The X10 Project: Towards Productive Programming of Concurrent and Distributed Systems

David Grove
IBM Research
Agenda

- Why X10?
- X10 in a Nutshell
  - APGAS Programming Model
  - X10 by Example
  - X10 Tool Chain
  - Core Sequential Language
  - Concurrency
  - Distribution
- Experience with Dependent Types
- Evolving from a Research Prototype to a Useful System
- Research Topics
- Conclusion and Discussion
What is X10?

- X10 is a new language being developed by IBM Research as part of the DARPA program on High Productivity Computing Systems (HPCS)

- X10 is an instance of the APGAS programming model in the Java family of languages

- X10 is an open-source project (http://x10-lang.org)
X10 Project Genesis

▪ Project started in 2004
▪ Motivation
  – DARPA HPCS program: Challenge how to productively program a peta-scale machine
    • Blue Waters (IBM Power7-based cluster; NCSA; 2011)
    • 300,000 compute cores
    • Peak 10+ petaflops; sustained >1 petaflop
    • Massive memory bandwidth (fully connected two-tier network)
  – How do you program such a machine?
    • Migrate existing users beyond MPI so they can productively use machine at scale
    • Make it possible for a new class of users (non-MPI) to program such machines
  – the “P” in HPCS is Productive!

▪ This kind of system architecture (in miniature) has become widely available and is being used outside of classic HPC.
▪ Also seeing the rise of heterogeneous systems (GPUs, custom accelerators) and diversity of system architectures
What is Partitioned Global Address Space (PGAS)?

Computation is performed in multiple places.
A place contains data that can be operated on remotely.
Data lives in the place it was created, for its lifetime.

A datum in one place may reference a datum in another place.
Data-structures (e.g. arrays) may be distributed across many places.
Places may have different computational properties (e.g. PPE, SPE, GPU, ...).

A place expresses locality.
Core X10 Programming Model

**Locality Rule:**
Any access to a mutable datum must be performed by a local activity ➔ remote data accesses can be performed by creating remote activities.

- **PGAS:** Replicated Data
- Local Heap
- Remote Heap

- **Locally Synchronous:** Guaranteed coherence for local heap ➔ sequential consistency
- **Globally Asynchronous:** No ordering of inter-place activities ➔ use explicit synchronization for coherence

**Place** = collection of resident activities and objects

**Activity** = sequential computation that runs at a place

**Storage classes:**
- Activity-local
- Place-local
- Partitioned global
- Immutable

**Ordering Constraints (Memory Model)**
X10 Constructs

Fine grained concurrency
- async

Atomicity
- atomic
- when (c)

Place-shifting operations
- at (P)

Ordering
- finish
- clock

Global data-structures
- points, regions, distributions, arrays

Two basic ideas: Places and Asynchrony
Parallel Hello World

import x10.io.Console;

class HelloWorldPar {
    public static def main(args:Rail[String]):void {
        finish ateach (p in Dist.makeUnique()) {
            Console.OUT.println("Hello World from Place" +p);
        }
    }
}

(1) x10c++ -o HelloWorldPar -O HelloWorldPar.x10

(2) runx10 -n 4 HelloWorldPar
Hello World from Place(0)
Hello World from Place(2)
Hello World from Place(3)
Hello World from Place(1)

(3)
Integration via Gaussian Quadrature (recursive divide and conquer)

class Integrate {
    const epsilon = 1.0e-12;
    val fun:(double)=>double;
    static final class resHolder { var value:double; }

    def computeArea(left:double, right:double) {
        return recEval(left, fun(left), right, fun(right), 0);
    }

    def recEval(l:double, fl:double, r:double, fr:double, a:double) {
        val h = (r - l) / 2;       val hh = h / 2;
        val c = l + h;             val fc = fun(c);
        val al = (fl + fc) * hh;   val ar = (fr + fc) * hh;
        val alr = al + ar;
        if (Math.abs(alr - a) < epsilon) return alr;
        val resHolder = new resHolder();
        var expr2:double = 0;
        finish {
            async { resHolder.value = recEval(c, fc, r, fr, ar); }
            expr2 = recEval(l, fl, c, fc, al);
        }
        return resHolder.value + expr2;
    }
}
Distributed 2-D Stencil Calculation

\[
A: \sum \left( \begin{array}{c}
1.0 \\
\end{array} \right) \div 4
\]

repeat until max change < \( \epsilon \)
Distributed 2-D Stencil Calculation

class HeatTransfer_v2 {
    const BigD = Dist.makeBlock([0..n+1, 0..n+1], 0);
    const D = BigD | ([1..n, 1..n] as Region);
    const LR = [0..0, 1..n] as Region;
    const A = DistArray.make[double](BigD, (p:Point)=>{ LR.contains(p) ? 1 : 0 });
    const Temp = DistArray.make[double](BigD);

    static def stencil_1((x,y):Point(2)) {
        return ((at(A.dist(x-1,y)) A(x-1,y)) +
                (at(A.dist(x+1,y)) A(x+1,y)) +
                A(x,y-1) + A(x,y+1)) / 4;
    }
    def run() {
        val D_Base = Dist.makeUnique(D.places());
        var delta:double = 1.0;
        do {
            finish ateach (z in D_Base)
            for (p:Point(2) in D | here)
                Temp(p) = stencil_1(p);

            delta = A.lift(Temp, D.region, (x:double,y:double)=>Math.abs(x-y))
                   .reduce(Math.max.(Double,Double), 0);

            finish ateach (p in D) A(p) = Temp(p);
        } while (delta > epsilon);
    }
}
X10 Status

- X10 is an open source project ([http://x10-lang.org](http://x10-lang.org))
  - Releases, documentation, mailing lists, etc. etc.
  - Version 2.0.3 released mid-April (on X10 Day)

- X10 Community
  - X10 Innovation Grants (18 awards in 2010)
  - X10 Day in mid-April (material available at web site)
  - Handful of non-IBM written X10 applications in the 5-10kloc range

- X10 Implementations
  - JVM-based
    - any platform with Java 5
    - all places run in single JVM process (ie, single node only)
  - C++-based
    - Multi-process (one place per process; multi-node)
    - Linux, AIX, MacOS, Cygwin, BlueGene
    - x86, x86_64, PowerPC,
    - x10rt: IBM PGAS runtime (binary only) or MPI-based (open source)
X10DT: Eclipse-based IDE for X10

A work in progress: version 2.1 (July 2010) first “truly usable” release
Overview of X10's Sequential Features

Many features inherited from Java more or less unchanged
   Classes (single inheritance) with multiple inheritance of interfaces
   static/instance fields/methods; initializers, constructors, etc.

Structs (user-defined primitives)
   headerless, “inlined” in containing object/array/variable
   no subclassing, but can implement interfaces
   currently immutable (all instance fields final)

Closures (functions)

Type system
   Generic types different than Java (instantiation instead of erasure)
   Dependent types
   Function types
   Type inference
Async

async S

- Creates a new child activity that executes statement $S$
- Returns immediately
- $S$ may reference final variables in enclosing blocks
- Activities cannot be named
- Activity cannot be aborted or cancelled

$Stmt ::= \text{async}(p,l) \ Stmt$

cf Cilk’s spawn

```java
// Compute the Fibonacci sequence in parallel.
def run() {
    if (r < 2) return;
    val f1 = new Fib(r-1),
            f2 = new Fib(r-2);
    finish {
        async f1.run();
        f2.run();
    }
    r = f1.r + f2.r;
}
```
Finish

L: finish S

- Execute S, but wait until all (transitively) spawned asyncs have terminated.

Rooted exception model

- Trap all exceptions thrown by spawned activities.
- Throw an (aggregate) exception if any spawned async terminates abruptly.
- implicit finish at main activity

finish is useful for expressing “synchronous” operations on (local or) remote data.
Atomic

atomic S

◆ Execute statement S atomically

◆ Atomic blocks are conceptually executed in a single step while other activities are suspended: isolation and atomicity.

◆ An atomic block body (S) ...
  • must be nonblocking
  • must not create concurrent activities (sequential)
  • must not access remote data (local)

Stmt ::= atomic Statement
MethodModifier ::= atomic

// target defined in lexically enclosing scope.
atomic def CAS(old:Object, n:Object) {
  if (target.equals(old)) {
    target = n;
    return true;
  }
  return false;
}

// push data onto concurrent // list-stack
val node = new Node(data);
atomic {
  node.next = head;
  head = node;
}
When

when (E) S

◆ Activity suspends until a state in which the guard \( E \) is true.

◆ In that state, \( S \) is executed atomically and in isolation.

◆ Guard \( E \) is a boolean expression

  • must be nonblocking
  • must not create concurrent activities (sequential)
  • must not access remote data (local)
  • must not have side-effects (const)

await (E)

◆ syntactic sugar for when (E) ;
At

\( \text{at}(p) \ S \)

- Execute statement \( S \) at place \( p \)
- Current activity is blocked until \( S \) completes

async \( (p) \ S \)

syntactic sugar for \( \text{async at}(p) \ S; \)

\[
\begin{align*}
\text{Stmt} & \ ::= \ \text{at}(p) \ \text{Stmt} \\
\end{align*}
\]

// Copy field f from a to b
def copyRemoteFields(a, b) {
    at (b) b.f =
    at (a) a.f;
}

// Increment field f of obj
def incField(obj, inc) {
    at (obj) obj.f += inc;
}

// Invoke method m on obj
def invoke(obj, arg) {
    at (obj) obj.m(arg);
}
X10 Distributed Object Model

Classes (object instances)
- Each object has a home (the place where it was created)
- Objects have global identity (==)
- Remote reference created via lexical capture in at(p) S or async(p) S
- All mutable instance state lives in the object's home location
- Val (final) instance state may optionally be declared global
- Global state serialized along with object; available any place the remote reference is
- Instances' methods may be declared global (can only access global state) and thus callable in any place

Structs
- All state is immutable
- Freely replicated between places (no global identity)
- Home of a struct is always “here”

Closures
- Like structs (slight technical differences for == semantics)
Agenda

- Why X10?
- X10 in a Nutshell
  - APGAS Programming Model
  - X10 by Example
  - X10 Tool Chain
  - Core Sequential Language
  - Concurrency
  - Distribution
- Experience with Dependent Types
- Evolving from a Research Prototype to a Useful System
- Research Topics
- Conclusion and Discussion
Adventures in Dependent Types

X10's static type system includes dependent types

\[ T\{e\} \]

\( e \) is a conjunction of equalities and disequalities over immutable state.

\[ C\{c\} \text{ is a subtype of } D\{d\} \text{ if } C \text{ is a subtype of } D \text{ and } c \text{ entails } d \]
Uses of Dependent Types in X10 2.0

Static checking of place/locality constraints (before 2.0, checked dynamically)

Constraints over the home of an object
  - field has same location as containing object
  - two method parameters have the same home
  - a method parameter is “here”

Used heavily in array/region/distribution class library (x10.array)
  - Specify rank of an array
  - Specify that a region is dense and rectangular (enables for loop optimization)
  - Specify that multiple arrays have the same region/distribution

Can be used to require variables/fields to be non-null

Can be used to encode ownership relationships

Lots of other uses.....

As language designers/implementors we thought this was great
User Reaction

X10 2.0.1 was first release which really enforced static checking of place types

Users were unhappy. Very unhappy.

“My code used to work; now it doesn't compile, and I don't understand the error message”

And these were PhD's in the Programming Technology Dept at IBM Research

Post Mortem

- Error messages could have been better
- Errors were only “real” in multi-place programs, irrelevant for single place code (some programs only need async/finish not at)
- Unforeseen interactions with type inference
  - Programmers like writing types as documentation, but are unwilling to write the most precise constrained type explicitly
  - Introduced new syntax: `val x <: T = ...`
Blending static and dynamic typing

After some thinking, we introduced the following in X10 2.0.3

If the compiler can't statically prove or disprove a constraint that is required to enable a method to be invoked, it inserts code to dynamically check that the constraint is true.

In effect, the compiler injects a cast for the user from $T\{c\}$ to $T\{c'\}$ if it can't prove $c$ entails $c'$

Will not automatically inject a cast from $T$ to $S$

Command line options to disable cast injection (for hardening production code)

Jury still out on whether this is a good idea, but initial experience is positive.
What does it take for a language to be useful?
- Applicability: provides an advantage; makes a task easier in at least some domains
- Predictability: things work the way the programmer expects them to
- Robustness: implementation bugs must be extremely rare
- Self-service: can productively use X10 without knowing my phone number...
- Tool chain: IDE, debugger, profiling tools, etc.
- Rate of change: not too fast, but not too slow either
- ....

Standards are high; especially in the commercial space which is used to Java

Odds of success for a programming language to be useful?
Making X10 useful

IBM Research is making an attempt to mature X10 into a useful system by early 2011
See potential for impact beyond classic HPC community
Ramped up team: from 13 to 35 people in 2010

What are the major challenges?
Transition from primarily research to primarily development focus (IBM helps)
X10DT (IDE)
Building a good IDE (even leveraging Eclipse) takes a lot of work
Major implications for compiler too:
• Need “good as possible” ASTs for incorrect programs
• Need them quickly, incrementally, etc.
Robustness and predictability not hallmarks of research systems...
Test, build, package are specialized and critical skills
The unknown
Investing about 5 people in “just” writing X10 applications in various domains
Possible Research Topics

Constrained types to the limit
- How far can you go?
- User-defined predicates … support user-defined distributions?
- Type inference – place types.
- Exploit for optimization

Support for sparse computations
- Language support, compiler and runtime support

Determinacy checking
- Clocked types
- Commutativity analysis

Efficient implementation of (labeled) finish

Tooling for X10/APGAS

Multi-mode: mix Java/C++/Cuda backends in a single execution

Distributed GC for X10

Distributed VM for X10
- Concurrency, structs, functions, generics, distribution, heterogeneity, link-time verification
Conclusions and Discussion

X10/APGAS

A programming language/model for multi-core, clusters, MPPs, accelerators
Many things done; even more to do in the future
The next 12 months should be an exciting year for the project!
Abundance of research problems to be tackled!!
Consider giving it a try sometime this year...

More information: x10-lang.org

Thanks!

Questions?