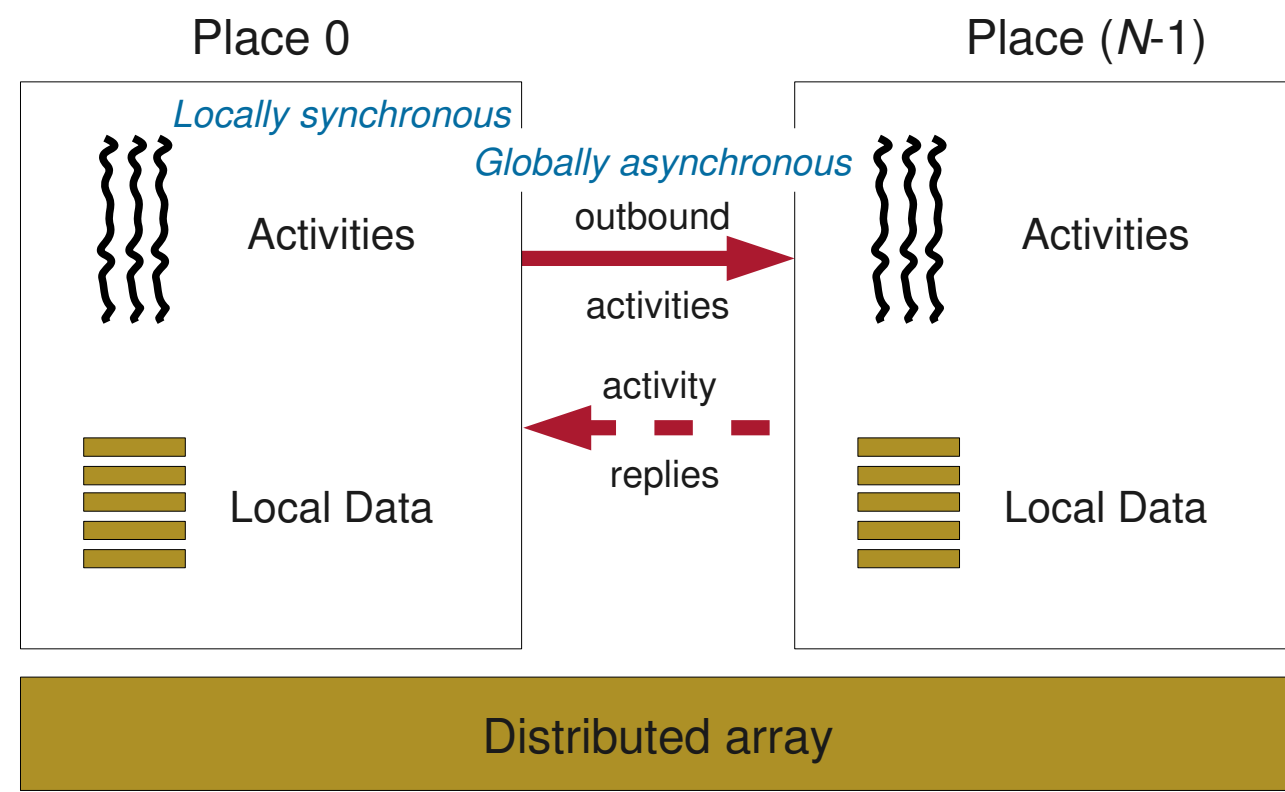


# Scalable scientific programming with the X10 language

Josh Milthorpe - Computer Systems Group - josh.milthorpe@anu.edu.au - Supervisor: Alistair Rendell

## The X10 programming language

X10 is an Asynchronous Partitioned Global Address Space (APGAS) language. It aims to address the architectural challenges of emerging HPC systems by reifying locality in the concept of **places**, and supporting asynchronous **activities** and generalised **synchronisation** and **atomicity** constructs.



Creation and termination detection for a set of distributed activities: Synchronising a set of activities:

```

finish {
  val x = ...;
  async compute(x);

  async at(place1) {
    val y = f(x);
    compute(y);
  }
  async at(place2) {
    val z = g(y);
    ...
  }
}

finish {
  val c1 = Clock.make();
  for (point in region)
    async clocked(c1) {
      compute1();
      Clock.advanceAll();
      compute2();
    }
}
    
```

## ANUChem: computational chemistry applications in X10

We have developed a suite of scientific applications to test **expressiveness**, **performance** and **scalability** of X10.

- **Hartree-Fock** (HF) – 4200 LOC
- **Particle-Mesh Ewald** (PME) – 1000 LOC
- **Fast Multipole Method** (FMM) – 2800 LOC

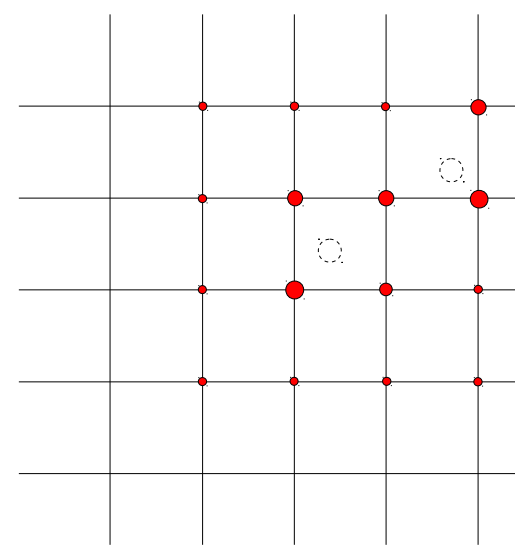
Our experiences have been shared with the language designers, motivating improvements including:

- **arrays**: statically distinguish local vs. distributed
- **complex math**
- **performance**: loops, object serialization, distributed array operations (see right)
- **object model**: global references

## Smooth particle mesh Ewald method

Used in **molecular dynamics** (MD) for long-range electrostatics, reducing cost from  $O(n^2)$  to  $O(n \log n)$ .

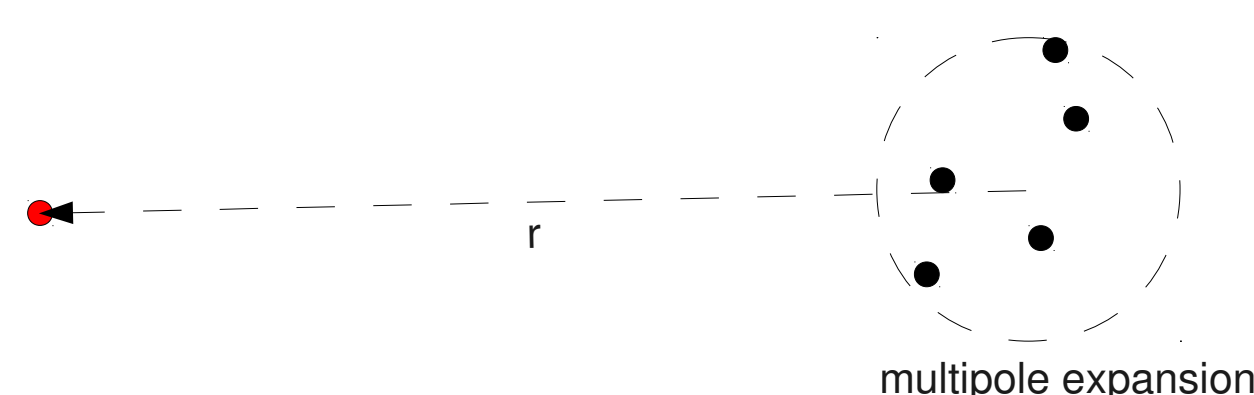
The interaction is split into short-range and long-range components. The **long-range** component is approximated by **interpolating** charges on a **mesh** of grid points, and evaluation proceeds via forward and reverse fast **Fourier transform** (FFT) and convolution. The **short-range** component is computed directly with a cutoff.



## Fast multiple method

Also used for electrostatics in MD, reducing cost from  $O(n^2)$  to  $O(n)$ .

This method divides the simulation space into an **octree** of cubic boxes. Interactions between particles in nearby boxes are evaluated directly; distant interactions are treated by means of **multipole expansions** around box centers.



Characterized by complex data structures (**octrees** – see right) and localized communication patterns.

## Scalable data structures and operations

**Goal**: to develop scalable data structures and high-level operations to support asynchronous partitioned global address space (APGAS) programming of scientific applications.

- Locality preservation
- Heterogeneity
- Adaptive parallelism
- Irregular problems

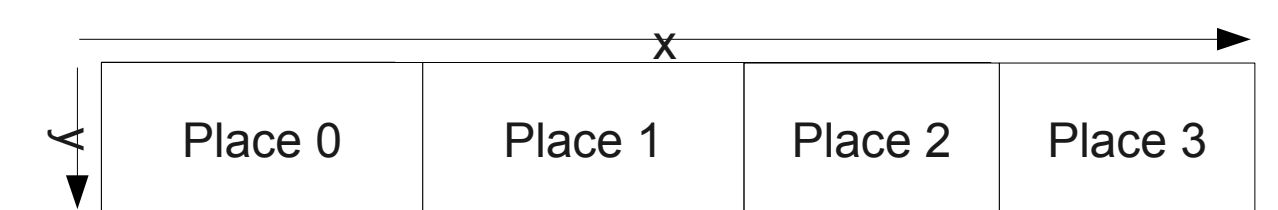
## Distributed array

Distributed arrays are the primary **work-sharing** construct in PGAS languages.

**Array**: maps index Point  $\rightarrow$  data

**Region**: set of index Points

**Distribution**: maps Point  $\rightarrow$  Place



```

val r : Region(2) = 0..15 * 0..3;
val d : Dist(2) = Dist.makeBlock(r);
val da = DistArray.make[Double](d);

ateach([i,j] in d) { // distributed parallel loop
  da(i,j) = ...;
}
    
```

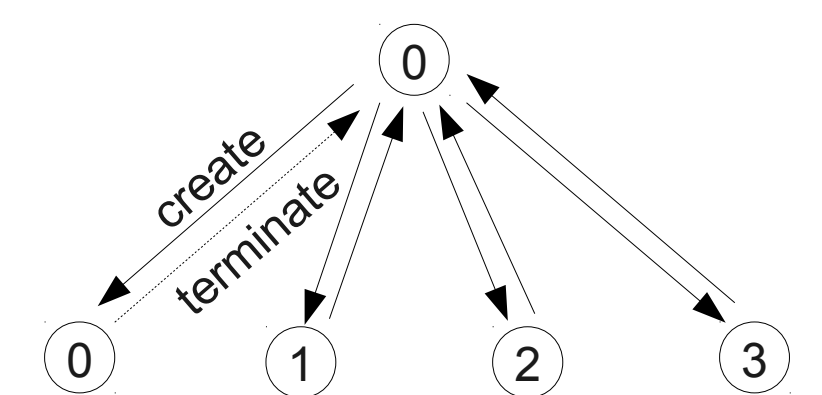
## Implementing ateach

`finish ateach(p in dist) S`  
 "Execute S in parallel locally at every point"

Current implementation [ $O(p)$ ]:

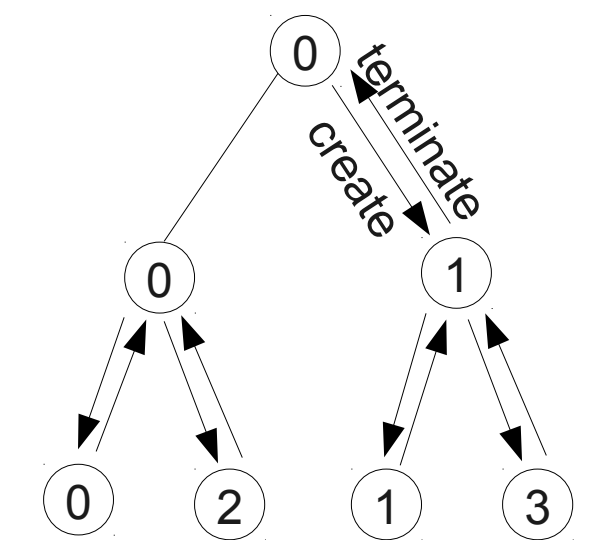
```

for(place in dist.places())
  async at(place)
    for (p in dist(here))
      async S
    
```



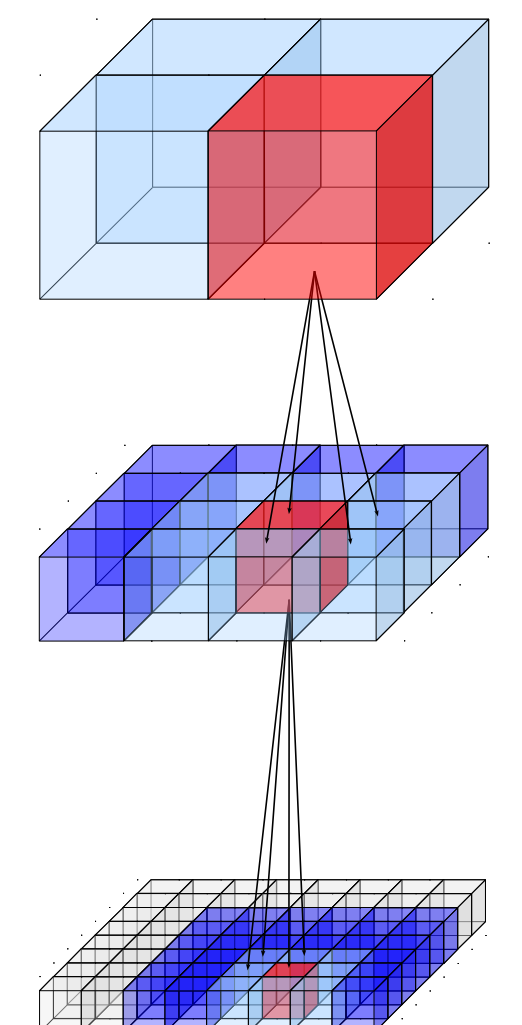
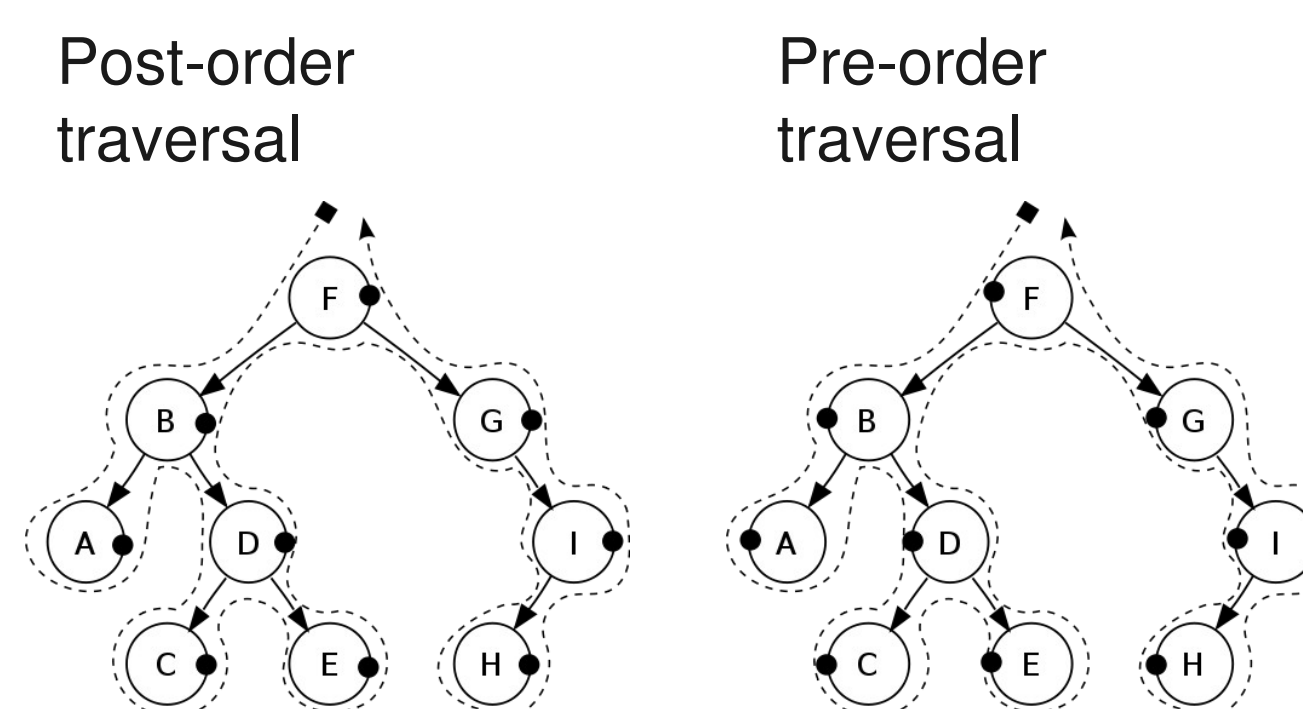
A better possible implementation [ $O(\log(p))$ ]:

1. broadcast active message  
`[ateach,environment(S)] to dist.places()`
2. execute S over all points in `dist(here)`
3. gather or reduce `finish` to initiating place

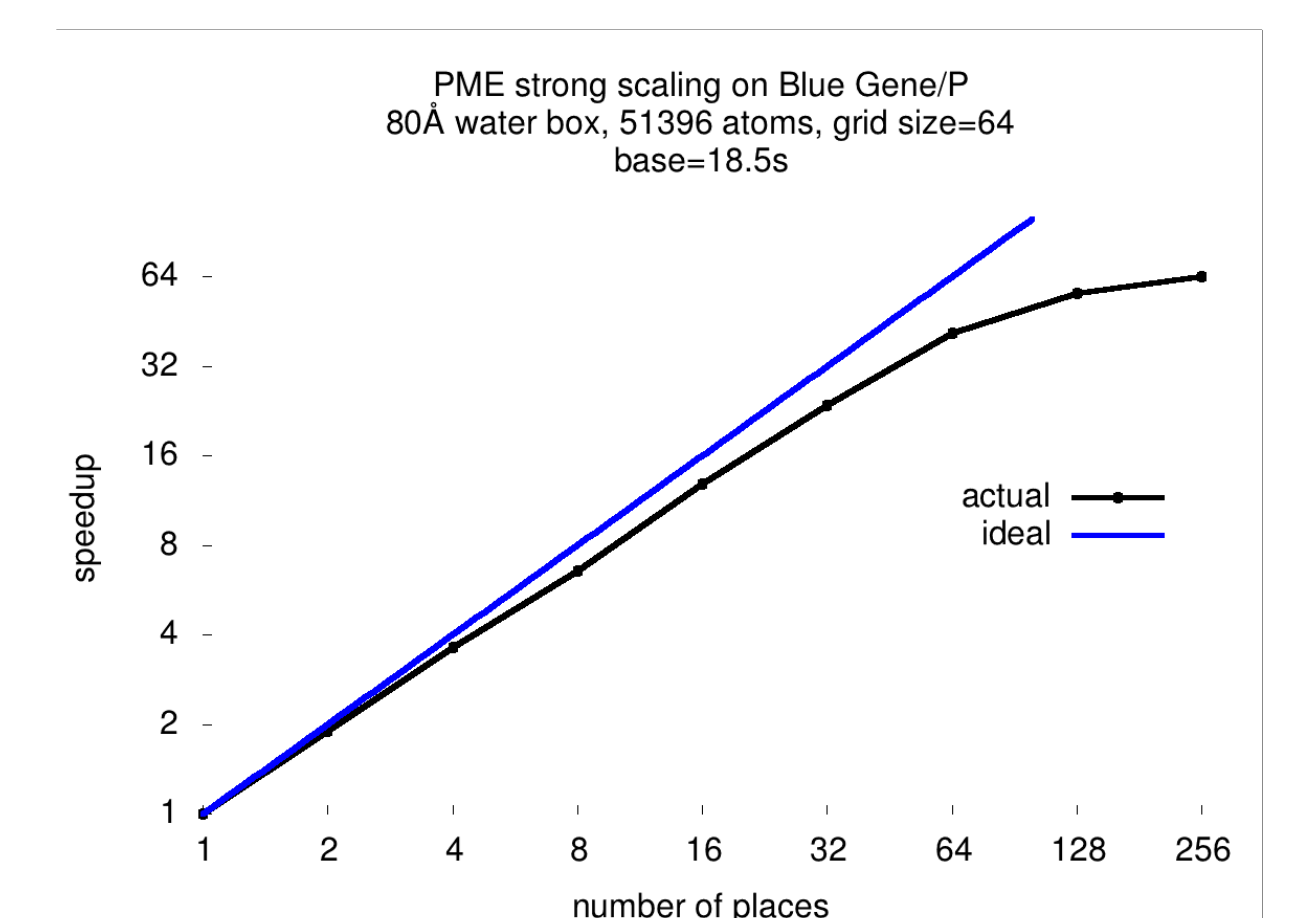
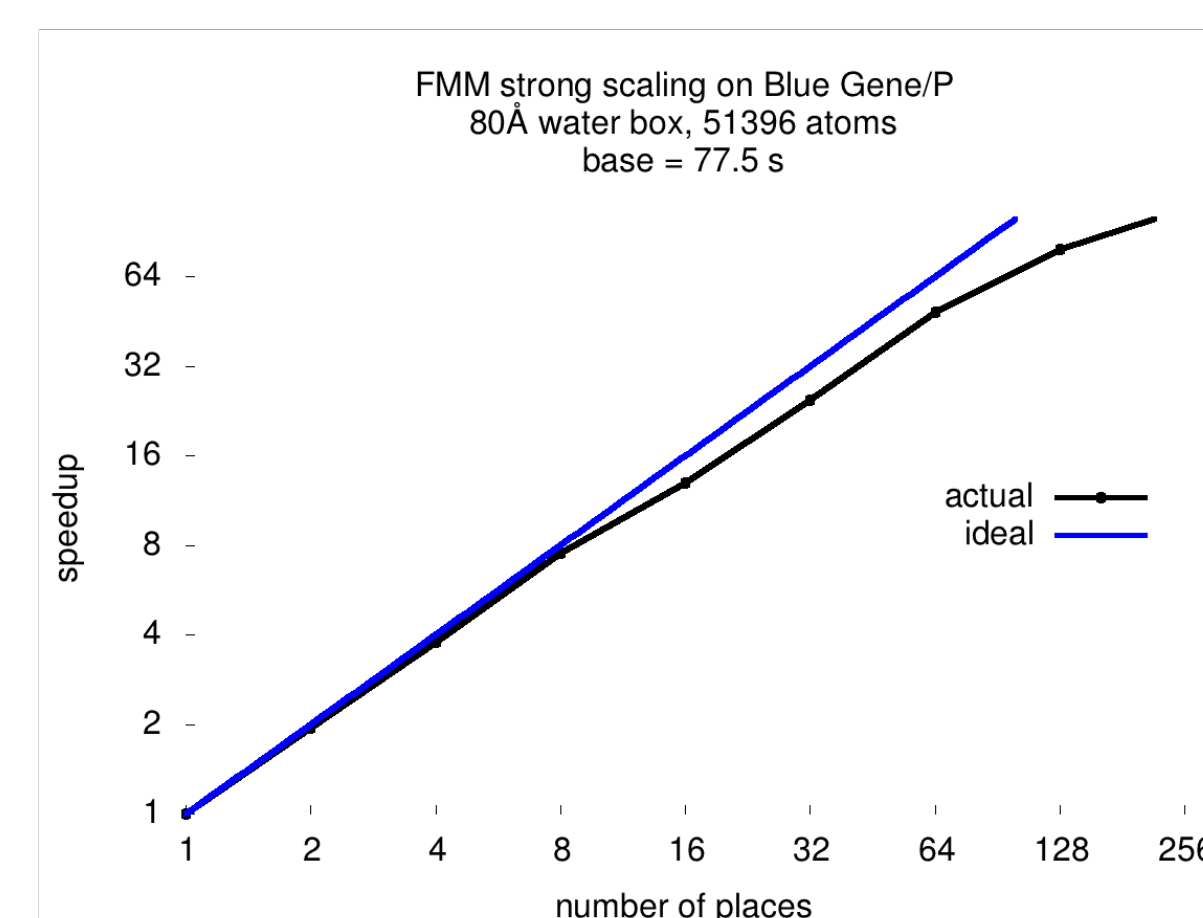


## Distributed octree

Used in **Fast Multipole Method**, finite-element methods. Requires post-order, pre-order, leaf-only traversals.



Efficient parallel algorithms require **bottom-up construction**, **2:1 size balancing** between neighbouring boxes for rigorous error bounds, and **locality preserving distributions** (e.g. Morton- or Z-indexing) for load balancing.



J. Milthorpe, V. Ganesh, A.P. Rendell and D. Grove, "X10 as a parallel language for scientific computation: practice and experience", proc. IPDPS 2011