A Tutorial on X10 and its Implementation

David Grove
IBM TJ Watson Research Center

This material is based upon work supported in part by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002
Central Challenge: Productive Programming of Large Scale Supercomputers
– Clustered systems
  • 1000’s of SMP Nodes connected by high-performance interconnect
  • Large aggregate memory, disk storage, etc.
Historical Performance of Top 500 Supercomputers

Productive programming of large-scale clustered system is rapidly becoming mainstream
X10 Performance and Productivity at Scale

- An evolution of Java for concurrency, scale out, and heterogeneity
  - Language focuses on high productivity and high performance
  - Bring productivity gains from commercial world to HPC developers
  - Bring high performance scale out computing to commercial developers

- The X10 language provides:
  - Java-like language (statically typed, object oriented, garbage-collected)
  - Ability to specify scale-out computations (exploit modern networks, clusters)
  - Ability to specify fine-grained concurrency (exploit multi-core)
  - Single programming model for computation offload and heterogeneity (exploit GPUs)
  - Migration path
    - Make X10 programming model (APGAS) available in other languages
      - APGAS library in Java (complete) C++ (underway) wraps X10 runtime
    - X10 interoperability with Java and C/C++ enables reuse of existing libraries and construction of mixed X10/Java applications
Outline

- X10 concepts and language overview
- X10 Implementation
- What's new in X10 since X10'12?
Partitioned Global Address Space (PGAS) Languages

Managing locality is a key *programming* task in a distributed-memory system

PGAS combines a single global address space with locality awareness
- PGAS languages: Titanium, UPC, CAF, X10, Chapel
- Single address space across all shared-memory nodes
  - any task or object can refer to any object (local or remote)
- Partitioned to reflect locality
  - each partition (X10 place) must fit within a shared-memory node
  - each partition contains a collection of tasks and objects

In X10
- tasks and objects are mapped to places explicitly
- objects are immovable
- tasks must spawn remote task or shift place to access remote objects
X10 Combines PGAS with Asynchrony (APGAS)

Fine-grain concurrency
- `async S`
- `finish S`

Place-shifting operations
- `at(p) S`
- `at(p) e`

Atomicity
- `when(c) S`
- `atomic S`

Distributed heap
- `GlobalRef[T]`
- `PlaceLocalHandle[T]`
Hello Whole World

```java
class HelloWholeWorld {
  public static def main(args:Rail[String]) {
    finish
    for (p in Place.places())
      at (p)
      async
      Console.OUT.println(p + " says " + args(0));
    Console.OUT.println("Goodbye");
  }
}
```

% x10c++ HelloWholeWorld.x10
% X10_NPLACES=4; ./a.out hello
Place 0 says hello
Place 2 says hello
Place 3 says hello
Place 1 says hello
Goodbye
import x10.io.Console;
import x10.util.Random;

class MontyPi {
  public static def main(args:Rail[String]) {
    val N = Int.parse(args(0));
    val r = new Random();
    var result:Double = 0;
    for (1..N) {
      val x = r.nextDouble();
      val y = r.nextDouble();
      if (x*x + y*y <= 1) result++;
    }
    val pi = 4*result/N;
    Console.OUT.println("The value of pi is " + pi);
  }
}
Concurrent Monty Pi

```scala
import x10.io.Console;
import x10.util.Random;

class MontyPi {
  public static def main(args: Rail[String]) {
    val N = Int.parse(args(0));
    val P = Int.parse(args(1));
    val result = new Cell[Double](0);
    finish for (1..P) async {
      val r = new Random();
      var myResult: Double = 0;
      for (1..(N/P)) {
        val x = r.nextDouble();
        val y = r.nextDouble();
        if (x*x + y*y <= 1) myResult++;
      }
      atomic result() += myResult;
    }
    val pi = 4*(result())/N;
    Console.OUT.println("The value of pi is " + pi);
  }
}
```
Distributed Monty Pi

```scala
import x10.io.Console;
import x10.util.Random;

class MontyPi {
  public static def main(args:Rail[String]) {
    val N = Int.parse(args(0));
    val result = GlobalRef[Cell[Double]](new Cell[Double](0));
    finish for (p in Place.places()) at (p) async {
      val r = new Random();
      var myResult:Double = 0;
      for (1..(N/Place.MAX_PLACES)) {
        val x = r.nextDouble();
        val y = r.nextDouble();
        if (x*x + y*y <= 1) myResult++;
      }
      at (result) atomic result()() += myResult;
    }
    val pi = 4*(result()())/N;
    Console.OUT.println("The value of pi is " + pi);
  }
}
```
APGAS Idioms

- **Remote evaluation**
  \[
  v = \text{at}(p) \ \text{evalThere}(\text{arg1}, \text{arg2});
  \]

- **Active message**
  \[
  \text{at}(p) \ \text{async} \ \text{runThere}(\text{arg1}, \text{arg2});
  \]

- **Recursive parallel decomposition**
  \[
  \text{def} \ \text{fib}(n:\text{Int}):\text{Int} \ {\}
  \quad \text{if} \ (n < 2) \ \text{return} \ 1;
  \quad \text{val} \ f1:\text{Int};
  \quad \text{val} \ f2:\text{Int};
  \quad \text{finish} \ {\}
  \quad \quad \text{async} \ f1 = \text{fib}(n-1);
  \quad \quad f2 = \text{fib}(n-2);
  \quad {\}}
  \quad \text{return} \ f1 + f2;
  \]

- **SPMD**
  \[
  \text{finish for} \ (p \ \text{in} \ \text{Place}.\text{places}()) \ {\}
  \quad \text{at}(p) \ \text{async} \ \text{runEverywhere}();
  \]

- **Atomic remote update**
  \[
  \text{at}(\text{ref}) \ \text{async} \ \text{atomic} \ \text{ref}() += v;
  \]

- **Data exchange**
  \[
  // \ \text{swap row i local and j remote}
  \text{val} \ h = \text{here};
  \text{val} \ \text{row}_i = \text{rows}().(i);
  \text{finish} \ \text{at}(p) \ \text{async} \ {\}
  \quad \text{val} \ \text{row}_j = \text{rows}().(j);
  \quad \text{rows}().(j) = \text{row}_i;
  \quad \text{at}(h) \ \text{async} \ \text{row}().(i) = \text{row}_j;
  \]

A handful of key constructs cover a broad spectrum of patterns
Outline

- X10 concepts and language overview
- X10 Implementation
- What's new in X10 since X10'12?
X10 Target Environments

- High-end large HPC clusters
  - BlueGene/P and BlueGene/Q
  - Power775 (aka PERCS machine, P7IH)
  - x86 + InfiniBand, Power + InfiniBand
  - Goal: deliver scalable performance competitive with C+MPI

- Medium-scale commodity systems
  - ~100 nodes (~1000 core and ~1 terabyte main memory)
  - Goal: deliver main-memory performance with simple programming model (accessible to Java programmers)

- Developer laptops
  - Linux, Mac OSX, Windows. Eclipse-based IDE
  - Goal: support developer productivity
X10 Implementation Summary

- X10 Implementations
  - C++ based ("Native X10")
    - Multi-process (one place per process; multi-node)
    - Linux, AIX, MacOS, Cygwin, BlueGene
    - x86, x86_64, PowerPC
  - JVM based ("Managed X10")
    - Multi-process (one place per JVM process; multi-node)
    - Current limitation on Windows to single process (single place)
    - Runs on any Java 6 JVM

- X10DT (X10 IDE) available for Windows, Linux, Mac OS X
  - Based on Eclipse 3.7
  - Supports many core development tasks including remote build/execute facilities
X10 Compilation & Execution

X10 Compiler Front-End

- X10 Source
  - Parsing / Type Check
  - X10 AST
  - AST Optimizations
  - AST Lowering

Managed X10

- Java Back-End
  - Java Code Generation
  - Java Source
  - XRJ
  - Java Compiler
  - Java Bytecode
  - JNI
  - Java VMs

Native X10

- C++ Back-End
  - C++ Code Generation
  - C++ Source
  - Cuda Source
  - Platform Compilers
  - Native executable
  - Native Environment (CPU, GPU, etc)

Java Interop Support

Existing Java Application

Native Environment (CPU, GPU, etc)

Existing Native (C/C++/etc) Application
X10 Runtime

- **X10RT (X10 runtime transport)**
  - active messages, collectives, RDMAs
  - implemented in C; emulation layer

- **Native runtime**
  - processes, threads, atomic operations
  - object model (layout, rtt, serialization)
  - two versions: C++ and Java

- **XRX (X10 runtime in X10)**
  - implements APGAS: async, finish, at
  - X10 code compiled to C++ or Java

- **Core X10 libraries**
  - x10.array, io, util, util.concurrent

```
X10 Core Class Libraries
XRX
Native Runtime
X10RT
PAMI TCP/IP MPI DCMF CUDA

X10 Application
```
XRX: Async Implementation

- Many more logical tasks (asyncs) than execution units (threads)

- Each async is encoded as an X10 Activity object
  - async body encoded as an X10 closure
  - reference to governing finish
  - state: clocks...
  - the activity object (or reference) is not exposed to the programmer

- Per-place scheduler
  - per-place pool of worker threads
  - per-worker deque of pending activities
  - cooperative
    - activity assigned to one thread from start to finish
    - number of threads in pool dynamically adjusted to compensated for blocked activities
  - work stealing
    - worker processes its pending activities first then steals activity from random coworker
XRX: At Implementation

- at (p) async
  - source side: synthesize active message
    - async id + serialized heap + control state (finish, clocks)
    - compiler identifies captured variables (roots)
    - runtime serializes heap reachable from roots
  - destination side: decode active message
    - polling (when idle + on runtime entry)
    - new Activity object pushed to worker’s deque

- at (p)
  - implemented as “async at” + return message
  - parent activity blocks waiting for return message
    - normal or abnormal termination (propagate exceptions and stack traces)

- ateach (broadcast)
  - elementary software routing
XRX: Finish Implementation

- Distributed termination detection is hard
  - arbitrary message reordering

- Base algorithm
  - one row of n counters per place with n places
  - increment on spawn, decrement on termination, message on decrement
  - finish triggered when sum of each column is zero

- Optimized algorithm(s)
  - local aggregation and message batching (up to local quiescence)
  - pattern-based specialization
    - local finish, SPMD finish, ping pong, single async
  - software routing
  - pure runtime optimizations + static analysis + pragmas
  - uncounted asyncs
Outline

- X10 concepts and language overview
- X10 Implementation
- What's new in X10 since X10'12?
X10 Highlights since X10’12

- Releases: 2.2.3, 2.3.0, 2.3.1
  - X10 2.2.3: won HPC Challenge Class II Best Performance at SC’12
  - X10 2.3: major improvements in Java Interoperability (last talk today)

- Focus on scalability: X10 at Peta-scale (Afternoon keynote)

- X10 Application work at IBM
  - M3R: Main Memory Map Reduce [VLDB’12; X10’12]
  - SatX10: Parallel SAT Solving [SAT’12 tools paper; CPAIOR 2013 workshop]
  - Enablement of X10 frameworks in Java-based middleware for scale out computing

- Many internal improvements to Managed X10, Native X10, and XRX/X10RT

- X10DT
  - Redesign of remote launch/build capabilities
  - Available as both “classic” X10DT and new “editor-only” edition

- X10 Community
  - 19 publications in 2012; 10 so far in 2013

Active & growing X10 community!
Summary of X10 PERCS results

- Productivity Study: workflow 2 (compact codes)
  - 3.3x gain using X10 vs C+MPI (for 6 programs, 129 days in C+MPI; 39 days in X10)
  - Adding in estimated gains from X10 Debugger reduces X10 time to 22 days (5.9x)
  - Scaling to full PERCS-size system: 384 days in C+MPI vs. 34 days in X10 (11.3x)

- X10 Scalability/Performance
  - Show scaling/performance for 8 benchmarks on PERCS system (P7IH; 47,000 cores)
  - Summary: all programs scaling well; successfully accomplished goals

UTS Characteristics
  - highly irregular ⇒ need global load balancing
X10 achieves 98% efficiency at 47k cores

Implementation:
  - 493 lines of X10
    (+ 642 lines of sequential C for SHA1 hash)

“X10 is a second generation PGAS language”

APGAS led to design of new algorithm for UTS
Highlights of X10 2.4 (summer 2013)

- Restructuring of core XRX runtime to facilitate C++ APGAS library
  - Library will provide C++ bindings for async/finish/at
  - Enable explicit memory management of core XRX (interop with C++)
  - Funded as part D-TEC project in DoE X-Stack program

- Major enhancements to X10 Arrays
  - Long based indexing (very large arrays, critical for DistArray)
  - Additional high-performance Rail/Array/DistArray implementations for common case of dense, rectangular, zero-based arrays
  - Existing Region-based Array/DistArray still available (updated to long indexing)

- New pure-Java X10RT implementation for Managed X10
  - Increase portability by removing need for JNI library

- Language mostly backwards compatible with X10 2.3, but Array changes are not backwards compatible and will impact virtually all programs to some degree.
  - Decision to break backwards compatibility was not easy, but we felt it was necessary
X10 Community: ANUChem

- Computational chemistry codes written in X10
  - FMM: electrostatic calculation using Fast Multipole method
  - PME: electrostatic calculation using the Smooth Particle Mesh Ewald method
  - Pumja Rasaayani: energy calculation using Hartree-Fock SCF-method

- Developed at ANU by Josh Milthorpe, V. Ganesh, and Taweetham Limpanuparb
  - Open source (EPL); tracks X10 releases
  - Around 16,000 lines of X10 code
  - Overview and initial results in [Milthorpe et al IPDPS 2011]
  - Recent publication in JCTC using ANUChem FMM code

FTICR-MS ion cloud evolution in the first 2.5ms of simulation;
packet of 5K lysine and 5K glutamine.
**Library Design**

- **Aim:** Define solid abstractions for billion scale graph processing

---

**Slide by Miyuru Dayarathna, X10 2012**

**New release:** ScaleGraph 2.0a, made in April 2013
X10 Community: Large-Scale Traffic Flow Simulation

X10-based Ultra-Large Scale Agent Simulation on the 2 Petaflops Supercomputer

Goal: To build a scalable large-scale agent simulation platform based on X10 that runs on a Super Computer with ten thousands of CPU cores and dual links of 40Gbps Infiniband network

Status: Completed the multi-node version and verified the scalable performance with the Hiroshima road network.

New in 2013: runs on BlueGene/Q

Slide by Toyotaro Suzumura, X10 2012
Welcome to X10 2013!