2} \text{size}), \text{base X} + \frac{1}{2} \text{size}) \times [\text{base Y}, \text{base Y} + \frac{1}{2} \text{size}) \times [\text{base Z}, \text{base Z} + \text{size})
\end{align*}
\]
Algorithm 2.1 (part 1) {Building the oct-tree bottom-up.}

Step.1 {Determine the sizes of the root box and the leaf boxes} Using all $P$ processors to compute the following 6 values:

- $X_{\text{min}} := \min_{i=0\ldots n-1} \text{part}[i].x$; $X_{\text{max}} := \max_{i=0\ldots n-1} \text{part}[i].x$;
- $Y_{\text{min}} := \min_{i=0\ldots n-1} \text{part}[i].y$; $Y_{\text{max}} := \max_{i=0\ldots n-1} \text{part}[i].y$;
- $Z_{\text{min}} := \min_{i=0\ldots n-1} \text{part}[i].z$; $Z_{\text{max}} := \max_{i=0\ldots n-1} \text{part}[i].z$.

Let $\delta := \frac{\sqrt{2}}{2}c_1$. Let maxlevel be the smallest positive integer such that $\delta 2^{\text{maxlevel}} > \max\{X_{\text{max}} - X_{\text{min}}, Y_{\text{max}} - Y_{\text{min}}, Z_{\text{max}} - Z_{\text{min}}\}$. Let $A := \delta \times 2^{\text{maxlevel}}$.

Step.2 {Allocate space} We will use a three-dimensional array of NODE for the nodes on each level of the oct-tree. Let $\text{tree}[l]$ be a pointer to the three-dimensional array of NODE with $2^l \times 2^l \times 2^l$ elements ($l = 0, 1, \ldots, \text{maxlevel}$).

It is clear that we require to allocate $O(n)$ space because there are $8^l$ tree nodes on level-$l$ of the tree. These arrays are dynamically allocated at this time.

Step.3 {Construct the leaf nodes} for $p = 0$ to $P$, processor $P_p$ does the following:

for $i = p$ to $n-1$ step $P$ do

Let $t = \left\lfloor \frac{\text{part}[i].x - X_{\text{min}}}{\delta} \right\rfloor$; $j = \left\lfloor \frac{\text{part}[i].y - Y_{\text{min}}}{\delta} \right\rfloor$; $k = \left\lfloor \frac{\text{part}[i].z - Z_{\text{min}}}{\delta} \right\rfloor$.

- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].x := \text{baseX}$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].y := \text{baseY}$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].z := \text{baseZ}$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].\text{size} := 1$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].\text{pindex} := t$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].\text{coordX} := \text{part}[i].x$);
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].\text{coordY} := \text{part}[i].y$;
- $\text{tree}[\text{maxlevel}] \rightarrow \text{node}[i][j][k].\text{coordZ} := \text{part}[i].z$.

endfor

endfor

Fig. 2. Building the oct-tree bottom-up (part 1).

\[
\begin{align*}
&\text{baseX, baseX + \frac{1}{2}\text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \frac{1}{2}\text{size}}, \\
&\text{baseX + \frac{1}{2}\text{size}, baseX + \text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \frac{1}{2}\text{size}}, \\
&\text{baseX + \frac{1}{2}\text{size}, baseX + \text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \text{size}}, \\
&\text{baseX + \frac{1}{2}\text{size}, baseX + \text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \text{size}}, \\
&\text{baseX + \frac{1}{2}\text{size}, baseX + \text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \text{size}}, \\
&\text{baseX + \frac{1}{2}\text{size}, baseX + \text{size}} \times \text{baseY, baseY + \frac{1}{2}\text{size}} \times \text{baseZ, baseZ + \text{size}}.
\end{align*}
\]

Note that the partition of a computation box takes constant time. The computation box in Fig. 1(a) is partitioned into 8 smaller computation boxes in Fig. 1(b), which in turn are partitioned into a total of 64 even smaller computation boxes in Fig. 1(c).

2.2. Building the oct-tree bottom-up in $O(\log n)$ time

We assume that the $n$ particles are given in an array of points so that $\text{part}[i].x$, $\text{part}[i].y$, $\text{part}[i].z$ represent the coordinates of particle $i$ ($i = 0, 1, 2, \ldots, n-1$). Our PRAM algorithm for oct tree construction is presented as Algorithm 2.1. Since $P$ is assumed to be a power of 8, there is an integer $L$ such that $8^L = P$. To make
Algorithm 2.1 (part 2) \{Building the oct-tree bottom-up.\}

Step 4 \{Building the tree bottom-up\}

for \( l := \text{maxlevel} - 1 \) downto 0 do

All processors \( P_{\text{ijk}} \) (0 \( \leq l, J, K < 2^l \)) do in parallel

for \( i := 2^{l-L} \) to \( 2^{l-L} + 2^{l-L} - 1 \) do

for \( j := 2^{l-L} \) to \( 2^{l-L} + 2^{l-L} - 1 \) do

for \( k := 2^{l-L} \) to \( 2^{l-L} + 2^{l-L} - 1 \) do

\( \text{tree}[l] \rightarrow \text{node}[i][j][k] \), \( \text{child}[0] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 0][2k + 0] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[1] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 0][2k + 1] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[2] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 1][2k + 0] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[3] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 1][2k + 1] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[4] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 0][2k + 0] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[5] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 0][2k + 1] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[6] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 1][2k + 0] \);

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{child}[7] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 1][2k + 1] \);

Also set the parent field for each of the 8 children of tree\([l] \rightarrow \text{node}[i][j][k] \):

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{base : } X := X_{\text{min}} + 2^{\text{maxlevel} - 1}; \)

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{size := } 2^{\text{maxlevel} - 1}; \)

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{base : } Y := Y_{\text{min}} + 2^{\text{maxlevel} - 1}; \)

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{base : } Z := Z_{\text{min}} + 2^{\text{maxlevel} - 1}; \)

Let tree\([l] \rightarrow \text{node}[i][j][k] \), weight be the sum of the weights of its children;

Let tree\([l] \rightarrow \text{node}[i][j][k] \), coord\(X\), tree\([l] \rightarrow \text{node}[i][j][k] \), coord\(Y\), and

\( \text{tree}[l] \rightarrow \text{node}[i][j][k], \text{coord}Z \) be the coordinates of the weighted center of

the particles contained in the computation box of the current node;

if \( \text{tree}[l] \rightarrow \text{node}[i][j][k].\text{weight} == 0 \) then

\( \text{tree}[l] \rightarrow \text{node}[i][j][k].\text{isLeaf} := 1; \)

elsif \( \text{tree}[l] \rightarrow \text{node}[i][j][k].\text{weight} == 1 \) then

\( \text{tree}[l] \rightarrow \text{node}[i][j][k].\text{isLeaf} := 1; \)

\( \text{Let tree}[l] \rightarrow \text{node}[i][j][k], \text{pindex be the index of the only particle} \)

\( \text{contained in the current computation box;} \)

endif

endfor \{k\}

endfor \{j\}

endfor \{i\}

endfor \{l\}

Fig. 3. Building the oct-tree bottom-up (part 2).

the description of the algorithm easier, we assume that the \( P \) processors are labeled as

\( P_{\text{ijk}} \) where \( 0 \leq l < 2^l, 0 \leq J < 2^L, 0 \leq K < 2^L \). We have taken the liberty of treating

the processors as a linear array in Step 3 of the algorithm.

In \( \Theta(n) \) sequential time, we can compute the base point of the computation box for

the root node by computing the minimum of the coordinates of the \( n \) particles for each of

the 3 dimensions. Similarly, we can compute the maximum of the coordinates of the

\( n \) particles for each of the 3 dimensions in \( \Theta(n) \) time. Since the maximum and

minimum of \( n \) numbers can be computed in \( \Theta(\log n) \) time on a \( \Theta(n/\log n) \) processor

PRAM, the above computation task can be accomplished in \( \Theta(\log n) \) parallel time on

the PRAM. By Assumption 1.1, we can now decide the size of the smallest computation
Algorithm 2.1 (part 3) {Building the oct-tree bottom-up.}

Step 5 {Building the tree bottom-up}

\[
\text{for } l := L - 1 \text{ downto } 0 \text{ do parallel}
\]

- All processors \( P_{ijk} (0 \leq i,j,k < 2^l) \) do in parallel

\[
tree[l] \rightarrow \text{node}[l][j][k].child[0] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 0][2k + 0];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[1] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 0][2k + 1];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[2] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 1][2k + 0];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[3] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 0][2j + 1][2k + 1];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[4] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 0][2k + 0];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[5] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 0][2k + 1];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[6] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 1][2k + 0];
\]

\[
tree[l] \rightarrow \text{node}[l][j][k].child[7] := \text{tree}[l + 1] \rightarrow \text{node}[2i + 1][2j + 1][2k + 1];
\]

Also set the parent field for each of the 8 children of \( \text{tree}[l] \rightarrow \text{node}[j][k] \):

\[
\text{tree}[l] \rightarrow \text{node}[l][j][k].base X := X_{\text{min}} + i2^{\text{maxlevel} - l};
\]

\[
\text{tree}[l] \rightarrow \text{node}[l][j][k].base Y := Y_{\text{min}} + j2^{\text{maxlevel} - l};
\]

\[
\text{tree}[l] \rightarrow \text{node}[l][j][k].base Z := Z_{\text{min}} + k2^{\text{maxlevel} - l};
\]

Let \( \text{tree}[l] \rightarrow \text{node}[j][k].weight \) be the sum of the weights of its children;

Let \( \text{tree}[l] \rightarrow \text{node}[j][k].\text{coord}X, \text{tree}[l] \rightarrow \text{node}[j][k].\text{coord}Y, \) and \( \text{tree}[l] \rightarrow \text{node}[j][k].\text{coord}Z \) be the coordinates of the weighted center of the particles contained in the computation box of the current node;

\[
\text{if } \text{tree}[l] \rightarrow \text{node}[j][k].weight == 0 \text{ then}
\]

\[
\text{tree}[l] \rightarrow \text{node}[j][k].\text{isLeaf} := 1;
\]

\[
\text{else} \text{ \ if } \text{tree}[l] \rightarrow \text{node}[j][k].weight == 1 \text{ then}
\]

\[
\text{tree}[l] \rightarrow \text{node}[j][k].\text{isLeaf} := 1;
\]

Let \( \text{tree}[l] \rightarrow \text{node}[j][k].pindex \) be the index of the only particle contained in the current computation box;

\[
\text{endfor}
\]

endfor {1}

Fig. 4. Building the oct-tree bottom-up (part 3).

box as well as the size of the largest computation box, in constant time. By then, we should know an \( \frac{1}{2} \log n + O(1) \) upper bound on the height of the oct-tree. Therefore, we can dynamically allocate space for every possible tree node. We assume that all the fields of a tree node are initialized to zero at the time the memory is allocated. The total space allocated is \( \Theta(n) \) since the number of nodes in a complete oct-tree of height \( \frac{1}{2} \log n + O(1) \) is \( \Theta(n) \).

Instead of inserting the particles to the tree from the root node, we insert the particles directly to the nodes corresponding to the smallest computation boxes. We then pass information from one layer of the tree to the layer above, starting from the bottom layer. Although there are \( \frac{1}{2} \log n + O(1) \) layers of the tree, the amount of time required is decreased by a factor of 8 every time we move up one layer. This is the key to achieving the \( \Theta(n) \) sequential time complexity. We will show that this can be done in \( \Theta(\log n) \) parallel time.
2.3. Analysis of the oct-tree construction algorithm

Note that each node at level $\text{maxlevel}$ in the oct-tree constructed by Algorithm 2.1 has a computation box whose size is $\delta = (\sqrt{3}/3)c_1$. Therefore, the largest Euclidean distance between any two points in such a computation box is smaller than $c_1$. It follows from Assumption 1.1 that no two particles can fall into a computation box of at level $\text{maxlevel}$. Again by Assumption 1.1, all $n$ particles are contained in the computation box of the root node.

Since there are $8^l$ tree nodes at level $l$ of the tree for ($l = 0, 1, \ldots, \text{maxlevel}$), the number of tree nodes required is

$$\sum_{l=0}^{\text{maxlevel}} 8^l = \frac{8^{\text{maxlevel}+1} - 1}{8 - 1} \leq \frac{8^{\text{maxlevel}+1}}{7} \leq \frac{192\sqrt{3}}{7} \left(\frac{c_2}{c_1}\right)^3 n,$$

where the last inequality follows from Assumption 1.1 and the definition of $\delta$ and $\text{maxlevel}$. Therefore our algorithm requires $\Theta(n)$ memory.

In the following, we will analyze the time complexity of each step of Algorithm 2.1. Since there are $n$ particles, the computation of $X_{\text{min}}$, $X_{\text{max}}$, $Y_{\text{min}}$, $Y_{\text{max}}$, $Z_{\text{min}}$, $Z_{\text{max}}$ requires $\Theta(\log n)$ time on a $\Theta(n/\log n)$ processor CREW PRAM. After the above quantities are computed, $\text{maxlevel}$ can be computed in constant time by taking a base 2 logarithm. Therefore Step 1 requires $\Theta(\log n)$ time.

In Step 2, $\Theta(n)$ space is dynamically allocated. We assume that every field of each tree node is initialized to zero. This may take up to $O(\log n)$ time. In Step 3, constant amount of time is spent on each particle. Therefore, a total of $\Theta(n)$ sequential time is required in this step. This $\Theta(n)$ time is evenly distributed among $\Theta(n/\log n)$ processors. Therefore Step 3 requires $\Theta(\log n)$ parallel time. Note that the computation of the gravitational center of the particles contained in the computation box can be done in constant time once we know the field weight and the gravitational center for each of its 8 children. Therefore, constant amount of time is spent on each tree node in Step 4 and Step 5. In Step 4, all $P$ processors are busy, with each processor processing $8^{L-l}$ tree nodes. Therefore the parallel run time of Step 4 is $\Theta(\sum_{l=\text{maxlevel}-1}^{L} 8^{L-l}) = \Theta(n/P) = \Theta(\log n)$. In Step 5, only part of the $P$ processors are busy, with processor $P_{000}$ as the busiest, processing $L$ tree nodes. Therefore, the parallel run time of Step 5 is $\Theta(L) = \Theta(\log P) = \Theta(\log n)$. To summarize, we have proved the following theorem.

**Theorem 2.1.** Algorithm 2.1 builds the oct-tree for $n$ particles using $\Theta(\log n)$ time and $\Theta(n)$ space, on a $\Theta(n/\log n)$ processor CREW PRAM, provided that the particles satisfy Assumption 1.1. The constant behind the asymptotic notation is proportional to $(c_2/c_1)^3$.

Note that although we have allocated space for a full oct-tree, the actual oct-tree may not be a full oct-tree in most cases. As a result, the leaf nodes of the oct-tree may be at different levels of the tree. Note also that under the same assumption on the distribution
of the particles, the top-down construction of the oct-tree requires $\Theta(n \log n)$ sequential time \cite{3,4}. In our bottom-up construction, we are allocating space for some tree nodes which will never be used (i.e., the descendants of a node whose weight is 1). We should note that the asymptotic memory requirements for the top-down algorithm and the bottom-up algorithm are both $\Theta(n)$. However, the time complexity of the bottom-up algorithm is a $\Theta(\log n)$ factor lower than that of the top-down algorithm. In the next section, we will show that Appel’s algorithm for computing force field of a cluster of $n$ particles can be implemented in $\Theta(\log n)$ parallel time after the oct-tree is constructed. Therefore, the improved time complexity of the construction of the oct-tree has great impact on the simulation of large clusters.

3. Computing force fields in $\Theta(\log n)$ time

Given a cluster of $n$ particles, we need to compute the potential energy function and the force exerted on each particle by the other particles. In many applications, the potential energy function of a cluster is the sum of the pair-wise potential functions.

Let $p_1$ and $p_2$ be the positions of two particles of unit charge each, the gravitational potential between this pair of particles is defined by

$$f_G(p_1, p_2) = \frac{\sigma_1}{\|p_2 - p_1\|},$$

where $\|\cdot\|$ stands for the Euclidean norm and $\sigma_1$ is a given positive constant. This potential is widely used in astrophysical simulations. The Lennard–Jones potential function between this pair of particles is defined by

$$f_{LJ}(p_1, p_2) = \frac{\sigma_2}{\|p_2 - p_1\|^{12}} - \frac{\sigma_3}{\|p_2 - p_1\|^{6}},$$

where $\|\cdot\|$ stands for the Euclidean norm and $\sigma_2$ and $\sigma_3$ are given positive constants. This potential energy function is widely used in molecular conformation, molecular dynamics and protein folding. In both cases, the potential energy function approaches zero as the distance between the two particles approaches to infinity. In both cases, the “force” exerted on $p_1$ by $p_2$ is computed as the negative of the gradient of the potential energy function with respect to $p_1$. These are given by

$$-\frac{\sigma_1}{\|p_2 - p_1\|^3}(p_2 - p_1)$$

and

$$\left(\frac{-12\sigma_2}{\|p_2 - p_1\|^{14}} - \frac{-6\sigma_3}{\|p_2 - p_1\|^8}\right)(p_2 - p_1)$$

for the gravitational pair potential and the Lennard–Jones pair potential, respectively.

Since there are $n(n-1)/2$ pairs for a cluster of $n$ particles, conventional algorithm for computing the potential energy function and force field requires $\Theta(n^2)$ time.
In 1985, Appel [3] proposed a divide and conquer algorithm for computing the force field of a gravitational cluster. His algorithm has a proved time complexity of $\Theta(n \log n)$. Appel’s algorithm is based on the following idea: given two clusters of particles consisting of $n_1$ and $n_2$ unit weight particles each, there are $n_1 \times n_2$ particles pairs with one particle from the first cluster and the other particle from the other cluster. If the two clusters are well separated (i.e., the ratio of the maximum diameter of the clusters over the distance between the clusters is small), we may consider the first cluster as a big particle located at the gravitational center $P_1$ of the first cluster with a mass of $n_1$ and consider the second cluster as another big particle located at the gravitational center $P_2$ of the second cluster with a mass of $n_2$. We may then approximate the total interactions between particles from the first cluster and particles from the second cluster by the following weighted pair potential:

$$F_G(P_1, n_1; P_2, n_2) = n_1 n_2 \frac{\sigma_1}{\|P_2 - P_1\|^3}.$$  \hfill (3.5)

The negative of the gradient of the above function with respect to $P_1$ is

$$n_1 n_2 \frac{-\sigma_1}{\|P_2 - P_1\|^3} (P_2 - P_1).$$  \hfill (3.6)

Therefore, the force exerted on each particle in the first cluster by all the particles in the second cluster can be approximated by

$$n_2 \frac{-\sigma_1}{\|P_2 - P_1\|^3} (P_2 - P_1),$$  \hfill (3.7)

since there are $n_1$ particles in the first cluster. Similarly, the force exerted on each particle in the second cluster by all the particles in the first cluster can be approximated by

$$n_1 \frac{-\sigma_1}{\|P_2 - P_1\|^3} (P_1 - P_2).$$  \hfill (3.8)

In the case where Lennard–Jones potential energy function is used, the force exerted on each particle in the first cluster by all the particles in the second cluster can be approximated by

$$n_2 \left( \frac{-12 \sigma_2}{\|P_2 - P_1\|^{14}} - \frac{-6 \sigma_3}{\|P_2 - P_1\|^{8}} \right) (P_2 - P_1)$$  \hfill (3.9)

and the force exerted on each particle in the second cluster by all the particles in the first cluster can be approximated by

$$n_1 \left( \frac{-12 \sigma_2}{\|P_2 - P_1\|^{14}} - \frac{-6 \sigma_3}{\|P_2 - P_1\|^{8}} \right) (P_1 - P_2).$$  \hfill (3.10)

In this way, we can spend constant time to compute an approximation to the potential that requires $O(n_1 \times n_2)$ time in the conventional method. If the ratio of the maximum of the radii of the clusters over the distance between the clusters is $\delta$, the relative error in this approximation is $O(\delta^2)$ [3].
The performance of Appel’s algorithm depends on the parameter which defines well separateness. If this parameter is close to 0, we have more accuracy but need more computing time. If this parameter is close to 1, we have less accuracy but need less computing time. A generic description of Appel’s recursive top-down procedure can be found in [3, 6, 16]. In the next section, we will present an equivalent non-recursive bottom-up force field evaluation procedure.

3.1. The two-pass algorithm

After the oct-tree is constructed, Appel’s algorithm can be implemented using a bottom-up pass and a top-down pass. We assume that there is a global variable \( \text{FUNC} \) which is initialized to 0 and is used to accumulate the potential energy function of the cluster. We also assume that the fields \( \text{force}_X \), \( \text{force}_Y \) and \( \text{force}_Z \) at every tree node are all initialized to 0 before the computation. These fields are used to hold partial values of the force field during the computation.

For any two nodes \( A \) and \( B \) in the tree, a call to procedure \( \text{compGRAD}(A, B) \) does the following:

- Compute the force exerted on each particle in \( A \) by all particles in \( B \) and save the value in \( (A.\text{force}_X, A.\text{force}_Y, A.\text{force}_Z) \). The computation is done according to either (3.7) if the gravitational potential is used or (3.9) if the Lennard–Jones potential is used.
- Compute the potential between cluster \( A \) and cluster \( B \) and add this value to the global variable \( \text{FUNC} \). The computation is done according to either (3.5) if the gravitational potential is used or (3.1) if the Lennard–Jones potential is used.

Our algorithm is presented as Algorithm 3.1.

During each call to procedure \( \text{compGRAD}(A, B) \), the potential function between cluster \( A \) and cluster \( B \) is added to the global variable \( \text{FUNC} \). The force exerted on each particle in cluster \( A \) by the particles in cluster \( B \) is stored in \( (A.\text{force}_X, A.\text{force}_Y, A.\text{force}_Z) \). Note that the force exerted on each particle in cluster \( B \) by the particles in cluster \( A \) is computed in the call to \( \text{compGRAD}(B, A) \). Therefore, at the end of the computation, \( \text{FUNC} \) is 2 times the actual potential function value and \( (A.\text{force}_X, A.\text{force}_Y, A.\text{force}_Z) \) is the force exerted by all the other particles on the particle in node \( A \) for each leaf node \( A \) whose weight is 1. This force is exactly the force computed by Appel’s algorithm [3, 16].

3.2. Time complexity

We will analyze the time complexity of Algorithm 3.1. Given any parameter \( \delta > 0 \) which defines well separateness and a tree node \( A \), the number tree nodes which are on the same level as \( A \) and which are not well separated from \( A \) is bounded by \( O(1/\delta^3) \), which is a constant for any given \( \delta \). As a result, the innermost for loop in Step_1 of Algorithm 3.1 requires constant time. Therefore, the parallel run time of Step_1 of Algorithm 3.1 is \( \Theta(\sum_{l=\max\text{level}}^L 8^{l-1}) = \Theta(n/P) = \Theta(\log n) \). Similarly, we can show that the parallel run time of Step_4 of Algorithm 3.1 is also \( \Theta(\log n) \). In Step_2, only
Algorithm 3.1 (part 1) {Computing force field in two passes.}

Step_1 {Gathering information bottom-up}

\[
\text{for}\ l := \text{maxlevel} \text{ downto } L \text{ do}
\]
\[
\text{All processors } P_{ijk} \ (0 \leq i, j, k < 2^l) \text{ do in parallel}
\]
\[
\text{for}\ i := J2^L - L + 2^L - 1 \text{ downto } I2^L - L - 1 \text{ do}
\]
\[
\text{for}\ j := J2^L - L + 2^L - 1 \text{ downto } I2^L - L - 1 \text{ do}
\]
\[
\text{for}\ k := K2^L - L + 2^L - 1 \text{ downto } I2^L - L - 1 \text{ do}
\]
\[
\text{if} \ \text{tree}[l] \rightarrow \text{node}[i][j][k] \cdot \text{weight} \geq 1 \text{ then}
\]
\[
\text{Let } A \text{ be } \text{tree}[l] \rightarrow \text{node}[i][j][k].
\]
\[
\text{for}\ \text{all tree node } B \text{ on level-} l \text{ of the tree such that}
\]
\[
(1) \ B \cdot \text{weight} \geq 1; (2) A \text{ and } B \text{ are well separated;}
\]
\[
(3) \text{the parents of } A \text{ and } B \text{ are not well separated do}
\]
\[
\text{compGRAD}(A, B);
\]
\[
\text{endfor}
\]
\[
\text{endfor} \{k\}
\]
\[
\text{endfor} \{j\}
\]
\[
\text{endfor} \{i\}
\]
\[
\text{endfor} \{l\}
\]

Fig. 5. Computing force field in two passes (part 1).

part of the \( P \) processors are active. The parallel runtime of Step_2 of the algorithm is \( \Theta(L) = \Theta(\log P) = \Theta(\log n) \). Similarly, we can show that the parallel run time of Step_3 of Algorithm 3.1 is also \( \Theta(\log n) \). To summarize, we have proved the following theorem.

**Theorem 3.1.** Given a cluster of \( n \) particles satisfying Assumption 1.1, the force field can be computed using Algorithm 3.1 in \( \Theta(\log n) \) time, on a \( \Theta(n/\log n) \) processor CREW PRAM.

It is clear that Algorithms 2.1 and 3.1 yield a new linear time algorithm for computing force field for a cluster of \( n \) particles which are almost homogeneously distributed. In [5], Callahan and Kosaraju proved that a size \( \Theta(n) \) sequence of well-separated decomposition can be computed in \( \Theta(n) \) time once a fair-split tree is constructed.
Algorithm 3.1 (part 2) {Computing force field in two passes.}

Step 6 {Pushing information top-down}

for \( l := 1 \) to \( L \) do
  for every child node \( B \) of \( A \) such that \( B \) weight \( \geq 1 \) do
    \( B \) force\( X = B \) force\( X + A \) force\( X \);
    \( B \) force\( Y = B \) force\( Y + A \) force\( Y \);
    \( B \) force\( Z = B \) force\( Z + A \) force\( Z \);
  endfor
endfor \{1\}

Step 4 {Pushing information top-down}

for \( l := L + 1 \) to maxlevel \(- 1 \) do
  for \( j := J^{2^L} \) to \( J^{2^L} + 2^{L-1} - 1 \) do
    for \( k := K^{2^L} \) to \( K^{2^L} + 2^{L-1} - 1 \) do
      if \( \text{tree}[l] \rightarrow \text{node}[i][j][k] \) weight \( \geq 1 \) then
        let \( A \) be \( \text{tree}[l] \rightarrow \text{node}[i][j][k] \).
        for every child node \( B \) of \( A \) such that \( B \) weight \( \geq 1 \) do
          \( B \) force\( X = B \) force\( X + A \) force\( X \);
          \( B \) force\( Y = B \) force\( Y + A \) force\( Y \);
          \( B \) force\( Z = B \) force\( Z + A \) force\( Z \);
        endfor
      endif
    endfor
  endfor
endfor \{l\}

Fig. 6. Computing force field in two passes (part 2).

A fair-split tree for \( n \) particles can be constructed in \( \Theta(n \log n) \) time using the algorithm of [5], without any restriction on the distribution of the particles. However, Algorithm 3.1 is the first \( n \) \( \Theta(n \log n) \) time algorithm using \( \Theta(n/ \log n) \) processors.

**Remark 3.1.** Assumption 1.1 is essential to the \( O(n) \) time oct-tree construction algorithm. Without any assumption on the distribution of the particles, \( \Omega(n \log n) \) is lower bound on the construction of the oct-tree for \( n \) particles. Consider the case where all \( n \) particles lie on the \( X \)-axis. Suppose that one can construct the oct-tree of \( n \) particles in \( T(n) \) time, then the number of tree nodes is \( \Omega(n \log n) \). Taking an in-order traversal of the tree sorts the \( n \) particles. Since the in-order traversal takes \( \Theta(T(n)) \) time, this shows that \( \Theta(T(n)) = \Omega(n) \), under the algebraic comparison tree model. Which shows that \( T(n) = \Omega(n) \).
4. Conclusions

In this paper, we have presented a \( \Theta(\log n) \) time algorithm for computing force field in \( n \)-body simulations on a \( \Theta(n/\log n) \) processor CREW PRAM. A key to this improved complexity is an \( O(n) \) time bottom-up construction of the oct-tree which was constructed top-down using \( O(n \log n) \) time in previous studies. We have also replaced the traditional recursive top-down force field computation with a non-recursive bottom-up computation method. We have also studied the dependency of the constant behind the asymptotic notation on the distribution parameters \( c_1 \) and \( c_2 \) and on the well-separateness parameter \( \delta \). This analysis is important because good software for these evaluations is badly needed in practice. Computational studies of the proposed algorithm on existing architectures will be reported in a forthcoming paper.

References