X10

Jonathan Lee
Daniel Lee

What is X10?

- Programming language designed for high-performance, high-productivity computing on high-end computers
- Development at IBM Research
- Object oriented (OO) Language
- Intended to have simple and clear semantics

Key Design Decisions

- Introduce a new programming language
- Use the Java programming language as a starting point
  - Added a few new things, took away some old things
- Uses partitioned global address space (PGAS) model

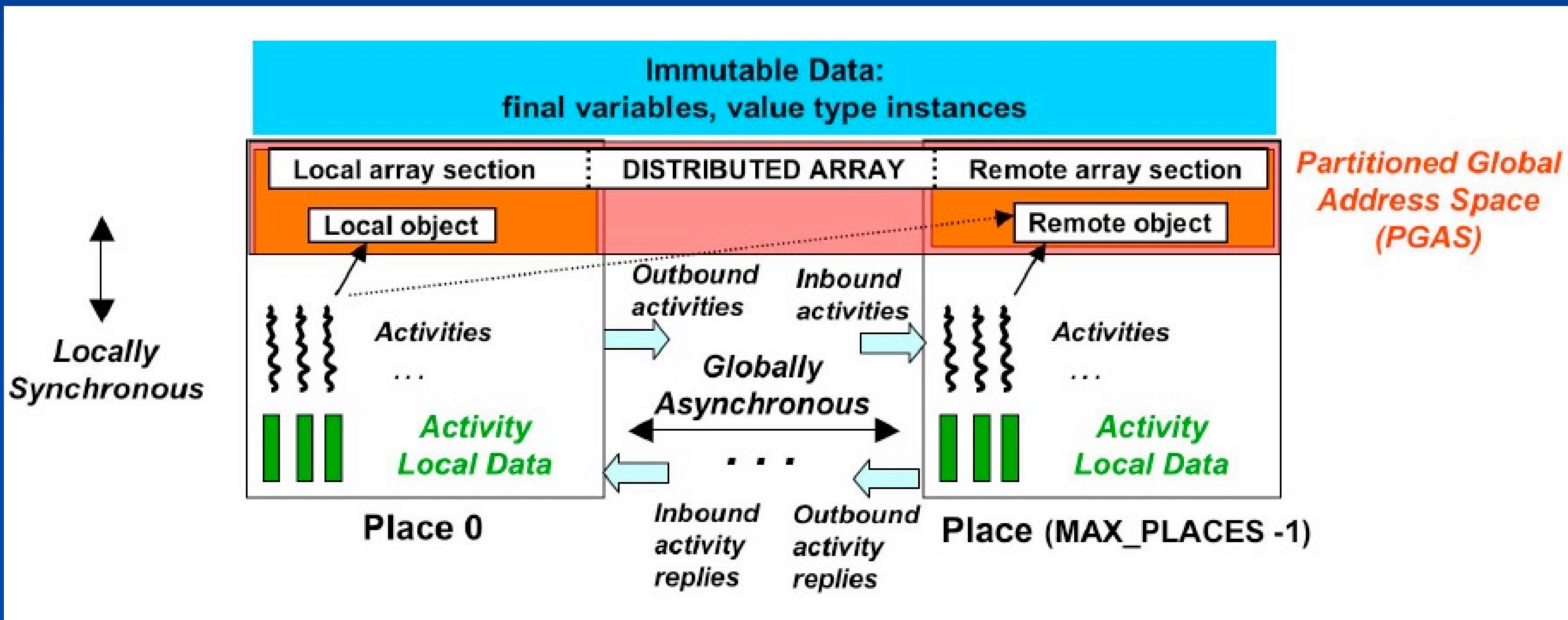
Programming Model: Places

- Collection of data objects and activities (think of as threads) that operate on the data
- Can think of as a "virtual shared-memory multi-processor"
- Every X10 activity runs in a place
- Can get reference to the current place with the constant here
- Places are ordered and the methods next() and prev() can be used to cycle through them

Programming Model: PGAS

- X10 uses PGAS (Partitioned Global Address Space)
- Each place has "partition" of address space
- Scalar objects are allocated completely at a single place
- Elements of an array may be distributed across multiple places

X10 Activities, Places, PGAS Diagram



X10 activities, places, and PGAS

Programming Construct: async

- Can create asynchronous activities using *async* statement
- async (P) S
  - Spawns an activity at the place designated by P to execute S
- Creates parallelism!
- Can be thought of as extremely lightweight threads

Async Example

```
System.out.println(1);
async (place.next()) {
    System.out.println(2);
}
System.out.println(3);
```

Data Structures: Region

- Regions: Just a collection of points
  - Simple contiguous ranges: [0:N]
  - Multidimensional blocks: [0:N,0:M]
  - Can create arbitrary regions of any dimension

Data Structures: Region

- Region Operations:
  - Union: R1 || R2
  - Intersection: R1 && R2
  - Set Difference: R1 - R2

Data Structures: Distributions

- Distributions: Maps each point in a region to a specific place
  - Built in Distributions:
    - Constant: all points map to a single place
    - Block: contiguous sets of points equally divided among places
    - Cyclic: Every Nth point assigned to a place

Data Structures: Distributions

- Distribution Operations:
  - Also include:
    - Range Restriction: D | R
    - Place Restriction: D | P
    - Indexing for places: D[p]
- Example: Block Star Distribution

```
Distribution d = dist.factory.block([0,N],places);
Distribution blockstar = [0:-1,0:-1]->here;
for (point p : d) {
    blockstar = blockstar || [0:M]->d[i];
}
```

Data Structures: Arrays

- X10 Arrays:
  - Takes a distribution as a parameter to assign data to places
  - Example: `double[.] data = new double[[0:N]->here];`
  - Built in and user defined functions support
    - Scans
    - Overlays
    - Reductions
    - Lifting
    - Initialization

Programming Construct: for

- for (point p : R) S
  - Pointwise for for sequential iteration by a single activity
  - Equivalent to Java foreach loops
  - Example:

```
Region r = [0:N];
int[.] x = new int[r->here];
for (point p(i) : r) {
   x[p] = i * 2;
}
```

Programming Construct: foreach

- foreach (point p : R) S
  - For parallel iteration in a single place
  - ≡ for (point p : R) async (here) { S }
  - Example:

```
Region r = [0:N];
int[.] x = new int[r->here];
foreach (point p(i) : r) {
   x[p] = i * 2;
}
```

Programming Construct: ateach

- ateach (point p : D) S
  - For parallel iteration across multiple places
  - ≡ for (point p : D) async (D[p]) { S }
  - Example:

```
Distribution d = [0:4]->place(0) ||
                 [5:9]->place(1);
int[.] x = new int[d];
ateach (point p(i) : r) {
   x[p] = i * 2;
}
```

Programming Construct: future

- f = future(P) E
  - Spawns an activity at place P to execute expression E
  - When parent activity wants the result of E, it executes a f.force()
    - Parent activity blocks until the future activity completes
  - Example:

```
Distribution d = [0:4]->place(0) ||
                 [5:9]->place(1);
int[.] x = new int[d] (point (i)) { return i; };
Future<int> fx5 = future (place(1)) { x[5] };
…
int x5 = fx5.force();
```

Synchronization: Clocks

- X10's synchronization mechanism
- Acts much like a barrier
- Activities register with a clock
- An activity can perform a *next* operation to indicate that it is ready to advance all the clocks it is registered with
- When all activities registered with clock perform next command, activities on clock can continue

Synchronization: finish

- finish S
  - Essentially a join
  - Must block until all child activities recursively complete
  - Also acts as aggregation point for exceptions

- Example:

```
System.out.println("start");
finish foreach(point (i,j) : [0:N,0:M]) {
   System.out.println(N * i + j);
}
System.out.println("end");
```

Synchronization: atomic

- atomic S
  - Such a statement is executed by the activity as if in a single step during which all other activities are frozen
  - Type system ensures that statement S will dynamically access only local data
- Conditional atomic blocks
  - when(e) { s }
  - await(e)

Current Implementation

- Uses polyglot to generate Java code
  - Leverages java threads to achieve concurrence, but not much place partitioning
  - Runtime big and fat; lots of checks and indirection
  - Compiler is fairly simplistic

Advantages of X10

- Java syntax and libraries easy to transition for programmers
- Constructs realatively easy to learn and use
- Easy to use some constructs to gain some parallelism

Limitations of X10

- Hard to load balance places
- Implementation is slow and compiler is simplistic
- Since implementation uses inner classes, final modifiers need to be added in some places
- At current state, using parallelism constructs aggressively is slower

Demo

- Crypto
- Jacobi

The End

- Questions?