Karlsruher Institut für Technologie

X10 Language

A PGAS Parallel Language

Mihail Dzhurev

IPD Snelting, Lehrstuhl Programmierparadigmen

Übersicht

Einführung

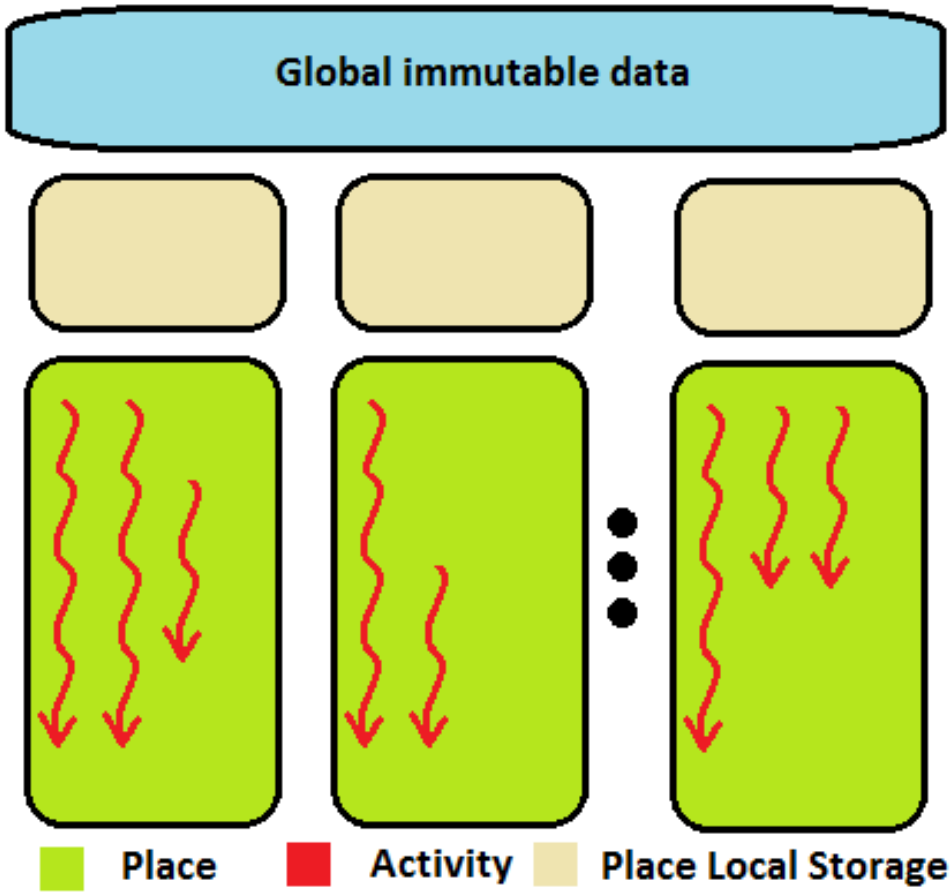
- Entwickelt von IBM seit 2004
- Ziel: Steigerung der Produktivität
- High-Level Objekt-Orientiert mit Closures
- Benutzt Partitioned Global Space Model
- Java oder C++/MPI backend

Hello World

```
class HelloWorld {
    public static def main(argv:Rail[String]) {
        x10.io.Console.OUT.println("Hello, World");
    }
}
```

- Syntax ähnlich zu Java (for, if, class, interface ...) und Scala
- Alles ist ein Objekt
- Es gibt auch Garbage Collection!

Programmiermodel



- Place : Rechner Knoten mit Lokalen Speicher
- Activity : Lightweight Thread

Places

- „A place for everything, everything in it's place“
- Local Storage
- Local Activities
- here
- Anzahl Places konstant während der Laufzeit

Activities

- Lightweight Threads, die leicht gestartet werden können
- bleibt in demelben Place
- kann andere Activities starten
- ... auch in ein anderen Place (mittels at)
- bietet verschiedene Synchronisationsmechanismen
- asynchronous expression (mittels future)

Async

```
import x10.io.Console;
public class ConcHello1 {
    public static def main(argv:Rail[String]!) {
        async Console.OUT.println("Hello, World");
        async Console.OUT.println("Hallo, Welt");
        async Console.OUT.println("Bonjour, Monde");
    }
}
```

- Neue Activities für jede async Statement
- Problem: kann das Output vermischen
- Lösung: benutze atomic!

Atomic

- Problem: atomic ist an blockierende Statements erlaubt
- Lösung: Puffer benutzen

```
public class ConcHello2 {
    public static def main(argv:Rail[String]!) {
        val buffer = Rail.make[String](1);
        buffer(0) = "";
        finish{
        async atomic buffer(0) = buffer(0) + "Hello, World\n";
        async atomic buffer(0) = buffer(0) + "Hallo, Welt\n";
        async atomic buffer(0) = buffer(0) + "Bonjour, Monde\n";
        }
        x10.io.Console.OUT.println(buffer(0));
    }
}
```

- Problem: manchmal wird nichts ausgegeben
- Lösung: finish benutzen

Functional Programming

- Variable/Value deklarieren

```
var i: Int = 5;
val j: Int = 6;
```

- Eine Funktion ist auch ein Value

```
val square = (i:Int) => i*i;
val of4 = (f: (Int)=>Int) => f(4);
val fourSquared = of4(square);
x10.io.Console.OUT.println("4 squared is " + fourSquared);
```

- Lambda Functions

```
val fourCubed = of4( (i:Int)=>i*i*i );
x10.io.Console.OUT.println("4 squared is " + fourCubed);
```

Guards

- Compile-time check
- Beispiel: Skalarprodukt

```
static def dot(v: ValRail[Double], w: ValRail[Double]
    {v.length == w.length}) = {
    var s : Double = 0.0;
    for((i) in 0 .. v.length - 1) {
        s += v(i) * w(i);
    }
    return s;
}
```

- Oder: definiere typ

```
static type Vector3 = ValRail[Double]{length == 3};
```

Guards (2)

- Beispiel: Kreuzprodukt

```
static def cross(v: ValRail[Double], w: ValRail[Double])
    {v.length == 3 && w.length == 3}
    :ValRail[Double]{length == 3}
    = [
        v(1)*w(2) - v(2)*w(1), // component 0
        v(2)*w(0) - v(0)*w(2), // component 1
        v(0)*w(1) - v(1)*w(0) // component 2
    ];
```

- Endlich, was bedeutet '!'

```
T!  := T{home == here}
T!p := T{home ==p}
```

Points

- N-Dimensionaler Tupel von Ints
- Anwendung: Array-Zugriff

```
val p1 : Point{rank==1} = [1];
val p2 : Point{rank==2} = [1,2];
val p8 : Point{rank==8} = [1,2,3,4,5,6,7,8];

val (i,j) = p2;
x10.io.Console.OUT.println("i=" + i + ", j=" +j);
```

Regions

- Menge von Points
- Gut für Schleifen

```
for(var (i) in 1..100)
    doSomething();
```

- kann auch Mehrdimensional sein

```
val R2 = [1..100, 1..200];
val R3 : Region{rank==3} = [0..1, 1..2, 2..4];
```

Regions (2)

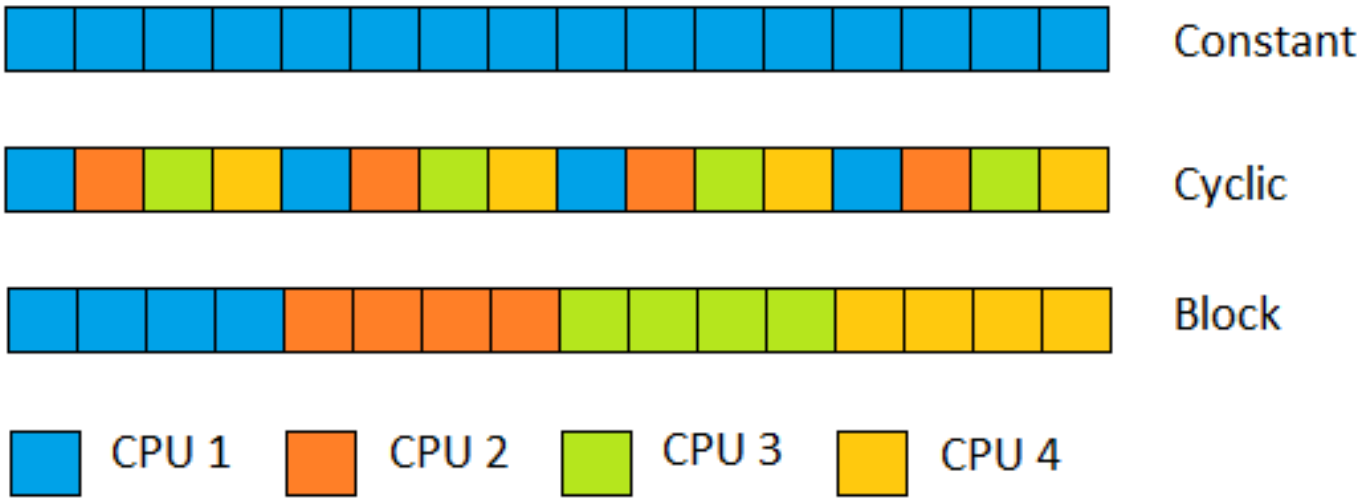
- können in beliebiger Form sein
  - rectangle
  - square
  - ganze Zahlen in Kreis C(O, 42)
- Menge Operationen
  Union, Intersection, Difference

```
val U1a : Region{rank==1} = [1..2];
val U1b : Region{rank==1} = [100..200];
val U1  = U1a || U1b;
val U2a : Region{rank==2} = [1..2, 3..4];
val U2b : Region{rank==2} =  [100..200, 300..400];
val U2  = U2a || U2b;
```

Distributions

- Abbildung von Point zu Place

```
val a : Dist = Dist.makeConst(R);
val b : Dist = Dist.makeCyclic(R);
val c : Dist = Dist.makeBlock(R);
```



Arrays

- Benutzt Point, Region und Dist
- Verteilte Datenstruktur
- kann durch eine Funktion initialisiert sein

```
public static def main(argv:Rail[String]!) {
    val R : Region = 1..35;
    val D : Dist = Dist.makeBlock(R);
    val f : (Point)=> Int = ((i):Point) => i*i;
    val a : Array[Int] = Array.make[Int](D, f);
}
```

Arrays (2)

- Reduce und scan

```
R : Region= 1..8;
val D : Dist = R -> here;
val f : (Point)=> Int = ((i):Point) => 10*i;
val a : Array[Int] = Array.make[Int](D, f);
/*a.reduce(f, unit);*/
a.reduce((i: Int, j: Int)=> i*j , 1);
```

Arrays (3)

- Beispiel: Array ausgeben

```
public static def str[T](a:Array[T]):String =
{
    var s : String = "";
    var first : Boolean = true;
    for(point in a) {
        if (first) first=false;
        else s += ",";
        // works 'cause toString is global
        s += a(point).toString();
    }
    return s;
}
```

Arrays Summe

```
public static def add(a:Array[Int], b:Array[Int])
{a.dist == b.dist} : Array[Int]{self.dist == a.dist} = {
    c : Array[Int]{dist == a.dist}
         = Array.make[Int](a.dist, (p:Point)=>0);
    for(val p in a.dist) {
        at(a.dist(p)) {
            c(p) = a(p) + b(p);
        }
    }
return c;
}
```

- Nachteil: Neue Activity für jede Iteration

Arrays Summe (2)

```
public static def add(a:Array[Int], b:Array[Int])
{a.dist == b.dist} :Array[Int]{self.dist == a.dist} = {
    c : Array[Int]{dist == a.dist}
         = Array.make[Int](a.dist, (p:Point)=>0);
    val D = a.dist;
    val places : ValRail[Place] = D.places();
    for(place in places) {
        at(place) {
            val pointsAtP : Region{rank == D.rank} = D.get(place);
            for(pt in pointsAtP)
               c(pt) = a(pt) + b(pt);
        }// at(p)
    }//for
    return c;
}
```

- Besser: Nur eine neue Activity per Place

Clocks

- Verallgemeinerung von Barrieren
- bestimmte Phasen

```
val c1: Clock = Clock.make();
...
async clocked (c1,c2,c3) {
...
//Bereit fuer naechste Phase
c1.resume();
...
//Bereit fuer naechste Phase, warte fuer andere Activities
c1.next();
...
c2.drop()

}
```

- Statish auf Deadlocks geprüft

Fazit

X10 Language

Leichtes Syntax
Flexibel
Plugin für Eclipse
Noch nicht sehr verbreitet

Fragen?

Literatur/Links

Literatur

- Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar. X10: an Experimental Language for High Productivity Programming of Scalable Systems. P-PHEC workshop, HPCA 2005.
- Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar. X10: an Experimental Language for High Productivity Programming of Scalable Systems. P-PHEC workshop, HPCA 2005.
- Report on the Programming Language X10. Vijay Saraswat, Bard Bloom

Links

http://x10-lang.org/