A (somewhat) faster tree traversal algorithm for finding neighbors

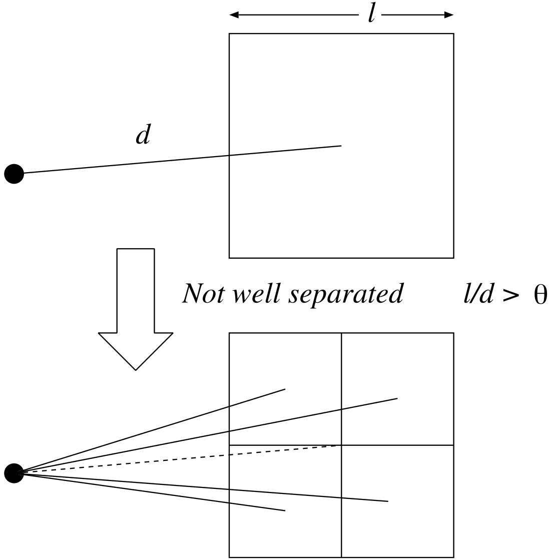
J. Makino

Internal Seminar, March 10, 2021

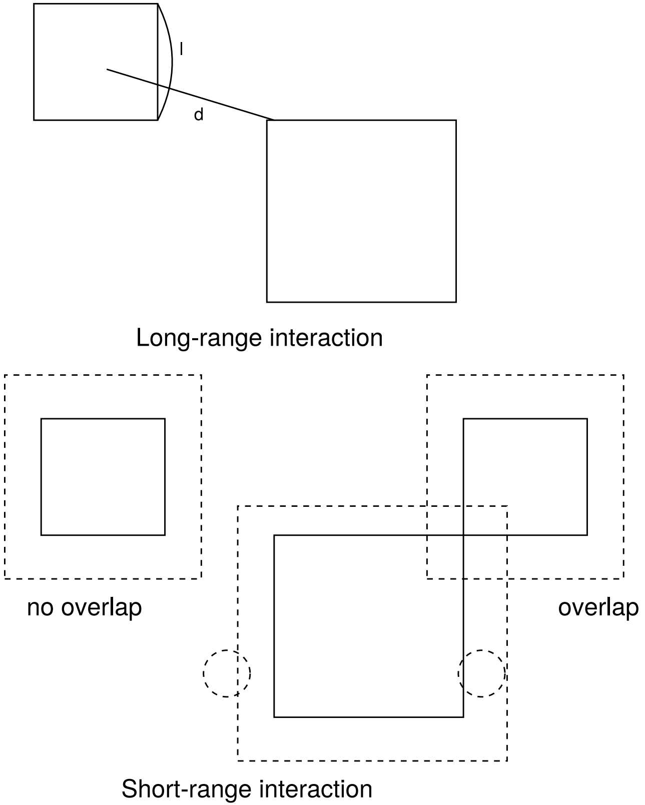
Overview

- Tree traversal
- Performance of tree traversal on various CPUs
- A somewhat faster scheme
- Measurement result

Tree traversal



box-particle condition



box-box condition

Tree traversal for short-range interactions

The condition for “well-separated” is different from that for long-range interactions.

- Long-range: Opening angle
- Short-range: overlap (extended box which covers all neighbor spheres for particles in that box)

Performance of tree traversal on various CPUs

```
code: https://github.com/jmakino/C-tree (nbtest)
./nbtest -i hom1M.stoa -N 128 -n 128 -t 0.03
(hom1M.stoa made with NEMO mkhomsph)
```

CPU	time(s)
Core i7-1065G7 (HP note)	0.3126
Skylake	0.4885
Threadripper	0.2674
FX700/g++	1.869
FX700/FCC(clang)	0.816
FX700/FCC(trad)	2.576
HP apollo70(g++)	0.8435

Caveats

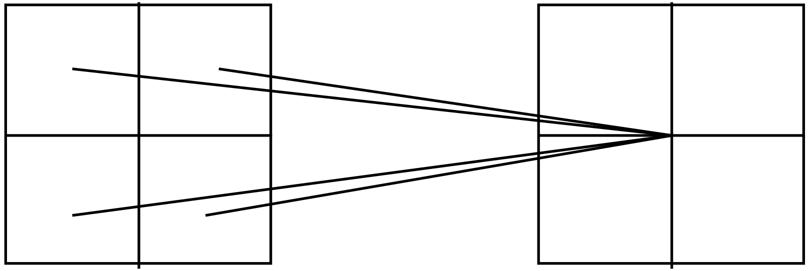
- All measurements are on single core
- x86 clocks are thus at their highest (2.5?, 3.4, 4.2GHz for Skylake, Core i7 and Threadripper)
- FX700 and Apollo70 clocks are fixed at 2.0 and 2.2GHz

Per-core performance of FX700

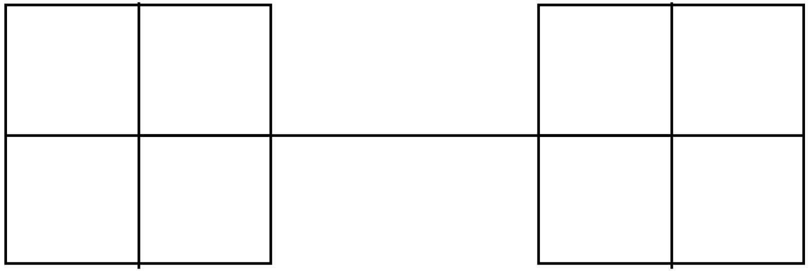
- In this test, the average length of the list is 218, and the number of lists is 30250 (shared by around 33 particles). So the speed to make list is 10M particles/sec.
- theoretical peak performance is (for single prec) 128Gflops. Even with 25% of the peak, we can still perform around 100M interactions/sec (if number of FP ops/interaction is around 300).
- List construction does take significant fraction of the total time

A somewhat faster scheme

- Usual scheme: do treewalk for each box
- “New” algorithm: omit distant nodes at higher level Theoretically $O(\log N) \rightarrow O(1)$



Usual treewalk: each box determines if the other box is well separated or not



Improved treewalk: If not overlapped at higher level, that box will be skipped for all children

Practical considerations

- We need to implement "double recursive" algorithm. When to go down the tree of which side? (source or destination)

One possible algorithm (not necessarily the best)

```
function dual_walk(src, dest)
  if (src and dest are not overlapped) return
  if (dest has more than nglimit particles)
     if (src has less than nllimit particles) or (src is smaller than dest)
        go down for dest
     else
        go down for src
  else
     if (src has less than nlimit particles)
        check and add particles in src to nblist of dest
     else
        go down for src
```

Code for decision making

```
int src_down = 0; int dest_down = 0;
if(!are_overlapped_with_cutoff(this,&source_node, cutoff)){
    return;
}
if(nparticle > ncrit){
    if (source_node.is_leaf()|source_node.nparticle < nplimit){
        dest_down = 1;
    }else{
        if (srclevel < destlevel){
            src_down = 1;
        }else{
            dest_down = 1;
        }
    }
}else{
    if ((!source_node.is_leaf() &&(source_node.nparticle < nplimit ))
        &&(this != &source_node)) src_down = 1;
}
```

Measured Performance

CPU	original(s)	new
Core i7-1065G7 (HP note)	0.3126	0.2339
Skylake	0.4885	0.3336
Threadripper	0.2674	0.2068
FX700/g++	1.869	1.225
FX700/FCC(clang)	0.816	0.6889
FX700/FCC(trad)	2.576	2.117
HP apollo70(g++)	0.8435	0.5740

Results

- 15-35% reduction in time
- Unfortunately, the improvement is the smallest on FX700(clang)...
- On FX700, the main bottleneck is the judgment for individual particles. Need to accelerate this part.

Summary

- Tried to improve the performance of tree traversal for neighbor list
- Implemented dual-walk algorithm.
- Improvement from original algorithm: 15-35% reduction in time. Unfortunately smallest on FX700(clang)
- For this treewalk, FX700 compiler in trad mode generates the code 3X slower than that in clang mode.
- Compared to x86 architecture, clock-normalized performance of A64fx for this tree walk is around 60% (with Clang mode).