Introduction to X10

Olivier Tardieu

IBM Research

This material is based upon work supported by the Defense Advanced Research Projects Agency, by the Air Force Office of Scientific Research, and by the Department of Energy.

Take Away

- X10 is
  - a programming language
    - derived from and interoperable with Java
  - an open source tool chain
    - compilers, runtime, IDE
    - developed at IBM Research since 2004 with support from DARPA, DoE, and AFOSR
    - >100 contributors (IBM and Academia)
  - a growing community
    - >100 papers
    - workshops, tutorials, courses

- X10 tackles the challenge of programming at *scale*
  - first HPC, then clusters, now cloud
  - scale out: run across many distributed nodes
  - scale up: exploit multi-core and accelerators
  - elasticity and resilience
  - double goal: *productivity* and *performance*

Links

- Main X10 website
  http://x10-lang.org
- X10 Language Specification
  http://x10.sourceforge.net/documentation/languagespec/x10-latest.pdf
- A Brief Introduction to X10 (for the HPC Programmer)
  http://x10.sourceforge.net/documentation/intro/intro-223.pdf
- X10 2.5.3 release (command line tools only)
  http://sourceforge.net/projects/x10/files/x10/2.5.3/
- X10DT 2.5.3 release (Eclipse-based IDE)
  http://sourceforge.net/projects/x10/files/x10dt/2.5.3/

Current IBM X10 Team

Agenda

- X10 overview
  - APGAS programming model
  - X10 programming language
- Tool chain
- Implementation
- Applications
- 2014/2015 Highlights
  - Grid X10

X10 Overview

Asynchronous Partitioned Global Address Space (APGAS)

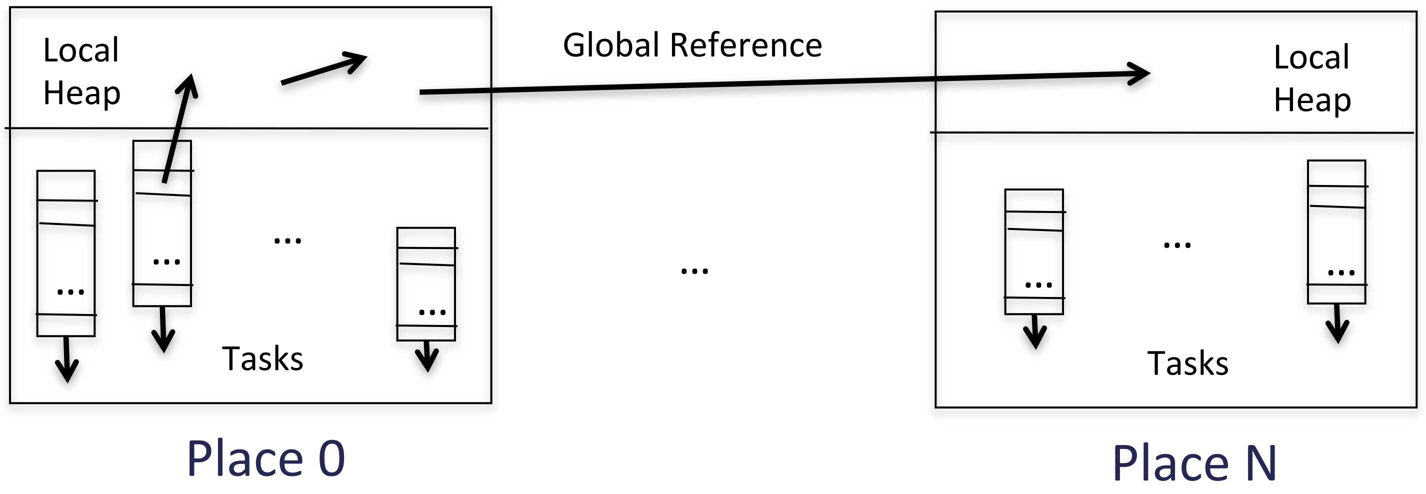
Memory abstraction

- Message passing
  - each task lives in its own address space; example: MPI
- Shared memory
  - shared address space for all the tasks; example: OpenMP
- PGAS
  - global address space: single address space across all tasks
  - partitioned address space: each partition must fit within a shared-memory node
  - examples: UPC, Co-array Fortran, X10, Chapel

Execution model

- SPMD
  - symmetric tasks progressing in lockstep; examples: MPI, OpenMP 3, UPC, CUDA
- APGAS
  - asynchronous tasks; examples: Cilk, X10, OpenMP 4 tasks

Places and Tasks



Task parallelism

- `async S`
- `finish S`

Place-shifting operations

- `at(p) S`
- `at(p) e`

Concurrency control

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

Distributed heap

- `GlobalRef[T]`
- `PlaceLocalHandle[T]`

Idioms

- Remote procedure call

```
v = at(p) evalThere(arg1, arg2);
```

- Active message

```
at(p) async runThere(arg1, arg2);
```

- Divide-and-conquer parallelism

```
def fib(n:Long):Long {
  if(n < 2) return n;
  val f1:Long;
  val f2:Long;
  finish {
    async f1 = fib(n-1);
    f2 = fib(n-2);
  }
  return f1 + f2;
}
```

- SPMD

```
finish for(p in Place.places()) {
  at(p) async runEverywhere();
}
```

- Atomic remote update

```
at(ref) async atomic ref() += v;
```

- Computation/communication overlap

```
val acc = new Accumulator();
while(cond) {
  finish {
    val v = acc.currentValue();
    at(ref) async ref() = v;
    acc.updateValue();
  }
}
```

BlockDistRail.x10

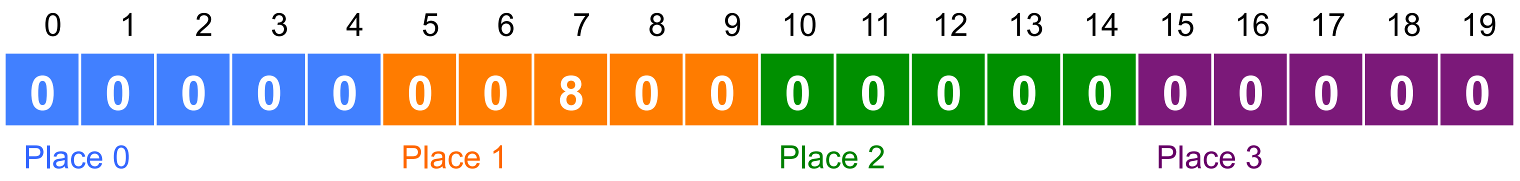
```
public class BlockDistRail[T] {
  protected val sz:Long; // block size
  protected val raw:PlaceLocalHandle[Rail[T]];

  public def this(sz:Long, places:Long){T haszero} {
    this.sz = sz;
    raw = PlaceLocalHandle.make[Rail[T]](PlaceGroup.make(places), ()=>new Rail[T](sz));
  }

  public operator this(i:Long) = (v:T) { at(Place(i/sz)) raw()(i%sz) = v; }

  public operator this(i:Long) = at(Place(i/sz)) raw()(i%sz);

  public static def main(Rail[String]) {
    val rail = new BlockDistRail[Long](5, 4);
    rail(7) = 8; Console.OUT.println(rail(7));
  }
}
```



Like Java

- Objects
  - classes and interfaces
    - single-class inheritance, multiple interfaces
  - fields, methods, constructors
  - virtual dispatch, overriding, overloading, static methods
- Packages and files
- Garbage collector
- Variables and values (final variables, but final is the default)
  - definite assignment (extended to tasks)
- Expressions and statements
  - control statements: if, switch, for, while, do-while, break, continue, return
- Exceptions
  - try-catch-finally, throw
- Comprehension loops and iterators

Beyond Java

- Syntax

types	"`x:Long`" rather than "`Long x`"
declarations	`val`, `var`, `def`
function literals	`(a:Long, b:Long) => a < b ? a : b`
ranges	`0..(size-1)`
operators	user-defined behavior for standard operators

- Types

local type inference	`val b = false;`
function types	`(Long, Long) => Long`
typedefs	`type BinOp[T] = (T, T) => T;`
structs	headerless inline objects; extensible primitive types
arrays	multi-dimensional, distributed; implemented in X10
properties and constraints	extended static checking
reified generics	templates; constrained kinds

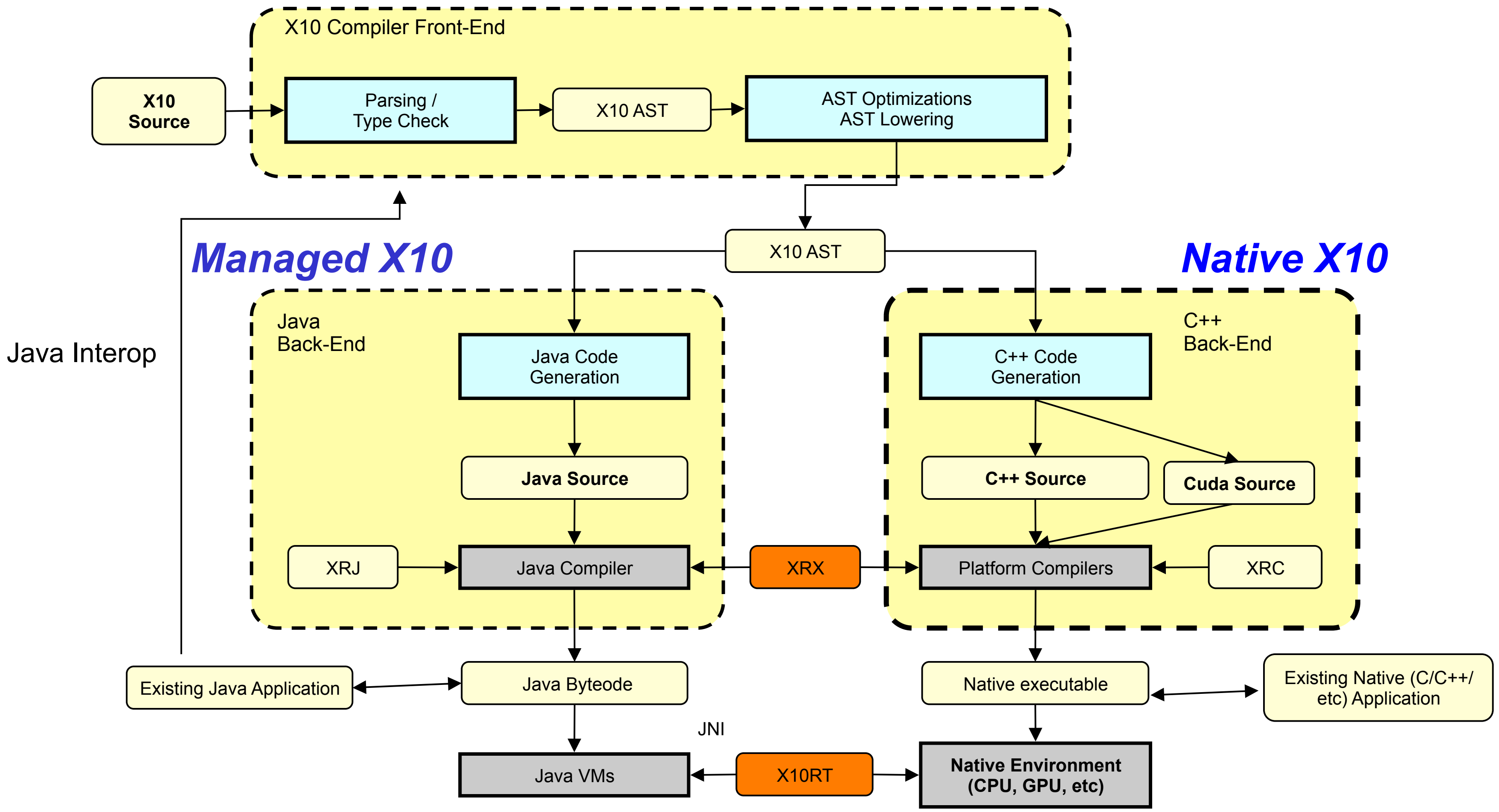
<u>gradual typing</u>

Tool Chain

Tool Chain

- Eclipse Public License
- “Native” X10 implementation
  - C++ based; CUDA support
  - distributed multi-process (one place per process + one place per GPU)
  - C/POSIX network abstraction layer (X10RT)
  - x86, x86_64, Power; Linux, AIX, OS X, Windows/Cygwin, BG/Q; TCP/IP, PAMI, MPI
- “Managed” X10 implementation
  - Java 6/7 based; no CUDA support
  - distributed multi-JVM (one place per JVM)
  - pure Java implementation over TCP/IP or using X10RT via JNI (Linux & OS X)
- X10DT (Eclipse-based IDE) available for Windows, Linux, OS X
  - supports many core development tasks including remote build & execute facilities

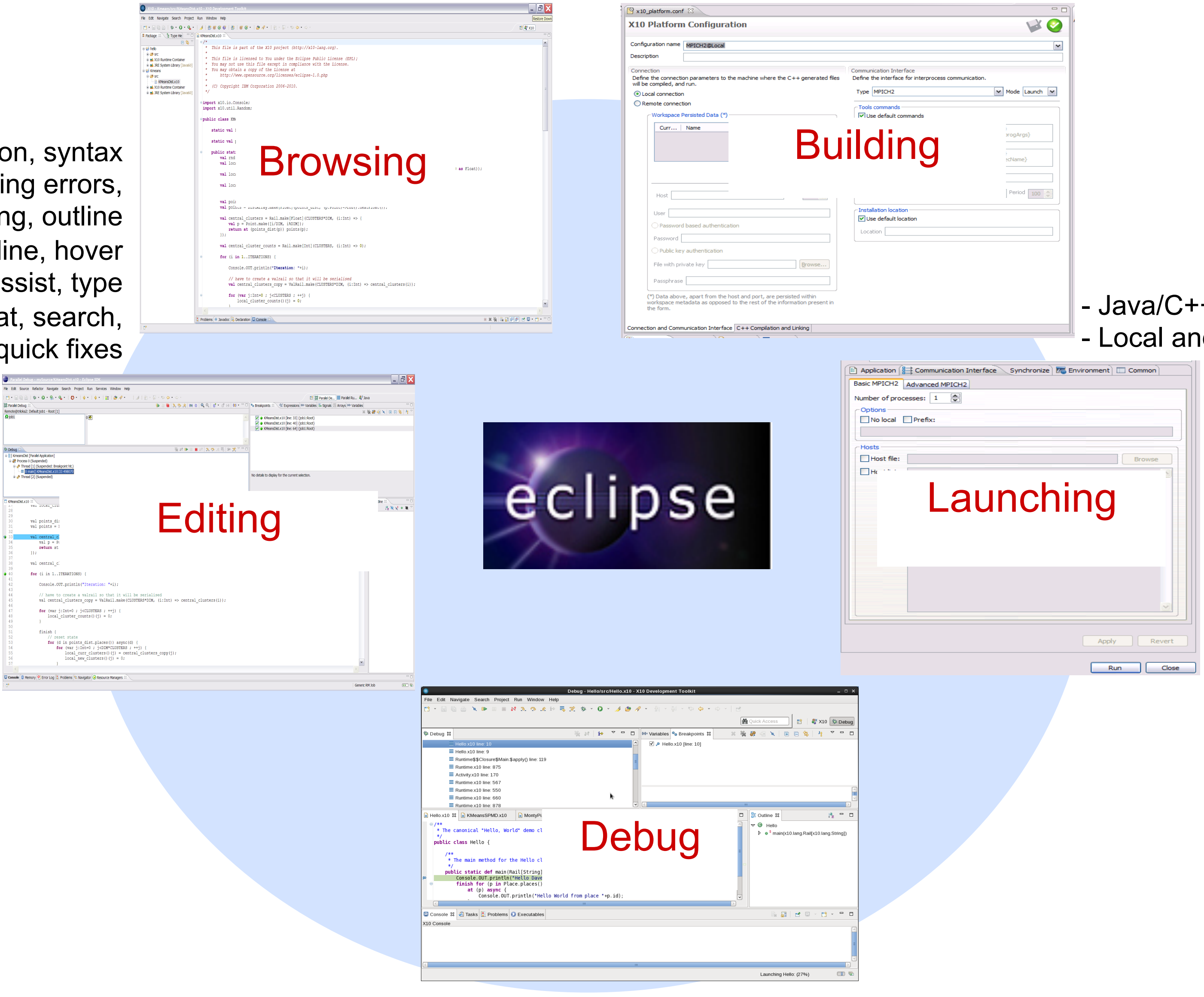
Compilation and Execution



X10DT

Source navigation, syntax highlighting, parsing errors, folding, hyperlinking, outline and quick outline, hover help, content assist, type hierarchy, format, search, call graph, quick fixes

- Java/C++ support
- Local and remote



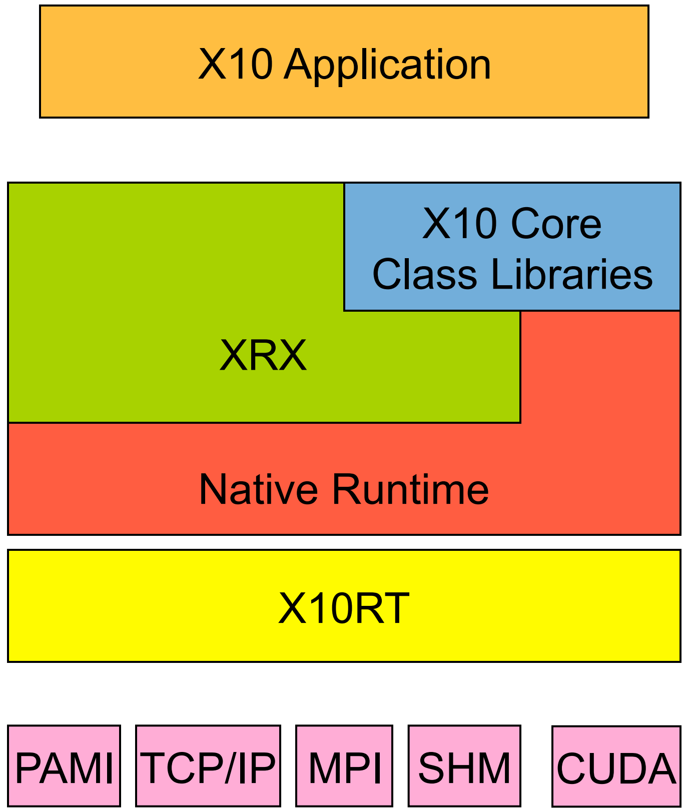
```
/**
 * The canonical "Hello, World" demo cl
 */
public class Hello {

    /**
     * The main method for the Hello cl
     */
    public static def main(Rail[String]
        Console.OUT.println("Hello World
        finish for (p in Place.places()
            at (p) async {
                Console.OUT.println("Hello World from place "+p.id);
```

Implementation

Runtime

- X10RT (X10 runtime transport)
  - core API: active messages
  - extended API: collectives & RDMAs
  - emulation layer
  - two versions: C (+JNI bindings) or pure Java
- Native runtime
  - processes, threads, atomic ops
  - object model (layout, RTTI, serialization)
  - two versions: C++ and Java
- XRX (X10 runtime in X10)
  - async, finish, at, when, atomic
  - X10 code compiled to C++ or Java
- Core X10 libraries
  - x10.array, io, util, util.concurrent

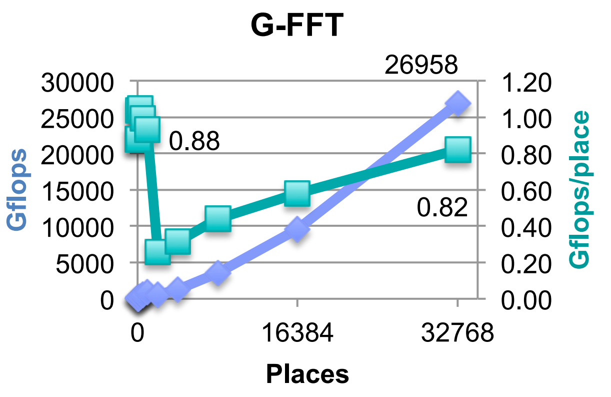


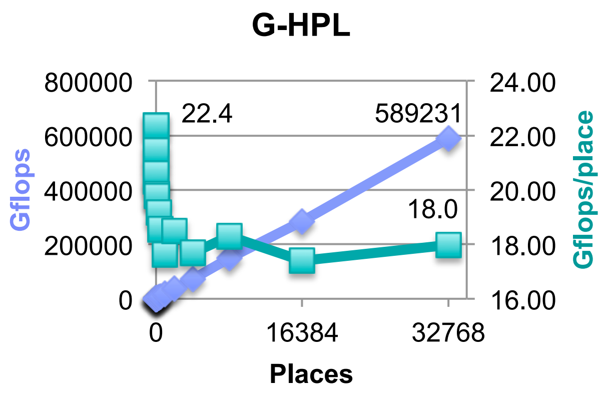
APGAS Constructs

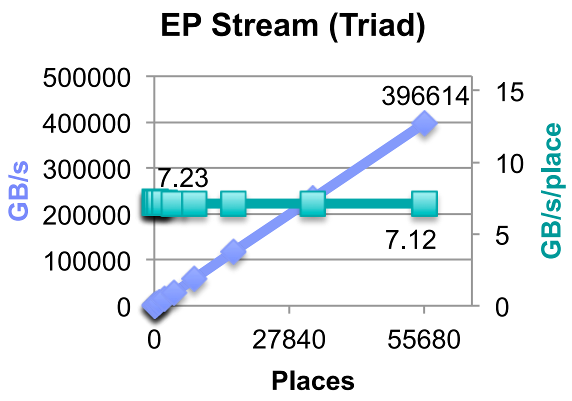
- One process per place
- Local tasks: async & finish
  - thread pool; cooperative work-stealing scheduler
- Remote tasks: at(p) async
  - source side: synthetize active message
    - async id + serialized heap + control state (finish, clocks)
    - compiler identifies captured variables (roots); runtime serializes heap reachable from roots
  - destination side: decode active message
    - polling (when idle + on runtime entry)
- Distributed finish
  - complex and potentially costly due to message reordering
  - pattern-based specialization; program analysis

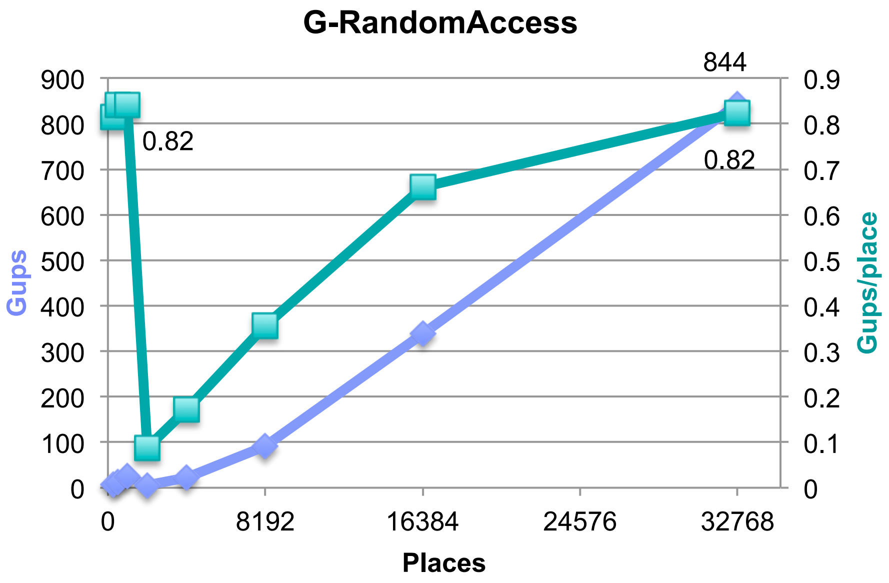
Applications

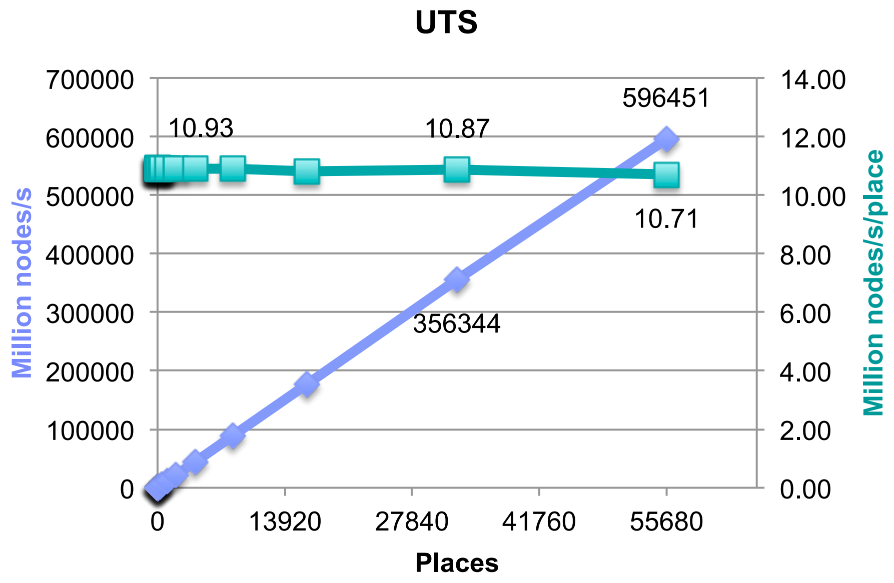
HPC Challenge 2012 – X10 at Petascale – Power 775











Community and Applications

- X10 applications and frameworks
  - ANUChem, [Milthorpe IPDPS 2011], [Limpanuparb JCTC 2013]
  - ScaleGraph [Dayarathna et al X10 2012]
  - Invasive Computing [Bungartz et al X10 2013]
  - XAXIS [Suzumura et al X10 2012]; used in Megaffic (IBM Mega Traffic Simulator)
  - Global Matrix Library: distributed sparse and dense matrices
  - Global Load Balancing [Zhang et al PPAA 2014]
- X10 as a coordination language for scale-out
  - SatX10 [Bloom et al SAT'12 Tools]
  - Power system contingency analysis [Khaitan & McCalley X10 2013]
- X10 as a target language
  - MatLab [Kumar & Hendren X10 2013]
  - StreamX10 [Wei et al X10 2012]

2014/2015 Highlights

2014/2015 Community Papers

- A Resilient Framework for Iterative Linear Algebra Applications in X10 (PDSEC’15)
- High Throughput Indexing for Large-scale Semantic Web Data (SAC'15)
- Malleable Invasive Applications (ATPS'15)
- Dynamic deadlock verification for general barrier synchronisation (PPoPP’15)
- Solving Hard Stable Matching Problems via Local Search and Cooperative Parallelization (AAAI-15)
- IMSuite: A benchmark suite for simulating distributed algorithms (Journal of Parallel and Distributed Computing)
- Design and Evaluation of Parallel Hashing over Large-scale Data (HiPC'14)
- Towards Scalable Distributed Graph Database Engine for Hybrid Clouds (DataCloud’14)
- Massively Parallel Reasoning under the Well-Founded Semantics using X10 (ICTAI'14)
- Robust and Skew-resistant Parallel Joins in Shared-Nothing Systems (CIKM'14)
- MIX10: compiling MATLAB to X10 for high performance (OOPSLA'14)
- Productivity in Parallel Programming: A Decade of Progress (ACM Queue)
- Resolutions of the Coulomb Operator: VIII. Parallel implementation using the modern programming language X10 (Journal of Computational Chemistry)
- Scalable Parallel Numerical CSP Solver (CP'14)
- A two-tier index architecture for fast processing large RDF data over distributed memory (HT '14)
- Semantics of (Resilient) X10 (ECOOP’14)

Releases

- X10 2.5.0 – October 2014
  - elastic X10 (includes backward incompatible changes to Place API)
  - X10 runtime as a service

- X10 2.5.1 – December 2014
  - major upgrade of resilient X10 (fixed thread leak)
  - grid X10 (resilient data store, Hazelcast integration)

- X10 2.5.2 – March 2015
  - ghost regions for distributed arrays
  - MPI 3 transport

- X10 2.5.3 – June 2015
  - new parser
    - dramatically improved error recovery in X10DT
    - minor syntax tweaks (backward incompatible)

Grid X10

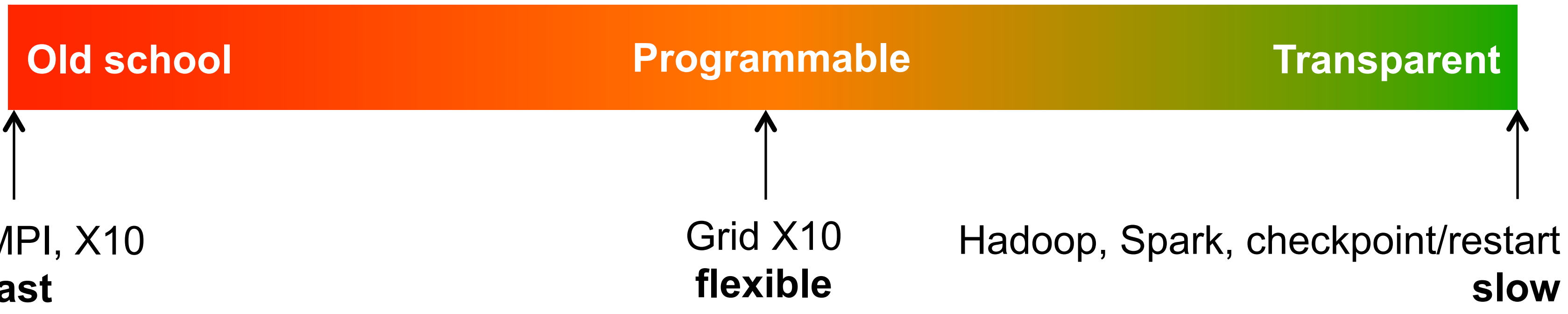
Problem

- failures are increasingly common in distributed systems
- available resources vary dynamically

Our Approach

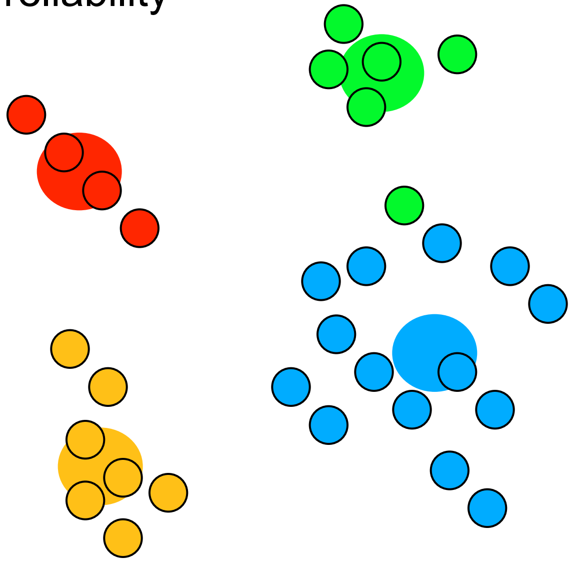
- support programming fault tolerance and resource management
- bake in only fundamental capabilities, build the rest as libraries

Design space



Motivation

- Application-level failure recovery
  - if the computation is approximate: trade accuracy for reliability
  - if the computation is repeatable: replay it
  - if lost data is unmodified: reload it
  - if data is mutated: checkpoint it
- Example: K-Means clustering
  - algorithm: iterative refinement computation
    - massively parallel
    - large immutable input data
    - small state
    - frequent global synchronization
  - resilient algorithm: checkpoint state after each iteration but not input data
    - on failure divide inputs associated with lost place into remaining places
      - reload state from checkpoint
      - reload lost input data from disk



Programming Model

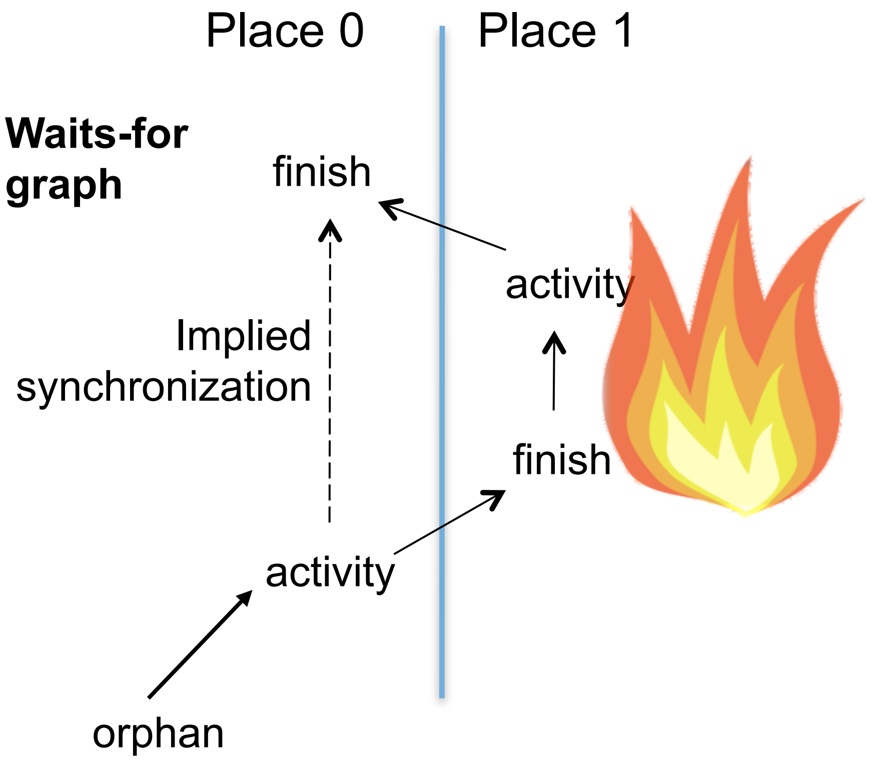
- Place granularity
  - places can be added and removed
  - fail-stop model
  - report changes with exceptions (loss) and callbacks (loss or addition)
- Resilient control
  - execution continues at healthy places
  - execution order is preserved (happen-before invariance)
- Resilient data
  - data at failed place is lost
  - data in resilient store is preserved
    - resilient store does not belong to any place (but shards can be hosted within X10 places)

Same language; richer semantics (extension); substantial changes to runtime

Happens Before Invariance

Failure of a place should not alter the happens before relationship.

```
val gr = GlobalRef(new Cell[Int](0));
try {
    finish at (Place(1)) async {
        finish at (Place(0)) async {
            gr()(10); // A
        }
    }
} catch (e:MultipleExceptions) { }
gr()(3); // B
assert gr()() != 10;
```



A happens before B, **even if place 1 dies**.

Without this property, avoiding race conditions would be very hard.

But guaranteeing it is non-trivial, requires more runtime machinery.

Runtime Improvements for Resilient X10

- Distributed resilient finish requires cross-place transactional updates of finish control state to ensure consistent view of happens-before relationship on Place failure
  - Initial Resilient X10 implementation relied on synchronous messages which could degenerate into needing a thread for every X10 activity
- New implementation overcomes this issuecomponents to allow transactional update of finish control state without needing unbounded number of threads
  - Stratify remote messages into two classes: immediate and normal
  - Immediate messages must be non-blocking, finite, handled by dedicated immediate network processing thread(s) in each Place
  - Tasks waiting on a response to an immediate message can safely suspend without spanwing a new thread to ensure global progress
  - Redesign finish state update protocols to only use immediate messages
- Resilient X10 and classic X10 now support the same levels of concurrency and remote activity creation

Elastic X10: YARN integration

- New integration with the YARN cluster manager, allowing X10 programs to be launched on a YARN-managed (Hadoop 2.x) cluster.
- Added the ability for an X10 program to request new places from the launcher, so we can add places on-demand. Resource requests are handled by YARN, and new places join the existing ones.
- Any X10 place or its host machine can fail at any time, and YARN can reuse the newly freed resources.
- YARN's design does not provide resiliency for the ResourceManager or ApplicationMaster, which monitor the cluster itself and the individual containers holding the X10 runtimes. We do not attempt to improve this, so these are single points of failure, but these are outside of X10 itself.
- X10 programs are launched on YARN by specifying -x10rt yarn as an argument to the x10 script.