

Reflections on X10

Towards Performance and Productivity at Scale

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

Acknowledgments

- X10 research project started in 2004...
- Major technical contributions from many people
 - IBM X10 team (current and former) > 50 people
 - External research collaborators
 - X10 user community
 - IBM PERCS project team
- Support from
 - DARPA HPCS
 - US Department of Energy
 - US Air Force Research Lab
 - IBM

Talk Objectives

- Fundamental challenges: What is X10 all about?
- Insights into the APGAS Programming Model
 - Applicability beyond the X10 language
 - Realization of APGAS Model in X10
- X10 In Use
 - X10 At Scale
 - X10 + Java ➔ Commercial Applications
 - X10 Community Activities
- Language design and implementation alternatives and their implications

Talk Outline

- X10 Programming Model
 - Overview
 - Code snippets
 - Larger examples
- X10 in Use
 - Scaling X10
 - Agent simulation (Megaffic/XAXIS)
 - Main Memory Map Reduce (M3R)
- Implementing X10
 - Compiling X10
 - X10 Runtime
- Final Thoughts

X10 Genesis: DARPA HPCS Program (2004)

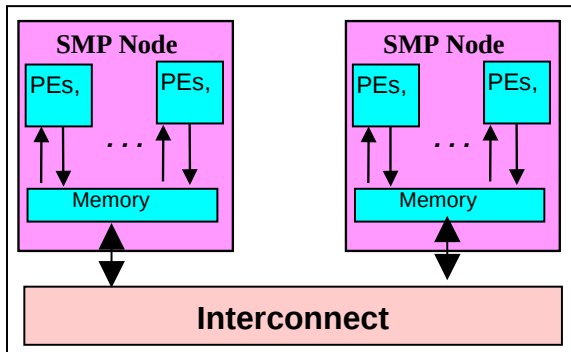
High **Productivity** Computing Systems

Central Challenge: Productive Programming of Large Scale Supercomputers

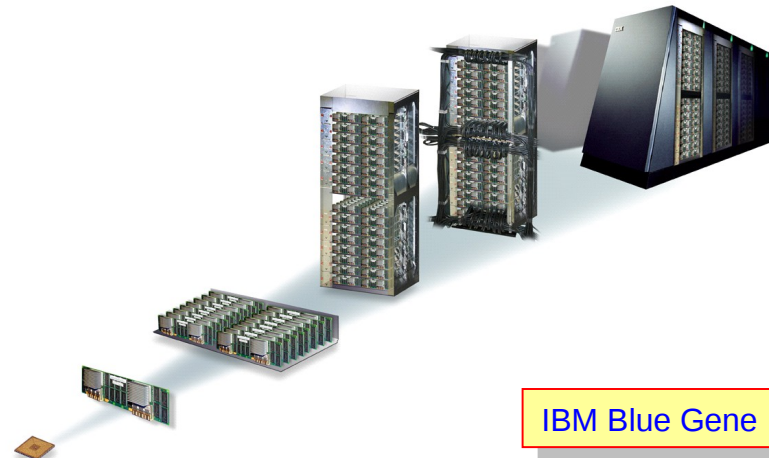
– Clustered systems

- 1000's of SMP Nodes connected by high-performance interconnect
- Large aggregate memory, disk storage, etc.

SMP Clusters



Massively Parallel Processor Systems



IBM Blue Gene

Programming Model Challenges

- Scale Out
 - Program must run across many nodes (distributed memory; network capabilities)
- Scale Up
 - Program must exploit multi-core and accelerators (concurrency; heterogeneity)
- Both Productivity and Performance
 - Bring modern commercial tooling/languages/practices to HPC programmers
 - Support high-level abstractions, code reuse, rapid prototyping
 - While still enabling full utilization of HPC hardware capabilities at scale

X10 Performance and Productivity at Scale

- X10 Language
 - 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)
- X10 Tool chain
 - Open source compiler, runtime, class libraries
 - Dual path: *Managed X10* (X10→Java) and *Native X10* (X10 →C++)
 - Command-line tooling (compile/execute)
 - Linux, Mac OSX, Windows, AIX; x86, Power; sockets, MPI, PAMI, DCMF
 - Eclipse-based IDE (edit/browse/compile/execute)

Partitioned Global Address Space (PGAS) Languages

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

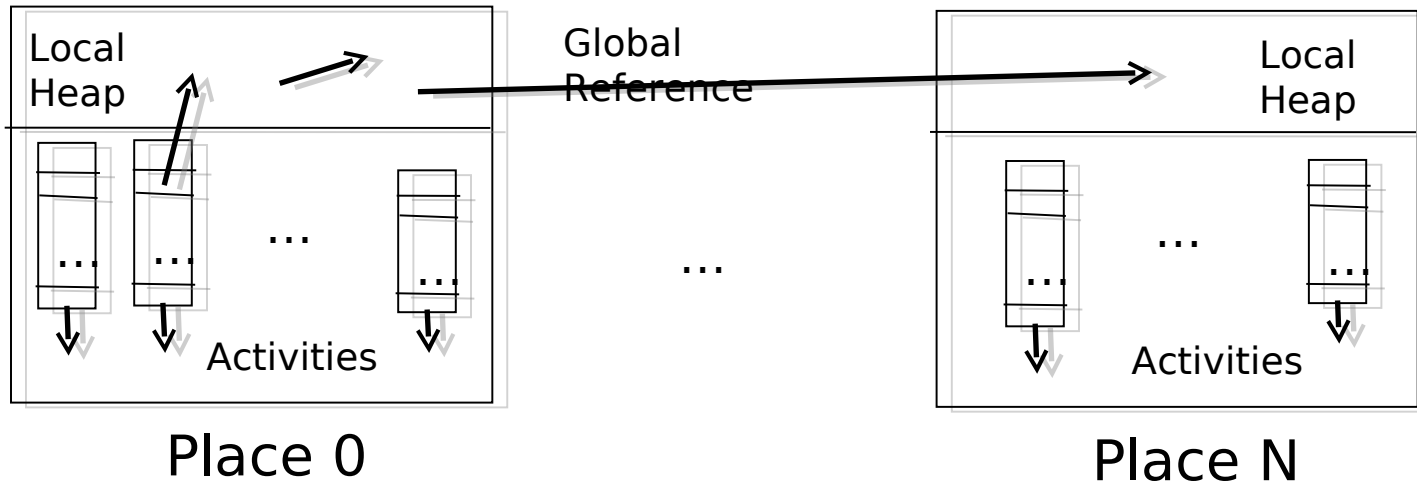
PGAS combines a single global address space with locality awareness

- 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]

Concurrency: async and finish

async S

- Creates a new child activity that executes statement **S**
- Returns immediately
- **S** may reference variables in enclosing blocks
- Activities cannot be named/cancelled.

finish S

- Execute **S**, but wait until all (transitively) spawned asyncs have terminated.
- **finish** is useful for expressing “synchronous” operations on (local or) remote data.
- Rooted exception model
 - Trap all exceptions and throw an (aggregate) exception if any spawned async terminates exceptionally.

```
// Compute the Fibonacci
// sequence in parallel.
def fib(n:Int):Int {
  if (n < 2) return 1;
  val f1:Int;
  val f2:Int;
  finish {
    async f1 = fib(n-1);
    f2 = fib(n-2);
  }
  return f1+f2;
}
```

Atomicity: atomic and when

atomic S

- Execute statement **S** atomically
- Atomic blocks are conceptually executed in a serialized order with respect to all other atomic blocks in a Place: **isolation** and **weak atomicity**.
- An atomic block body (**S**) must be **nonblocking**, **sequential**, and **local**

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 and must be **nonblocking**, **sequential**, **local**, and **pure**

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

```
class OneBuffer {
    var datum:Object = null;
    var filled:Boolean = false;
    ...
    def receive():Object {
        when (filled) {
            val v = datum;
            datum = null;
            filled = false;
            return v;
        }
    }
}
```

Atomicity: atomic and when (and locks)

- X10 currently implements **atomic** and **when** trivially with a per-Place lock
 - All **atomic/when** statements serialized within a Place
 - Scheduler re-evaluates pending when conditions on exit of atomic section
 - Poor scalability on multi-core nodes; **when** especially inefficient
- Pragmatics: class library provides lower-level alternatives
 - `x10.util.concurrent.Lock` – pthread mutex
 - `x10.util.concurrent.AtomicInteger` et al – wrap machine atomic update operations
- An aspect of X10 where our implementation has not yet matched our ambitions...
 - Area for future research
 - Natural fit for transactional memory (STM/HTM/Hybrid)

Distribution: Places and at

at (p) S

- Execute statement **S** at place **p**
- Current activity is blocked until **S** completes
- Deep copy of local object graph to target place; the variables referenced from **S** define the roots of the graph to be copied.
- **GlobalRef[T]** used to create remote pointers across at boundaries

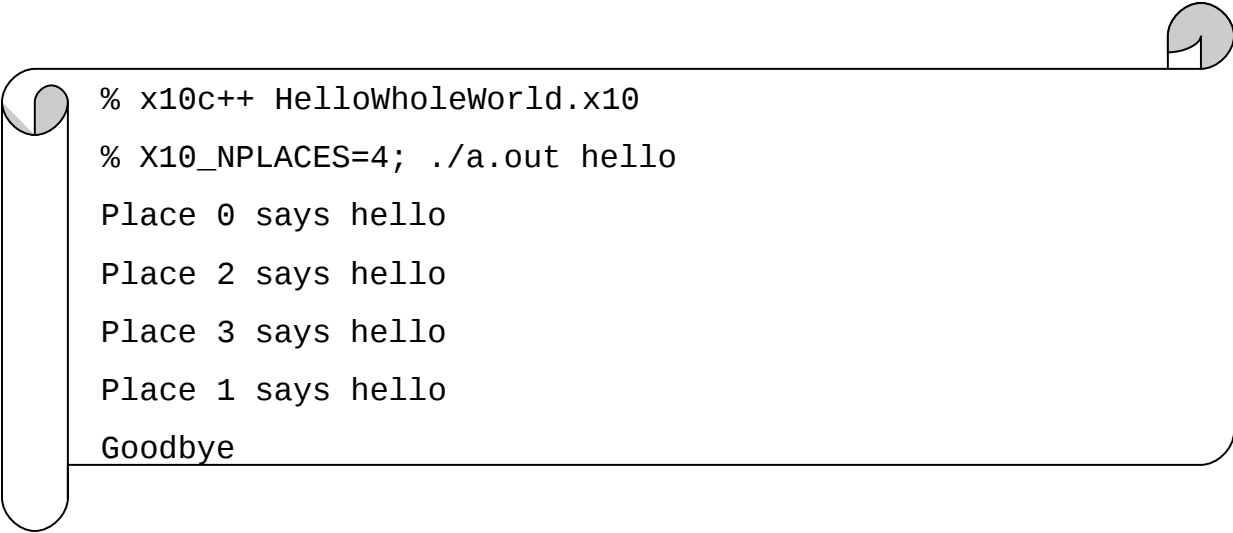
```
class C {  
    var x:int;  
    def this(n:int) { x = n; }  
}  
  
// Increment remote counter  
def inc(c:GlobalRef[C]) {  
    at (c.home) c().x++;  
}  
  
// Create GR of C  
static def make(init:int) {  
    val c = new C(init);  
    return GlobalRef[C](c);  
}
```

X10's Distributed Object Model

- Objects live in a single place
- Objects can only be accessed in the place where they live
- Cross-place references
 - **GlobalRef[T]** reference to an object at one place that can be transmitted to other places
 - **PlaceLocalHandle[T]** “handle” for a distributed data structure with state (objects) at many places. Optimized representation for a collection of **GlobalRef[T]** (one per place).
- Implementing **at**
 - Compiler analyzes the body of **at** and identifies roots to copy (exposed variables)
 - Complete object graph reachable from roots is serialized and sent to destination place
 - A new (unrelated) copy of the object graph is created at the destination place
- Controlling object graph serialization
 - Instance fields of class may be declared transient (won't be copied)
 - **GlobalRef[T]/PlaceLocalHandle[T]** serializes id, not the referenced object
 - Classes may implement CustomSerialization interface for arbitrary user-defined behavior
- Major evolutions in object model: X10 1.5, 1.7, 2.0, and 2.1 (fourth time is the charm? ☺)

Hello Whole World

```
1/class HelloWorld {  
2/  public static def main(args:Rail[String]) {  
3/    finish  
4/      for (p in Place.places())  
5/        at (p)  
6/          async  
7/            Console.OUT.println(p+" says " +args(0));  
8/    Console.OUT.println("Goodbye");  
9/  }  
10/}
```



```
% x10c++ HelloWorld.x10  
% X10_NPLACES=4; ./a.out hello  
Place 0 says hello  
Place 2 says hello  
Place 3 says hello  
Place 1 says hello  
Goodbye
```

APGAS Idioms

- Remote evaluation

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

- Active message

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

- Recursive parallel decomposition

```
def fib(n:Int):Int {
  if (n < 2) return 1;
  val f1:Int;
  val f2:Int;
  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;
```

- Data exchange

```
// swap row i local and j remote
val h = here;
val row_i = rows()(i);
finish at(p) async {
  val row_j = rows()(j);
  rows()(j) = row_i;
  at(h) async row()(i) = row_j;
}
```

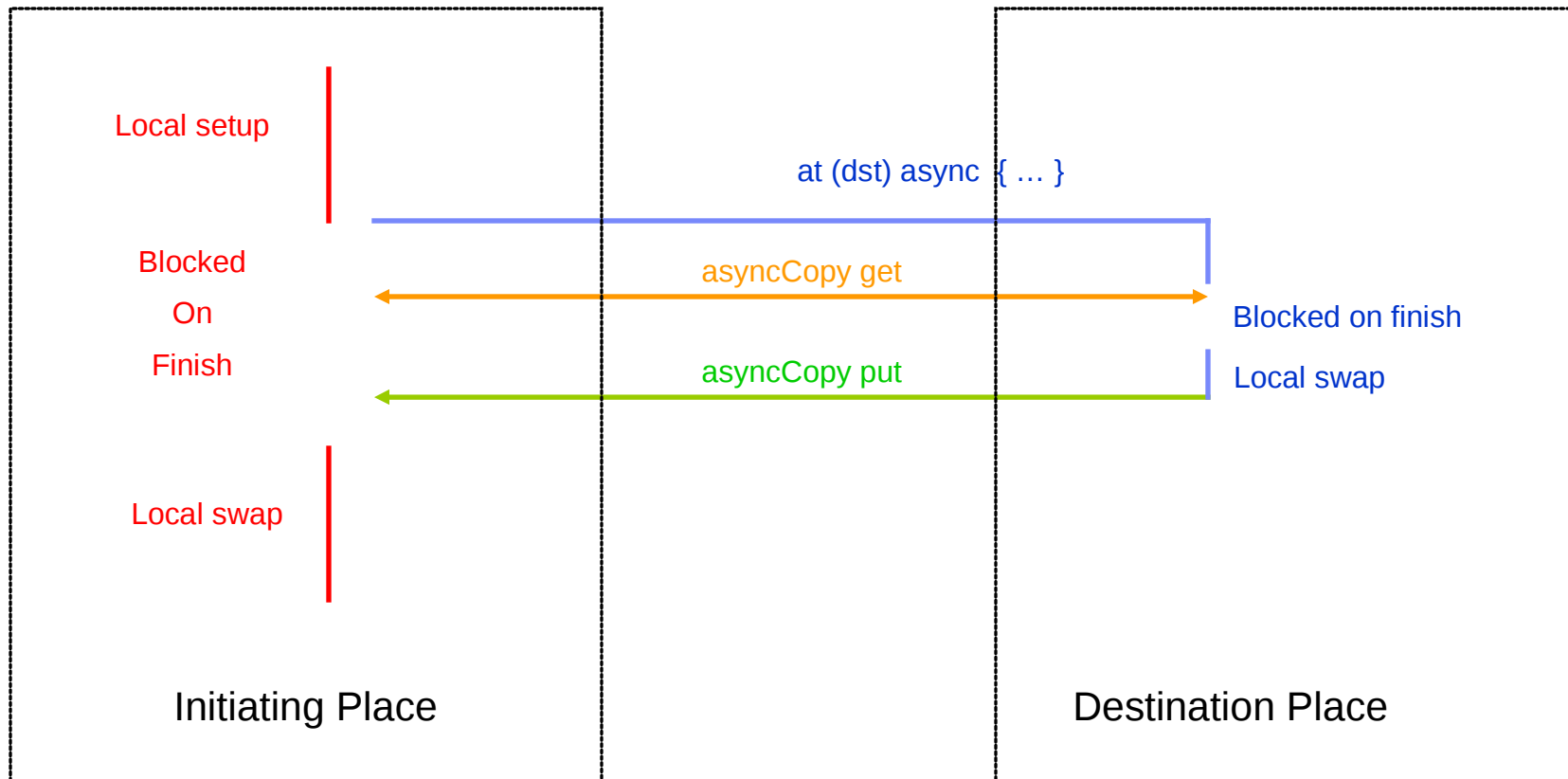
A handful of key constructs cover a broad spectrum of patterns

Outline

- X10 Programming Model
 - Overview
 - Code snippets
 - Larger examples
 - Row Swap from LU
 - Unbalanced Tree Search
- X10 in Use
 - Scaling X10
 - Agent simulation (Megaffic/XAXIS)
 - Main Memory Map Reduce (M3R)
- Implementing X10
 - Compiling X10
 - X10 Runtime
- Final Thoughts

Row Swap from LU Benchmark

- Programming problem
 - Efficiently exchange swap rows in distributed matrix with another Place
 - Exploit network capabilities
 - Overlap communication with computation



Row Swap from X10 LU Benchmark

```
// swap row with index srcRow located here with row with index dstRow located at place dst
def rowSwap(matrix:PlaceLocalHandle[Matrix[Double]], srcRow:Int, dstRow:Int, dst:Place) {
  val srcBuffer = buffers();
  val srcBufferRef = new RemoteRef(srcBuffer);
  val size = matrix().getRow(srcRow, srcBuffer);
  finish {
    at (dst) async {
      finish {
        val dstBuffer = buffers();
        Array.asyncCopy[Double](srcBufferRef, 0, dstBuffer, 0, size);
      }
      matrix().swapRow(dstRow, dstBuffer);
      Array.asyncCopy[Double](dstBuffer, 0, srcBufferRef, 0, size);
    }
  }
  matrix().setRow(srcRow, srcBuffer);
}
```

Global Load Balancing

- **Unbalanced Tree Search Benchmark**
 - count nodes in randomly generated tree
 - separable cryptographic random number generator
 - highly unbalanced trees
 - unpredictable
 - tree traversal can be easily relocated (no data dependencies, no locality)
- **Problems to be solved:**
 - Dynamically balance work across a large number of places efficiently
 - Detect termination of the computation quickly.
- **Conventional Approach:**
 - Worker steals from randomly chosen victim when local work is exhausted.
 - Key problem: When does a worker know there is no more work?

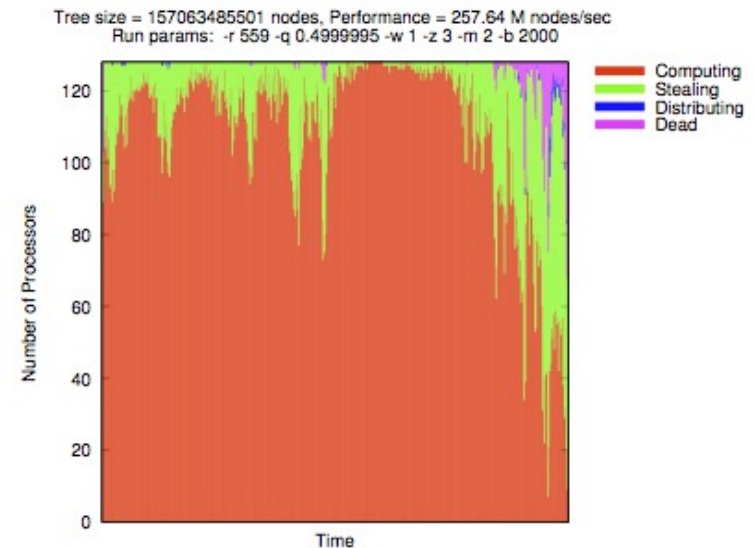
Global Load Balance: The Lifeline Solution

- Intuition: X0 already has a framework for distributed termination detection – **finish**. Use it!
- But this requires activities terminate at some point!
- Idea: Let activities terminate after w rounds of failed steals. But ensure that a worker with some work can “distribute” work (= use **at(p) async S**) to nodes known to be idle.
 - Lifeline graphs. Before a worker dies it creates z lifelines to other nodes.
 - These form a distribution “overlay” graph.
 - What kind of graph? Need low out-degree, low diameter graph.
Our paper/implementation uses hyper-cube.

Scalable Global Load Balancing

Unbalanced Tree Search

- Lifeline-based global work stealing [PPoPP'11]
 - n random victims then p lifelines (hypercube)
 - fixed graph with low degree and low diameter
 - synchronous (steal) then asynchronous (deal)
- Root finish accounts for
 - startup asyncs + lifeline asyncs
 - not random steal attempts
- Compact work queue (for shallow trees)
 - represent intervals of siblings
 - thief steals half of each work item
- Sparse communication graph
 - bounded list of potential random victims
 - finish trades contention for latency



genuine APGAS algorithm

UTS

Main Loop

```
def process() {
  alive = true;
  while (!empty()) {
    while (!empty()) { processAtMostN(); Runtime.probe(); deal(); }
    steal();
  }
  alive = false;
}

def steal() {
  val h = here.id;
  for (i:Int in 0..w) {
    if (!empty()) break;
    finish at (Place(victims(rnd.nextInt(m)))) async request(h, false);
  }
  for (lifeline:Int in lifelines) {
    if (!empty()) break;
    if (!lifelinesActivated(lifeline)) {
      lifelinesActivated(lifeline) = true;
      finish at (Place(lifeline)) async request(h, true);
    }
  }
}
```

UTS

Handling Thieves

```
def request(thief:Int, lifeline:Boolean) {
  val nodes = take(); // grab nodes from the local queue
  if (nodes == null) {
    if (lifeline) lifelineThieves.push(thief);
    return;
  }
  at (Place(thief)) async {
    if (lifeline) lifelineActivated(thief) = false;
    enqueue(nodes); // add nodes to the local queue
  } }

def deal() {
  while (!lifelineThieves.empty()) {
    val nodes = take(); // grab nodes from the local queue
    if (nodes == null) return;
    val thief = lifelineThieves.pop();
    at (Place(thief)) async {
      lifelineActivated(thief) = false;
      enqueue(nodes); // add nodes to the local queue
      if (!alive) process();
    } } }
}
```


Outline

- X10 Programming Model
 - Overview
 - Code snippets
 - Larger examples
 - Row Swap from LU
 - Unbalanced Tree Search
- X10 in Use
 - Scaling X10
 - Agent simulation (Megaffic/XAXIS)
 - Main Memory Map Reduce (M3R)
- Implementing X10
 - Compiling X10
 - X10 Runtime
- Final Thoughts

X10 At Scale

X10 has *places* and *asyns* in each place

We want to

- Handle millions of asyns (→ billions)
- Handle tens of thousands of places (→ millions)

We need to

- Scale up
 - shared memory parallelism (today: 32 cores per place)
 - schedule many asyns with a few hardware threads
- Scale out
 - distributed memory parallelism (today: 50K places)
 - provide mechanisms for efficient distribution (data & control)
 - support distributed load balancing

DARPA PERCS Prototype (Power 775)

- Compute Node
 - 32 Power7 cores 3.84 GHz
 - 128 GB DRAM
 - peak performance: 982 Gflops
 - *Torrent* interconnect
- Drawer
 - 8 nodes
- Rack
 - 8 to 12 drawers
- Full Prototype
 - up to 1,740 compute nodes
 - up to 55,680 cores
 - up to 1.7 petaflops
 - 1 petaflops with 1,024 compute nodes



Eight Benchmarks

- HPC Challenge benchmarks
 - Linpack TOP500 (flops)
 - Stream Triad local memory bandwidth
 - Random Access distributed memory bandwidth
 - Fast Fourier Transform mix

- Machine learning kernels
 - KMEANS graph clustering
 - SSCA1 pattern matching
 - SSCA2 irregular graph traversal
 - UTS unbalanced tree traversal

Implemented in X10 as pure scale out tests

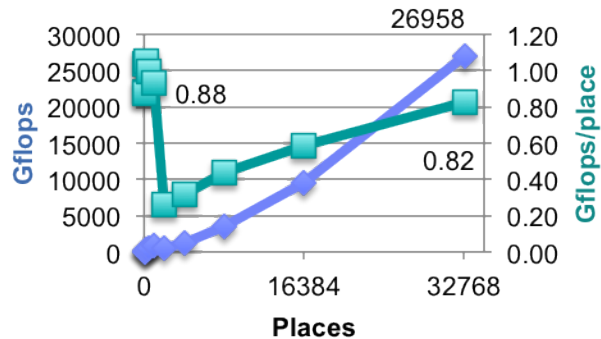
- *One core = one place = one main async*
- *Native libraries for sequential math kernels: ESSL, FFTW, SHA1*

Performance at Scale (Weak Scaling)

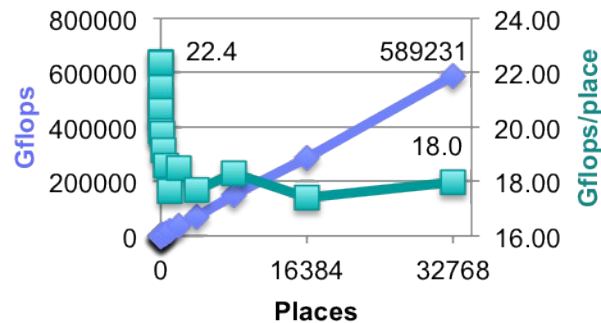
	cores	absolute performance at scale	parallel efficiency (weak scaling)	performance relative to best implementation available
Stream	55,680	397 TB/s	98%	85% (lack of prefetching)
FFT	32,768	27 Tflops	93%	40% (no tuning of seq. code)
Linpack	32,768	589 Tflops	80%	80% (mix of limitations)
RandomAccess	32,768	843 Gups	100%	76% (network stack overhead)
KMeans	47,040	depends on parameters	97.8%	66% (vectorization issue)
SSCA1	47,040	depends on parameters	98.5%	100%
SSCA2	47,040	245 B edges/s	> 75%	no comparison data
UTS (geometric)	55,680	596 B nodes/s	98%	<i>reference code does not scale 4x to 16x faster than UPC code</i>

HPCC Class 2 Competition: Best Performance Award

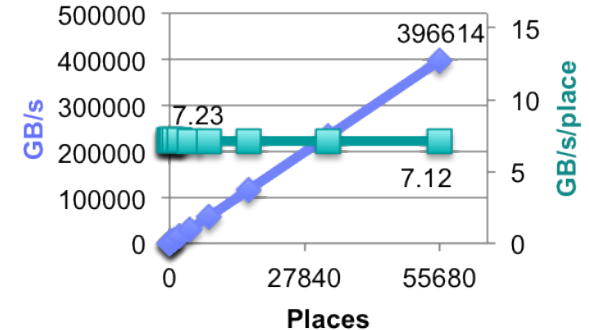
G-FFT



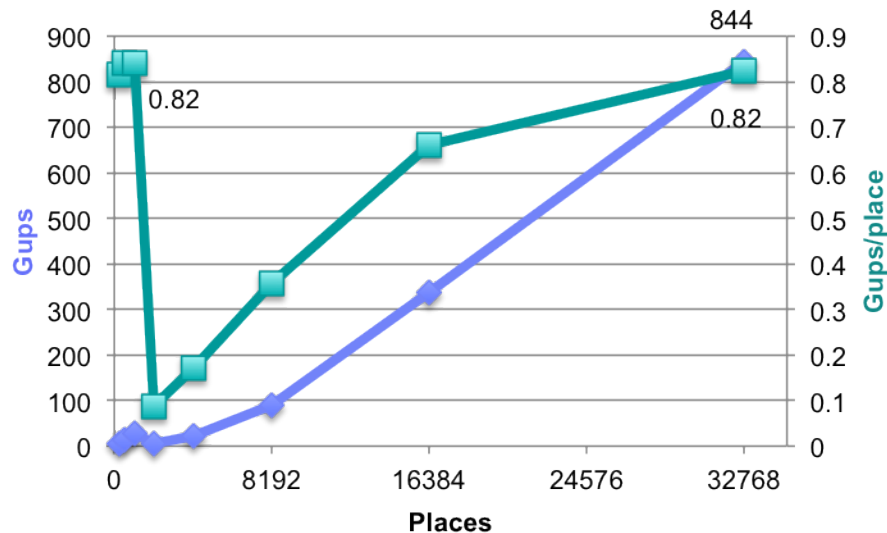
G-HPL



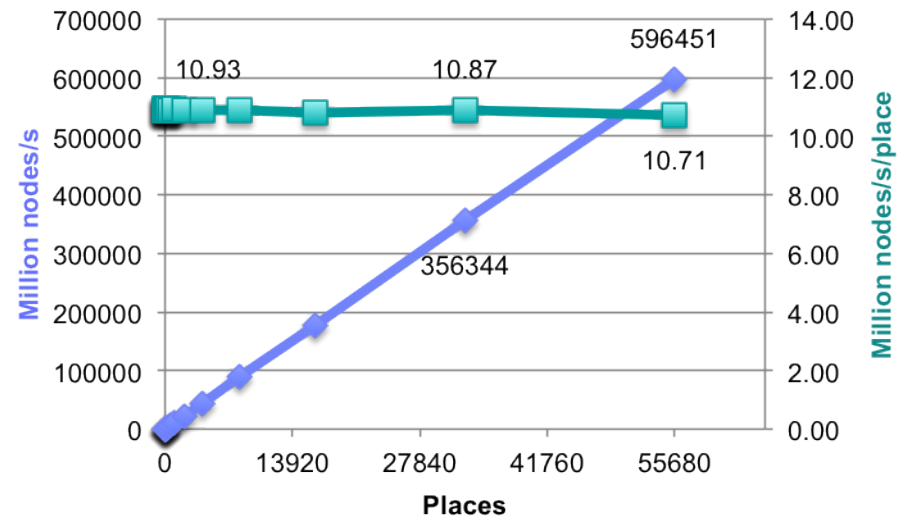
EP Stream (Triad)



G-RandomAccess

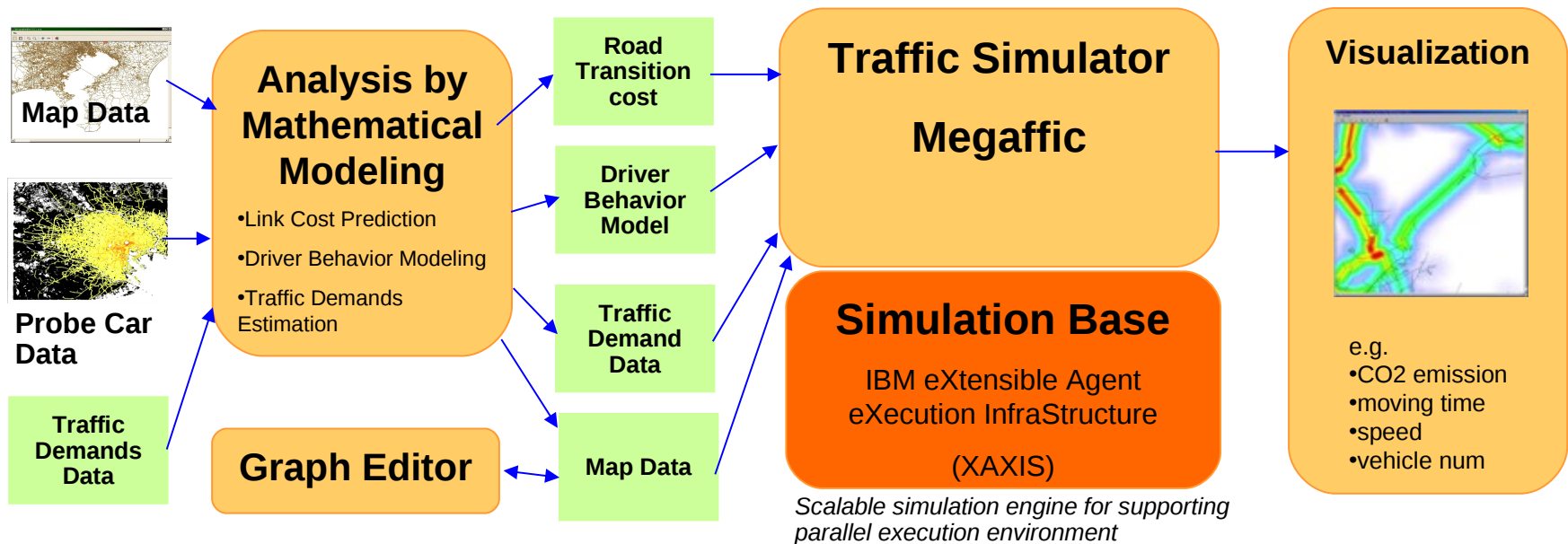


UTS



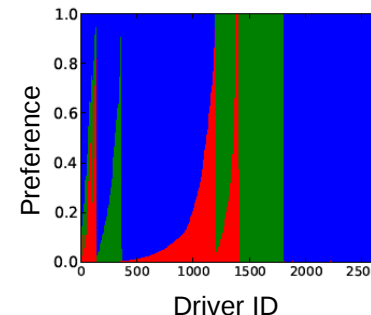
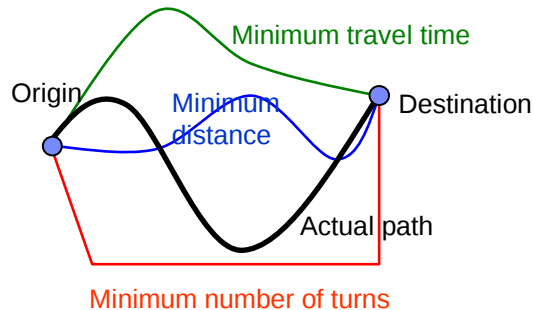
IBM Mega Traffic Simulator (Megaffic)

IBM Research has created a Large-scale multi-agent traffic simulator.



Driver Behavior Modeling:

Automatically generate preferences of drivers from probe car data. A preference is a set of weights over different policies such as minimum travel time, minimum # of turns, etc. The weights are automatically learned from the data.

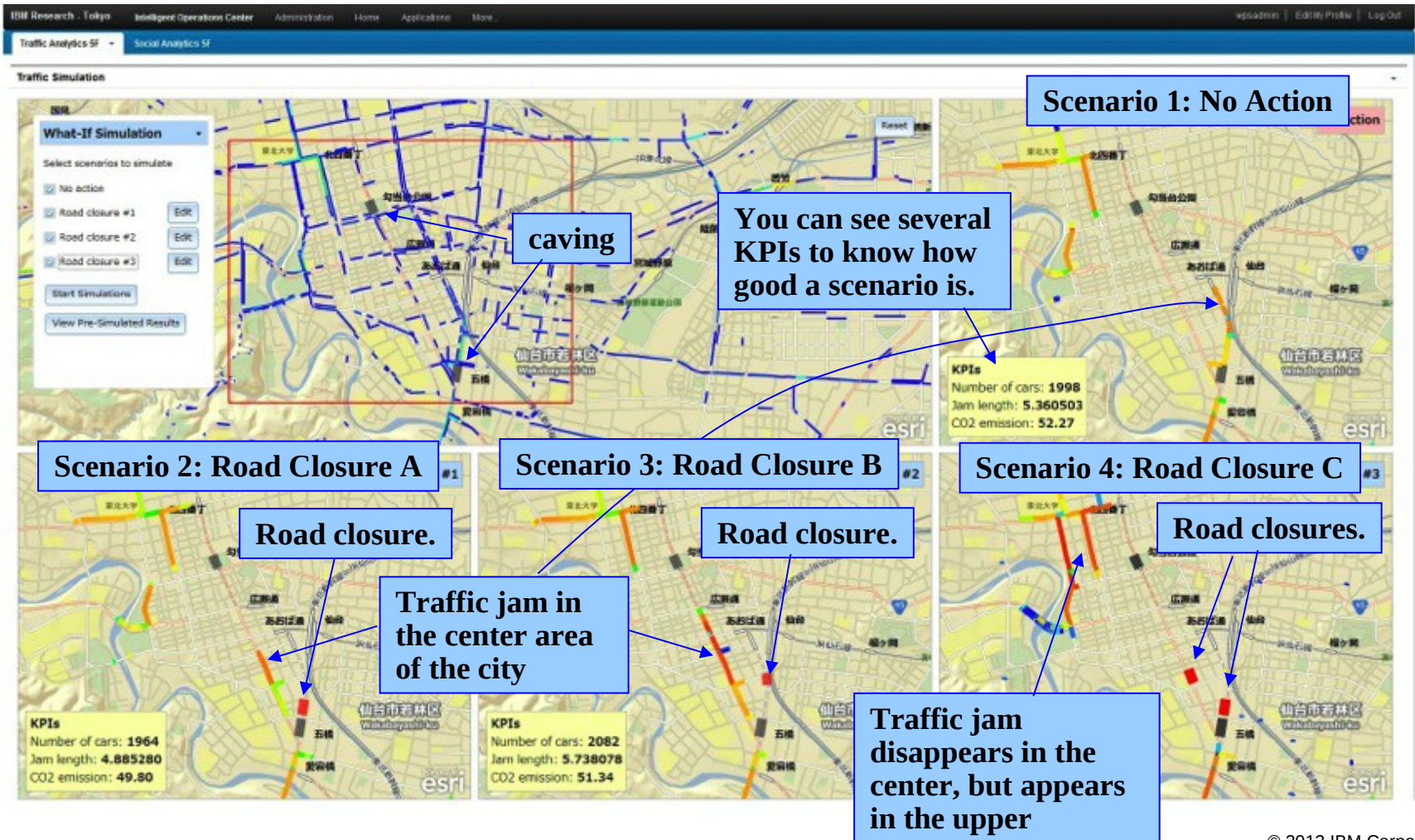


This project is funded by the PREDICT project of Ministry of Internal Affairs and Communications, Japan.



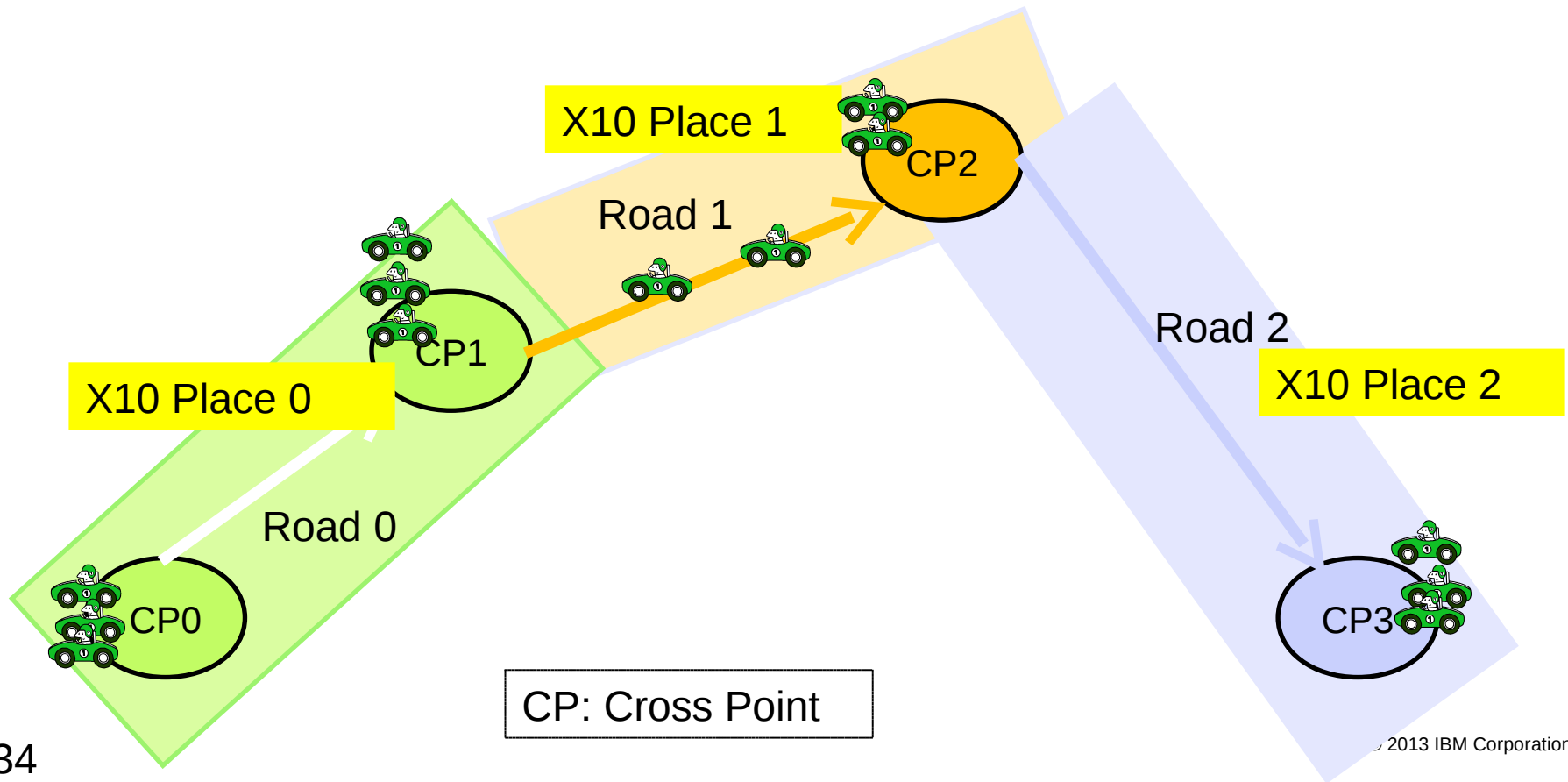
City Planning Example: What-if Simulation of Sendai City

After the Great East-Japan Earthquake, there were some cavings on the roads in Sendai city, and there was heavy traffic jam. The below picture shows simulation results of different (imaginary) scenarios of actions to reduce traffic jam by road closure. This what-if simulation capability can help traffic administrators pick the best decision out of some choices even in new situations.



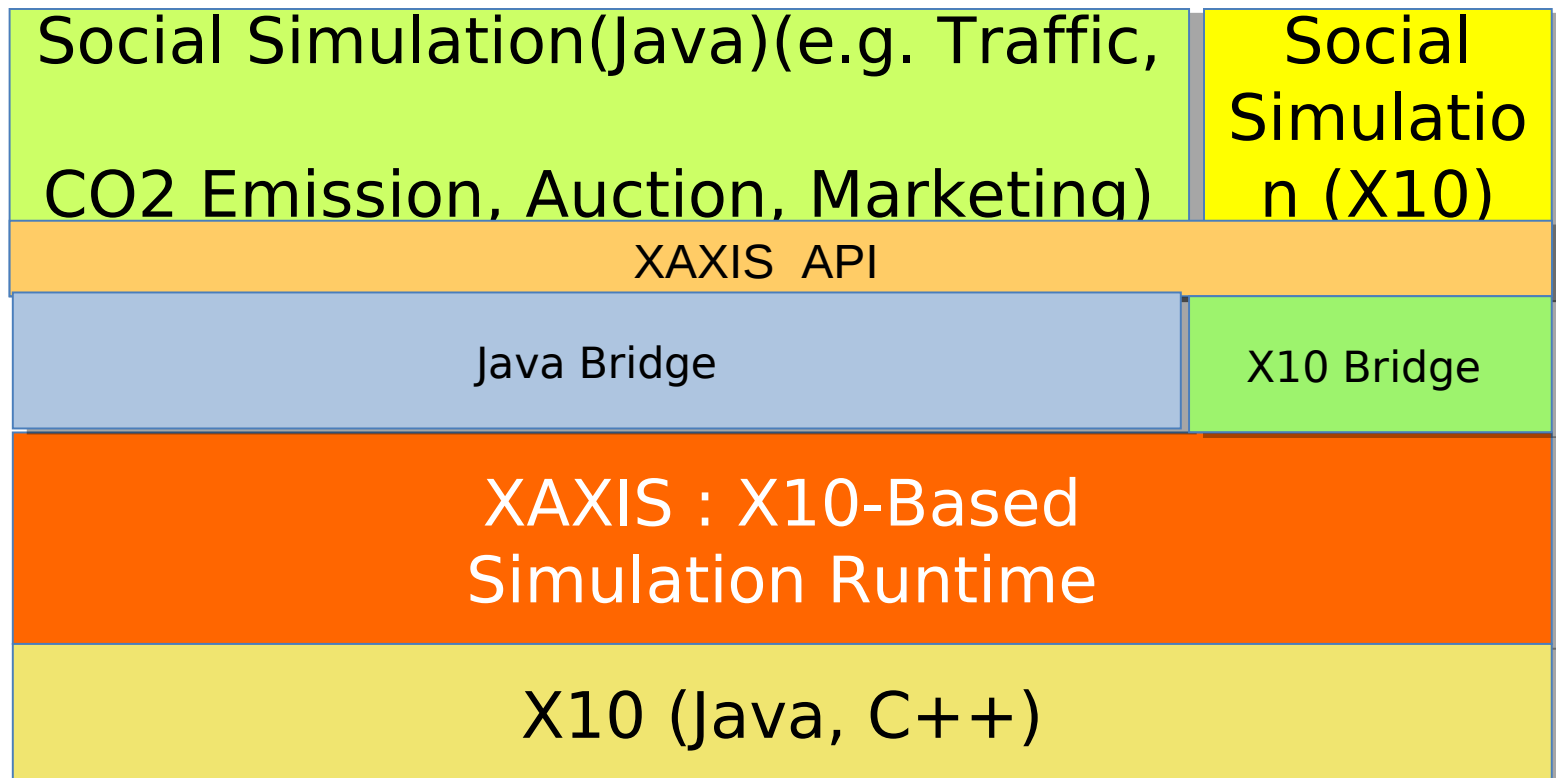
Why X10?

- X10/Java interoperability lowered risk of adopting X10
 - Scale out existing Java-based simulation framework with APGAS primitives
 - Gradual porting of core code to X10 (enabled Native X10 & BlueGene/Q)



XAXIS Software Stack

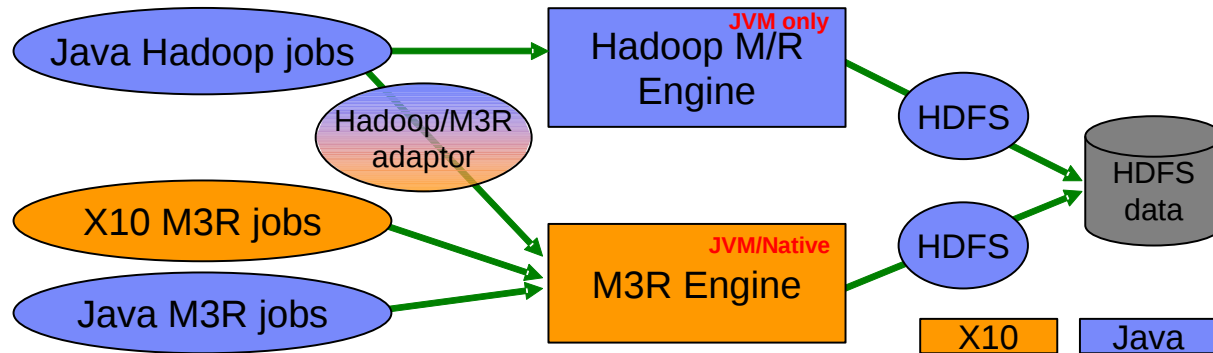
- The following diagram illustrates the software stack of XAXIS and its applications.



M3R – A Hadoop re-implementation in Main Memory, using X10

- Hadoop
 - Popular Java API for Map/Reduce programming
 - Out of core, resilient, scalable (1000 nodes)
 - Based on HDFS (a resilient distributed filesystem)
- M3R/Hadoop
 - Reimplementation of Hadoop API using Managed X10 (X10 compiled to Java)
 - X10 provides scalable multi-JVM runtime with efficient communication
 - Existing Hadoop 1.0 applications 'just work'
 - Reuse HDFS (and some other parts of Hadoop)
 - In-memory: problem size must fit in aggregate cluster RAM
 - Not resilient: cluster scales until MTBF barrier
 - But considerably faster (closer to HPC speeds)
 - Trade resilience for performance (both latency and throughput)

M3R Architecture



- The core M3R engine provides X10 and Java Map/Reduce APIs against which application programs can be written.
- On top of the engine, a Hadoop API compatibility layer allows unmodified Hadoop Map/Reduce jobs to be executed on the engine.
- The compatibility layer is written using a mix of X10 and Java code and heavily uses the Java interoperability capabilities of Managed X10.

Intuition: Where does M3R gain performance relative to Hadoop?

- **Reducing Disk I/O**
- **Reducing network communication**
- **Reducing serialization/deserialization**
 - Reduce translation of object graphs to byte buffers & back
- **Reducing “other” costs**
 - Job submission speed
 - Startup time (JVM reuse synergistic with JIT compilation)
 - ...
- **Partition Stability (iterative jobs)**
 - Enable application programmer to “pin” data in memory within Hadoop APIs
 - The reducer associated with a given partition number will always be run in the same JVM
- For details, see [Shinnar et al VLDB’12]

Additional X10 Community Applications

- Additional X10 Applications/Frameworks
 - ANUChem <http://cs.anu.edu.au/~Josh.Milthorpe/anuchem.html>, [Milthorpe IPDPS 2013]
 - ScaleGraph <https://sites.google.com/site/scalegraph/> [Dayarathna et al X10 2012]
 - Invasive Computing [Bungartz et al X10 2013]
- X10 as a coordination language for scale-out
 - SatX10 <http://x10-lang.org/satx10>, [SAT'12 Tools]
 - Power system contingency analysis [Khaitan & McCalley X10 2013]
- X10 as a target language
 - MatLab <http://www.sable.mcgill.ca/mclab/mix10.html>, [Kumar & Hendren X10 2013]
 - StreamX10 [Wei et al X10 2012]
- X10 Publications: <http://www.x10-lang.org/x10-community/publications-using-x10.html>

Outline

- X10 Programming Model
 - Overview
 - Code snippets
 - Larger examples
 - Row Swap from LU
 - Unbalanced Tree Search
- X10 in Use
 - Scaling X10
 - Agent simulation (Megaffic/XAXIS)
 - Main Memory Map Reduce (M3R)
- Implementing X10
 - Compiling X10
 - X10 Runtime
- Final Thoughts

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

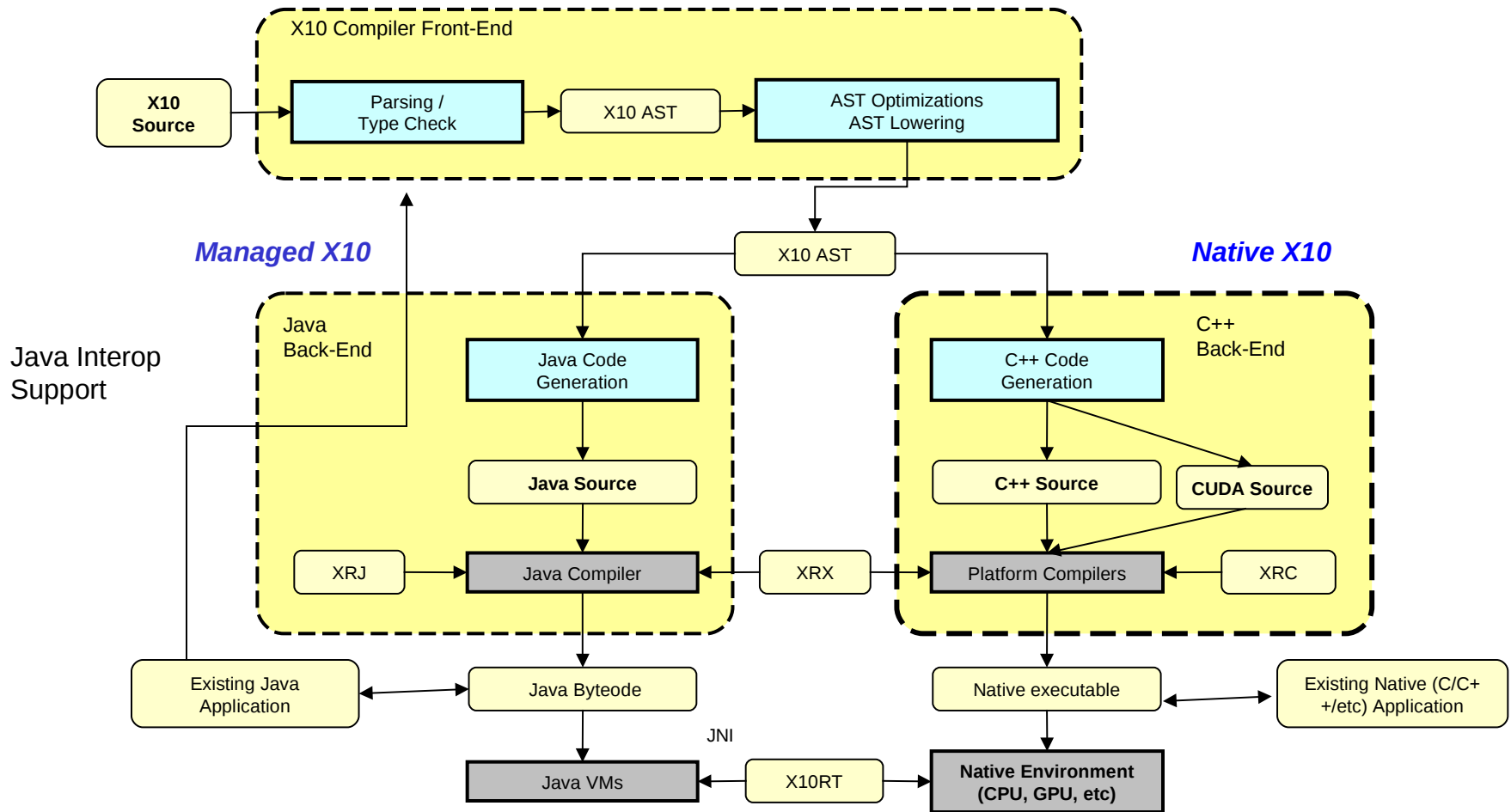
- Medium-scale commodity systems
 - ~100 nodes (~1000 core and ~1 terabyte main memory)
 - Goal: deliver main-memory performance with simple programming model (accessible to Java programmers)

- 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, GPUs (CUDA, language subset)
 - 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 or Java 7 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

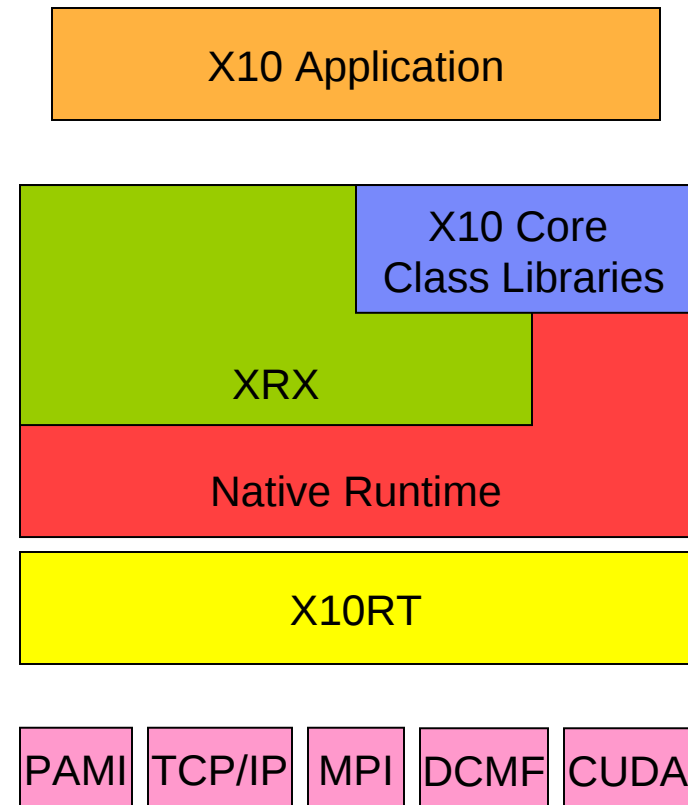


Java vs. C++ as Implementation Substrate

- Java
 - Just-in-time compilation (blessing & curse)
 - Sophisticated optimizations and runtime services for OO language features
 - Straying too far from Java semantics can be quite painful
 - Implementing a language runtime in vanilla Java is limiting
 - Object model trickery
 - Implementing Cilk-style workstealing
- C++
 - Ahead-of-time compilation (blessing & curse)
 - Minimal optimization of OO language features
 - Implementing language runtime layer
 - Ability to write low-level/unsafe code (flexibility)
 - Much fewer built-in services to leverage (blessing & curse)
- Dual path increases effort and constrains language design, but also widens applicability and creates interesting opportunities

X10 Runtime

- X10RT (X10 runtime transport)
 - active messages, collectives, RDMA
 - 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



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

- Basic dynamic optimization
 - Assume local, until first at executed
 - local aggregation and message batching (up to local quiescence)

XRX: Scalable 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
- Basic dynamic optimization
 - Assume local, until first at executed
 - local aggregation and message batching (up to local quiescence)
- Additional optimizations needed for scaling (@50k Places)
 - pattern-based specialization
 - local finish, SPMD finish, ping pong, single async
 - software routing
 - uncounted asyncs
 - runtime optimizations + static analysis + pragmas



good fit for APGAS

Scalable Communication

High-Performance Interconnects

- RDMAAs
 - efficient remote memory operations
 - fundamentally asynchronous
 - async semantics

 **good fit for APGAS**

```
Array.asyncCopy[Double](src, srcIndex, dst, dstIndex, size);
```

- Collectives
 - multi-point coordination and communication
 - all kinds of restrictions today

 **poor fit for APGAS today**

```
Team.WORLD.barrier(here.id);  
columnTeam.addReduce(columnRole, localMax, Team.MAX);
```

- bright future (MPI-3 and much more...)

 **good fit for APGAS**

Scalable Memory Management

- Garbage collector
 - problem 1: distributed heap
 - solution: segregate local/remote refs
 - GC for local refs; distributed GC experiment
 - problem 2: risk of overhead and jitter
 - solution: maximize memory reuse...

- Congruent memory allocator
 - problem: not all pages are created equal
 - large pages required to minimize TLB misses
 - registered pages required for RDMA
 - congruent addresses required for RDMA at scale
 - solution: dedicated memory allocator
 - configurable congruent registered memory region



not an issue in practice



issue is contained

Final Thoughts

- X10 Approach
 - Augment full-fledged modern language with core APGAS constructs
 - Problem selection: do a few key things well, defer many others
 - Enable programmer to evolve code from prototype to scalable solution
 - Mostly a pragmatic/conservative language design (except when its not...)

- X10 2.4 (today) is not the end of the story
 - A base language in which to build higher-level frameworks (arrays, ScaleGraph, M3R)
 - A target language for compilers (MatLab, DSLs)
 - APGAS runtime: X10 runtime as Java and C++ libraries
 - APGAS programming model in other languages

<http://x10-lang.org>

References

- Main X10 website
<http://x10-lang.org>
- “A Brief Introduction to X10 (for the HPC Programmer)”
<http://x10.sourceforge.net/documentation/intro/latest/html/>
- X10 2012 HPC challenge submission
<http://hpcchallenge.org>
<http://x10.sourceforge.net/documentation/hpcc/x10-hpcc2012-paper.pdf>
- Unbalanced Tree Search in X10 (PPoPP 2011)
<http://dl.acm.org/citation.cfm?id=1941582>
- M3R (VLDB 2012)
http://vldb.org/pvldb/vol5/p1736_avrahamshinnar_vldb2012.pdf