

Use of the X10 language to implement the Fast Multipole Algorithm

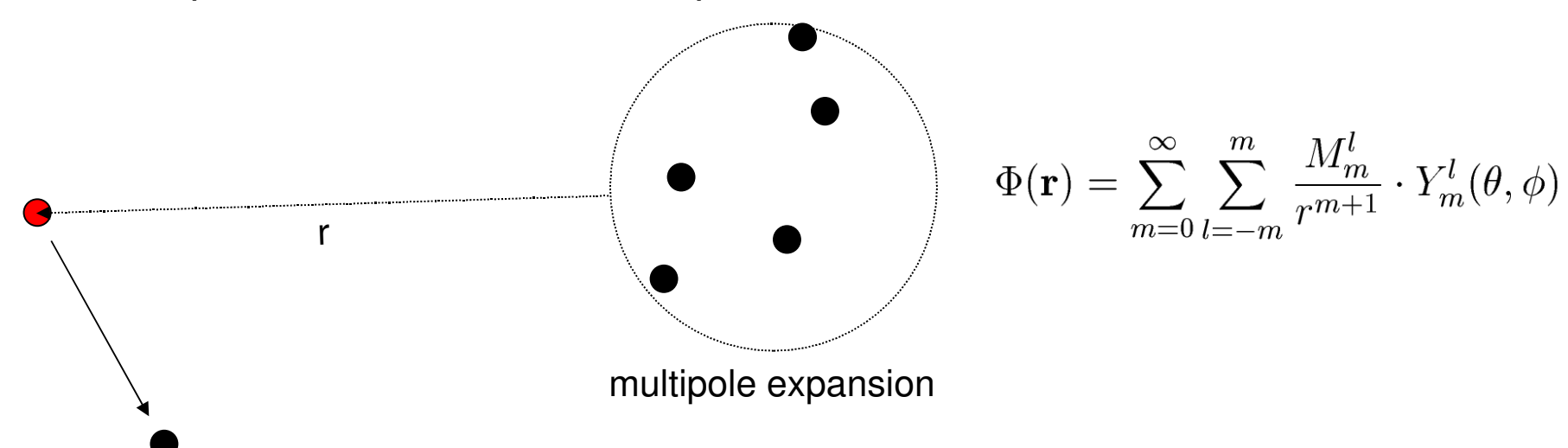


The Fast Multipole Algorithm

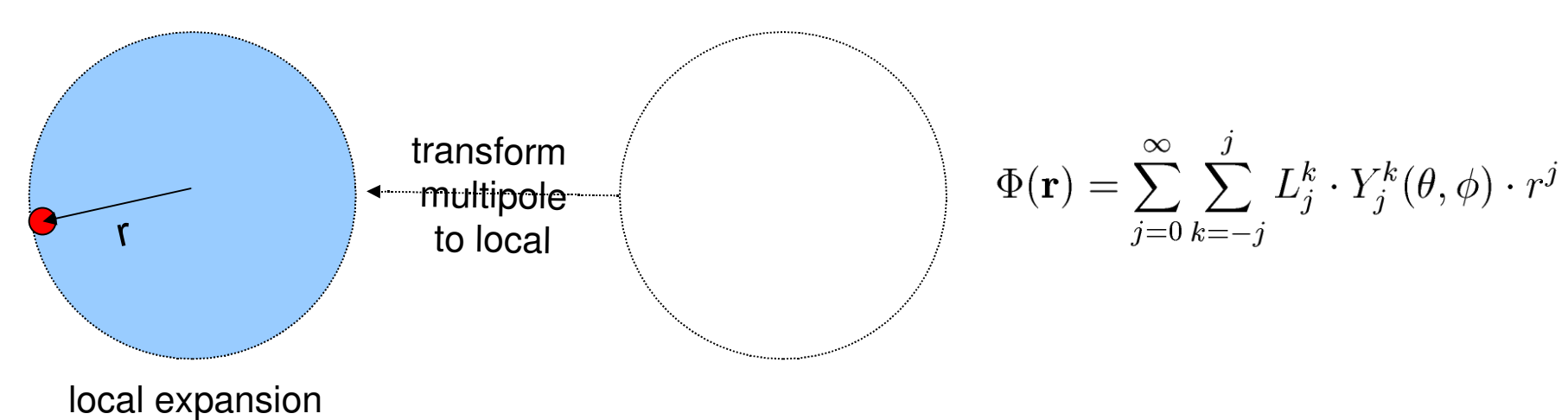
Evaluating the energy of a system of N bodies interacting via a pairwise potential is naïvely an $O(N^2)$ problem. The Fast Multipole Method^[1] allows approximation to a specified level of accuracy in only $O(N)$ time.

Expansions

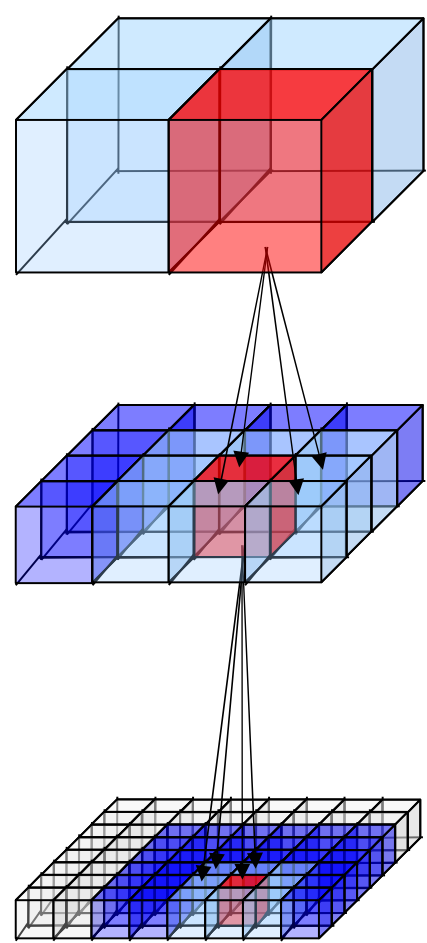
A multipole expansion approximates the potential at the origin due to particles within a sphere centered at a distant point r .



A local expansion approximates the potential at any point r within a sphere centered at the origin, due to distant particles.



Hierarchical division of space



3D space is divided into an octree of cubic boxes. (Only 4 of 8 boxes shown here.)

For the red box:

- Multipole expansions used for every box in interaction list (dark blue)
- Non-well-separated boxes (light blue) evaluated at a lower level in the tree
- More remote boxes (white) are evaluated at a higher level

Algorithm

- 1) Create the octree of boxes
- 2) Create multipole expansions for each box at the lowest level
- 3) Combine expansions for child boxes into an expansion for each parent box (upward pass)
- 4) Transform multipole expansion for each box to local expansion for all boxes in interaction list; translate parent expansion to each child box (downward pass)
- 5) At lowest level, use local expansion for each box to calculate potential and force due to all remote particles; use direct pairwise evaluation for near particles

References

- [1] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73, (1987), pp. 325-348
- [2] O. Coulaud, P. Fortin and J. Roman, *Hybrid MPI-Thread parallelization of the Fast Multipole Method*, ISPC '07 (2007)
- [3] P. Charles et al., *X10: an object-oriented approach to non-uniform cluster computing*, OOPSLA (2005)



Josh Milthorpe & Alistair Rendell
{josh.milthorpe,alistair.rendell}@anu.edu.au
Computer Systems Group
College of Engineering & Computer Science

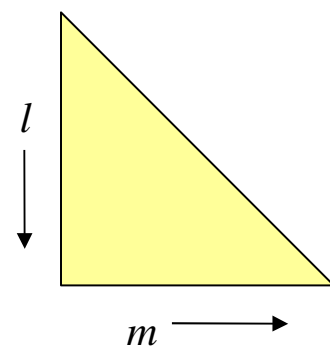
X10 implementation

Data representation

X10's array sublanguage permits succinct representation of key polynomials.

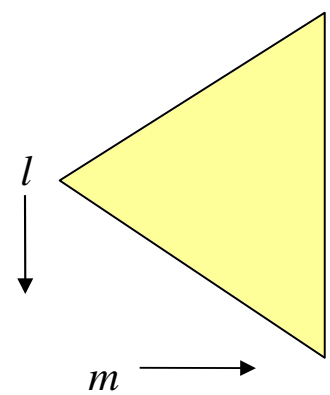
Expansions are defined in terms of associated Legendre polynomials P_m truncated at p terms, where $0 \leq l \leq m \leq p$

```
val Plm = Array.make[Double] (
  Region.makeLowerTriangular(p+1));
```



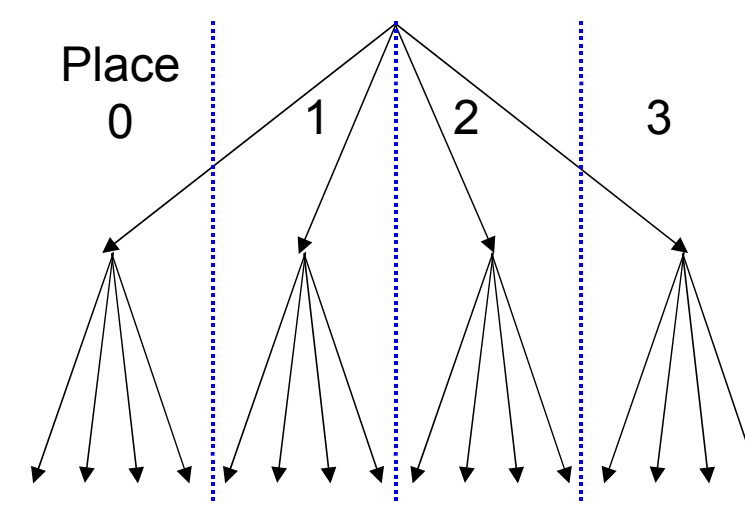
Multipole expansions O_m have unusually shaped regions $0 < m \leq p$; $abs(l) \leq m$

```
public class ExpansionRegion
  extends BaseRegion{self.rank==2} {
  global val p : Int;
  ...
}
this.terms = Array.make[Complex] (
  new ExpansionRegion(p),
  (Point) => Complex.ZERO);
```



Distribution

Current implementation uses block distribution of lowest level boxes to all places.



There are more efficient options for dividing the simulation space (e.g. costzones, Hilbert / Morton ordering)^[2].

How can X10 *distributions* be used to succinctly represent these options?

Most communication is between nearby boxes in the 3D simulation space.

Is it possible to improve performance by assigning nearby boxes to nearby (lower communication latency) *places*?

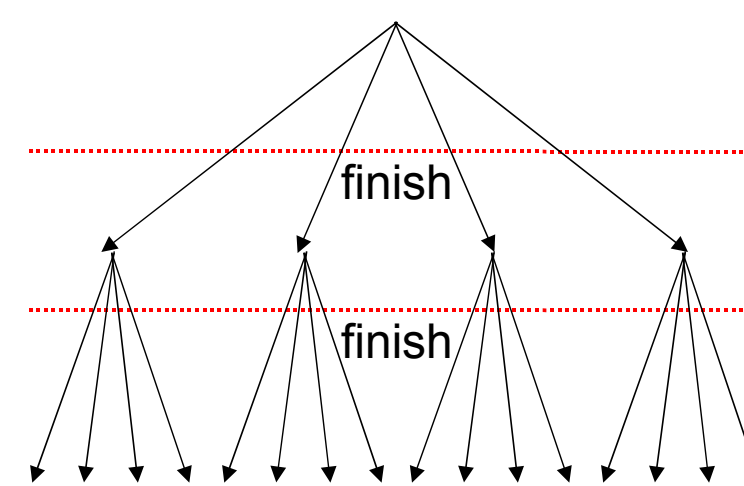
Remote memory access

Ability to (asynchronously) access remote data^[3] simplifies expression of data dependencies, compared to message passing (e.g. MPI):

```
// start retrieve of data from box2
val box2MultipoleExp = future(box2.location) box2.multipoleExp;
... // perform other computation
// use retrieved data
box1.localExp.transformAndAddToLocal (
  transform2l, box2MultipoleExp());
... but OpenMP is still simpler (no data locality)!
```

Synchronization

Simple: *finish* one level at a time... but this is too conservative.



Dependencies:

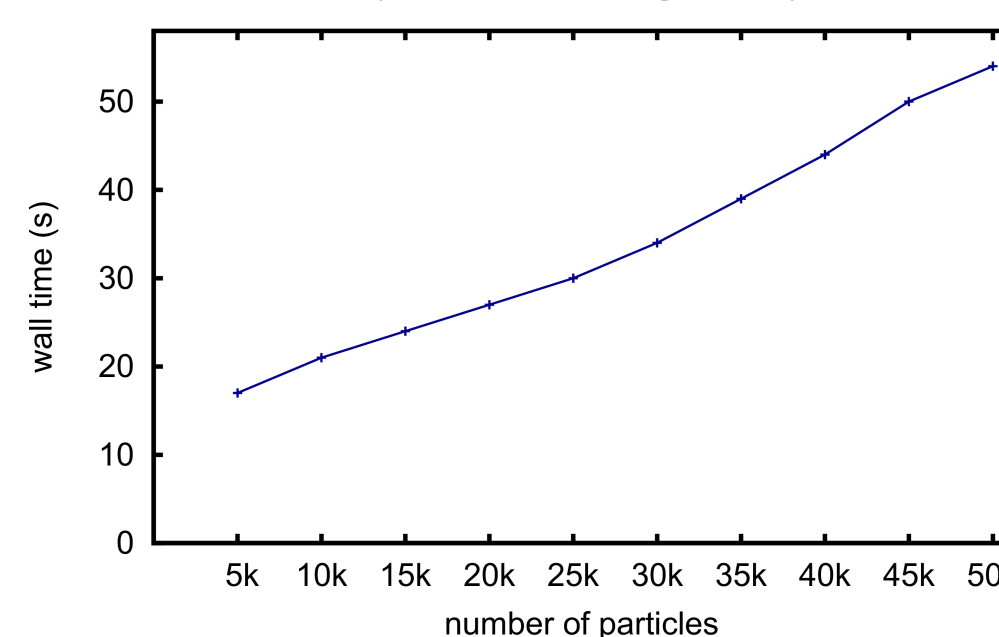
- Upward pass: parent depends on ≤ 8 child boxes
- Downward pass: each box depends on interaction list (≤ 189 boxes for $ws = 1$) and parent

Majority of time is spent in downward pass (multipole-to-local).

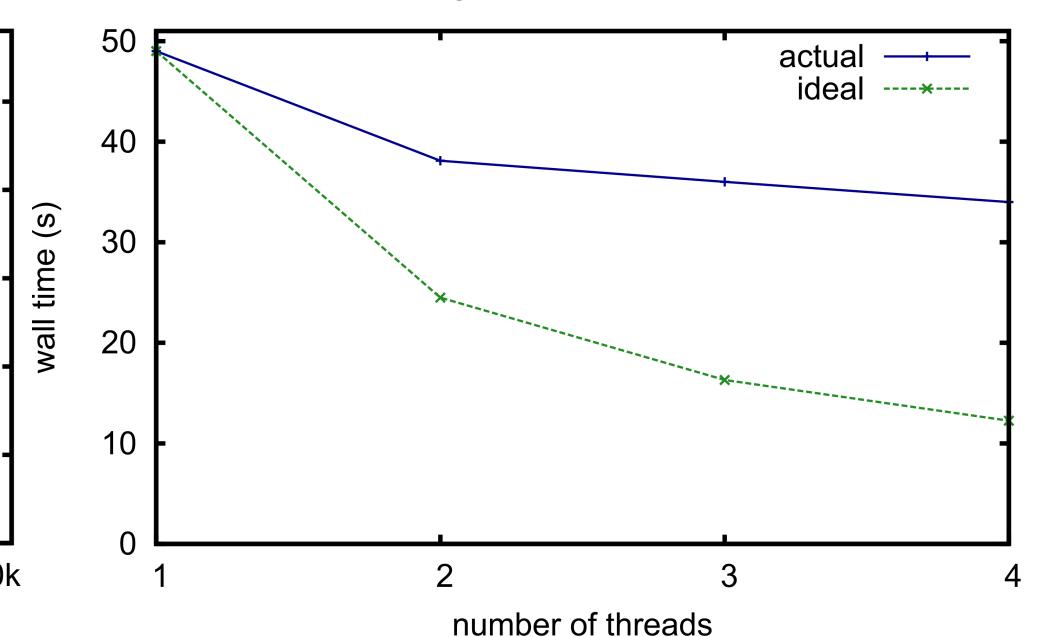
Can performance be improved by only synchronizing on dependencies?

Results

FMM potential evaluation - scaling with number of particles (Intel Core2 Q6600@2.4GHz)



FMM potential evaluation for 30K particles scaling with number of threads



Issues with X10 2.0

- Performance of array access
- Performance of complex arithmetic (may require *semantic expansion*)
- No distributed garbage collection \rightarrow memory leaks