# X10 at Petascale

*Lessons learned from running X10 on the PERCS prototype*

Olivier Tardieu

*http://x10-lang.org*

*This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002.*

# Outline

- X10 Overview
  - APGAS Programming model
  - implementation overview
- Benchmarks
  - PERCS prototype
  - performance results
- X10 at Scale
  - scheduling for SMPs and distributed systems
  - high-performance interconnects
- Wrap-Up

# X10 Overview

# X10: Productivity and Performance at Scale

>8 years of R&D by IBM Research supported by DARPA (HPCS/PERCS)

Programming language

- Bring Java-like productivity to HPC
  - evolution of Java with input from Scala, ZPL, CCP, …
  - imperative OO language, garbage collected, type & memory safe
  - rich data types and type system
- Design for scale
  - multi-core, multi-processor, distributed, heterogeneous systems
  - few simple constructs for concurrency and distribution

Tool chain

- Open source compilers, runtime, IDE
- Debugger *(not open source)*

# Partitioned Global Address Space (PGAS) Languages

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

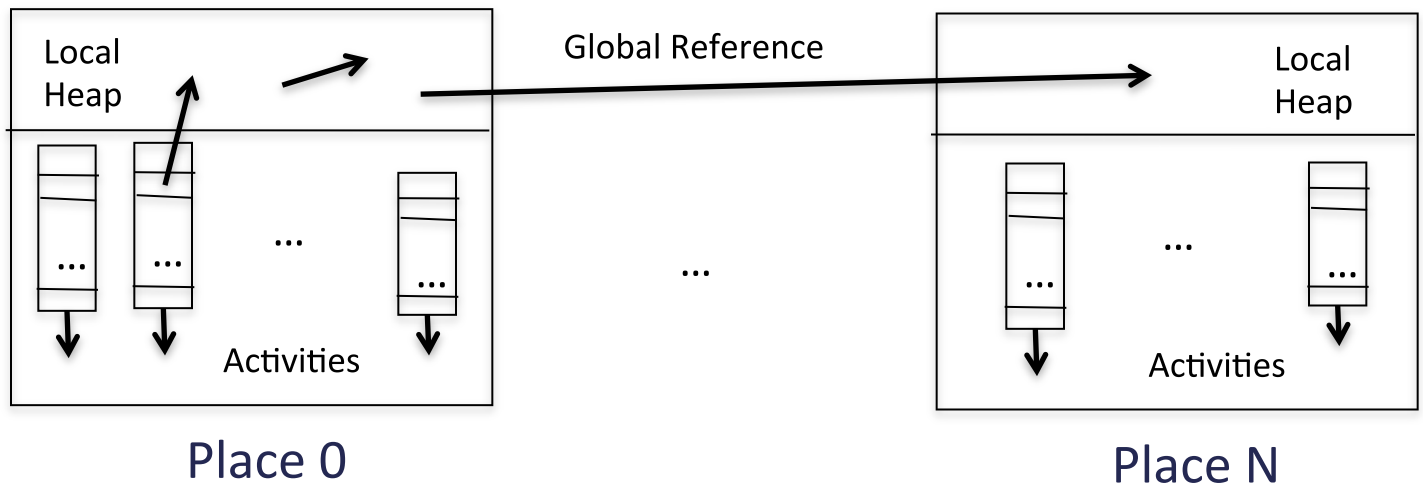
PGAS combines a single global address space with locality awareness

- 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 World

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

```
$ x10c++ HelloWorld.x10
$ X10_NPLACES=4 ./a.out hello
Place(1) says hello
Place(3) says hello
Place(2) says hello
Place(0) says hello
```

# 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

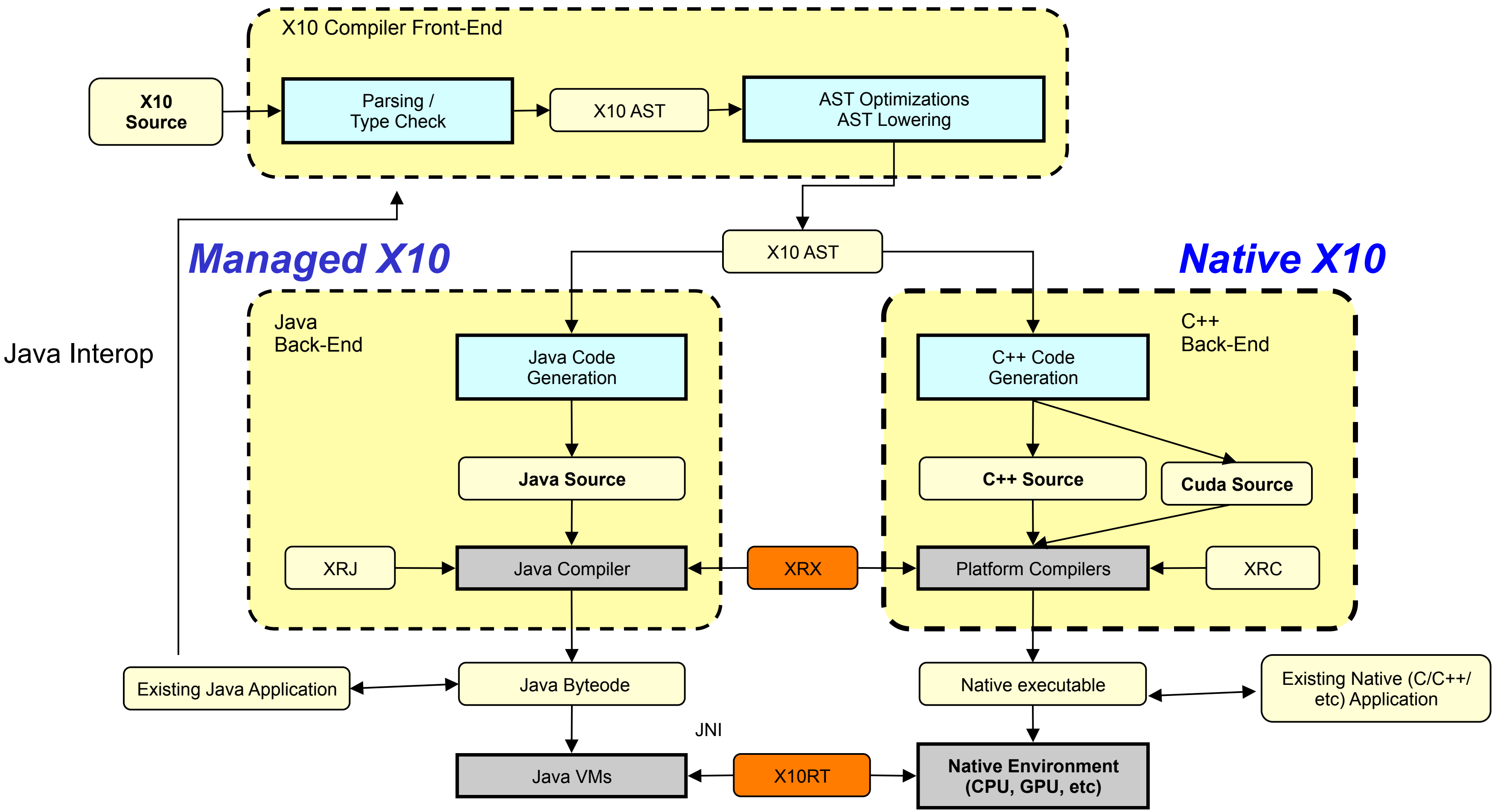
```
// swap row i local and j remote
val h = here;
val row_i = rows()(i);
finish at(p) async {
  val row_j = rows()(j);
  rows()(j) = row_i;
  at(h) async row()(i) = row_j;
}
```

# X10 Implementation Overview

# X10 Tool Chain

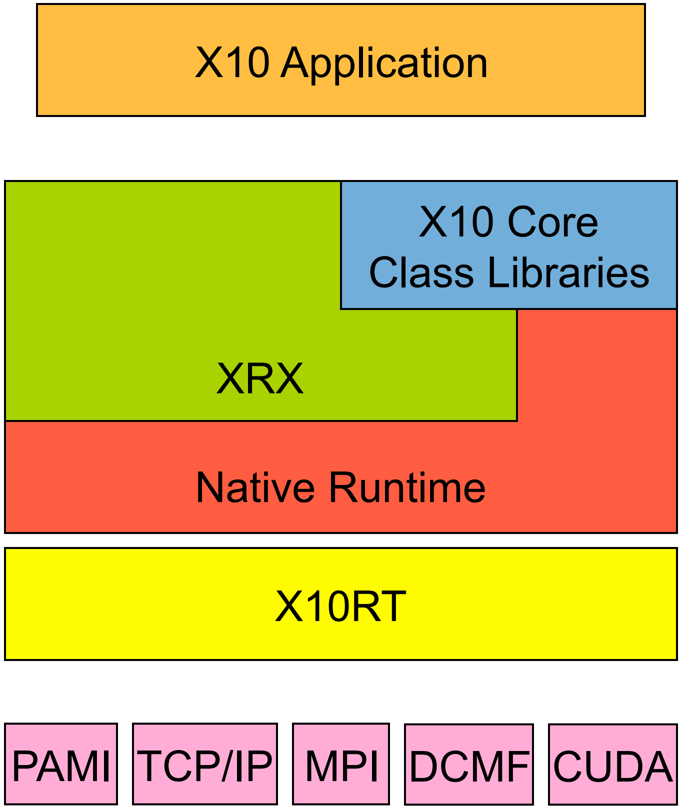
- X10 is an open source project (Eclipse Public License)
  - latest release (X10 2.3.1) available at http://x10-lang.org
  - active research/academic community; workshops, papers, courses, etc.
- X10 implementations
  - C++ based (“Native X10”)
    - multi-process (one place per process + GPU; multi-node)
    - x86, x86_64, Power; Linux, AIX, OS X, Cygwin, BG/P; TCP/IP, PAMI, DCMF, MPI; CUDA
  - JVM based (“Managed X10”)
    - multi-process (one place per JVM; multi-node) except on Windows (single place)
    - runs on any Java 6 or Java 7 JVM over TCP/IP
- X10DT (Eclipse-based X10 IDE) available for Windows, Linux, OS X
  - supports many core development tasks including remote build/execute facilities
  - IBM Parallel Debugger for X10 Programming *(not open source)*

# X10 Compilation and Execution



# 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



# Benchmarks

# Eight Kernels Running on the PERCS Prototype

- 4 HPC Challenge benchmarks

| | |
|---|---|
| Linpack | TOP500 (flops) |
| Stream | local memory bandwidth |
| Random Access | distributed memory bandwidth |
| Fast Fourier Transform | mix |

- Machine learning kernels

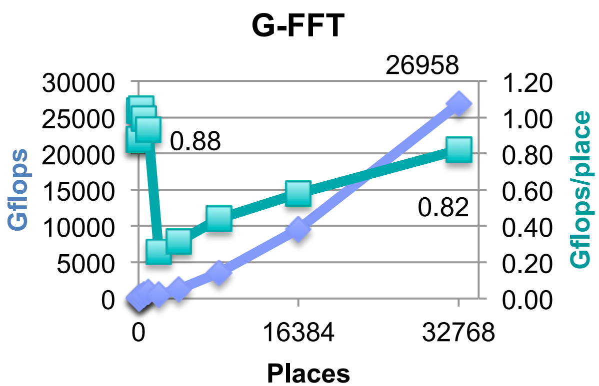
| | |
|---|---|
| SSCA1 | pattern matching |
| KMEANS | graph clustering |
| SSCA2 | irregular graph traversal |
| UTS | unbalanced tree traversal |

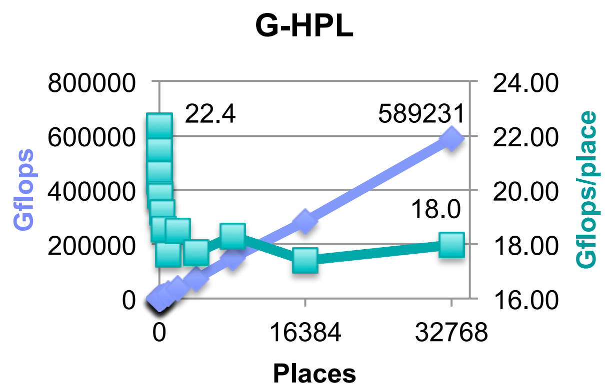
- At scale on the PERCS prototype (21 racks)
  - 55,680 Power7 cores (1.7 PFLOPS)

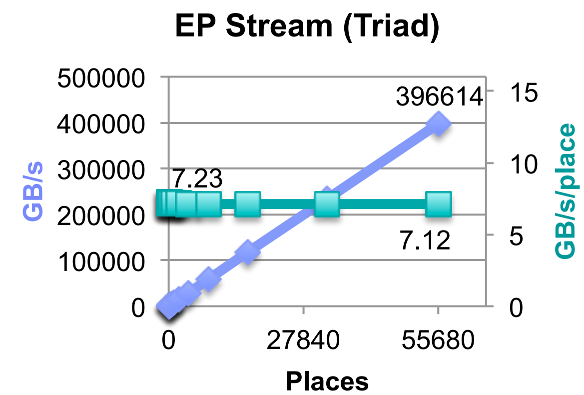
# Performance at Scale

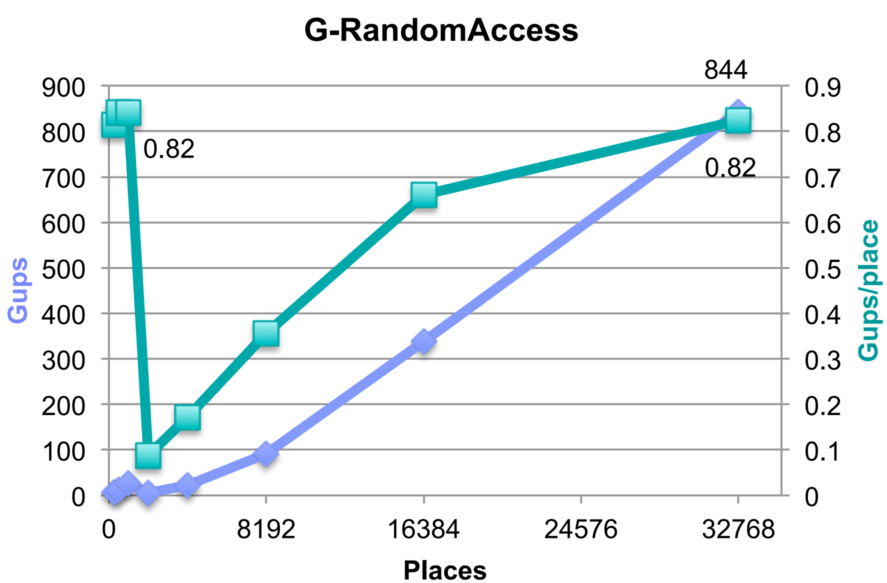
| | cores | absolute performance at scale | parallel efficiency (weak scaling) | performance relative to best implementation available |
|---|---|---|---|---|
| Stream | 55,680 | 397 TB/s | 98% | 85% (lack of prefetching) |
| FFT | 32,768 | 27 Tflops | 93% | 40% (no tuning of seq. code) |
| Linpack | 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% | *reference code does not scale*<br>*4x to 16x faster than UPC code* |

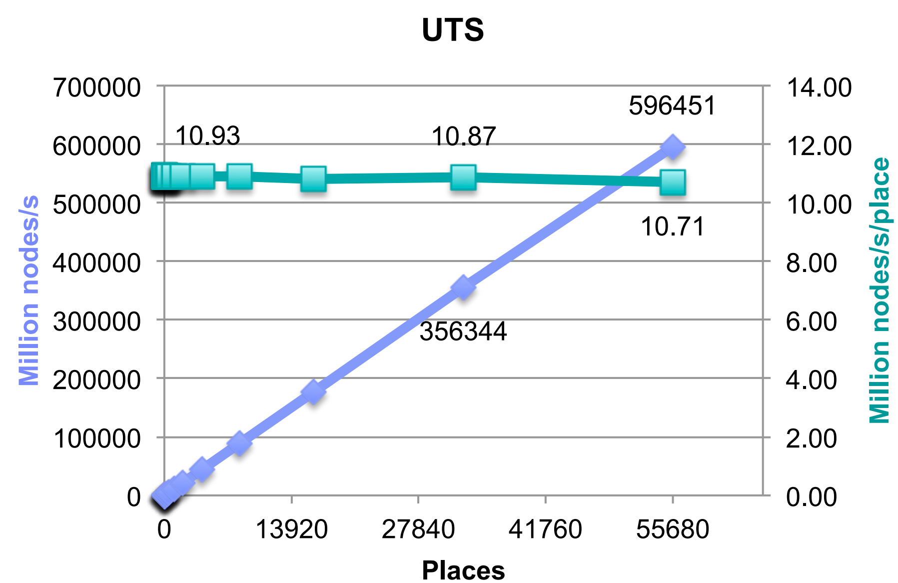
# HPCC Class 2 Competition: Best Performance Award



# X10 at Scale

# Challenges

- Scheduling
  - in each place: from many activities to few cores
  - across places: distributed load balancing
- Coordination
  - distributed termination detection
  - collective control-flow
- Communication
  - optimized point-to-point
  - collective data-flow
- Memory management
- And more…

# Scheduling for SMPs

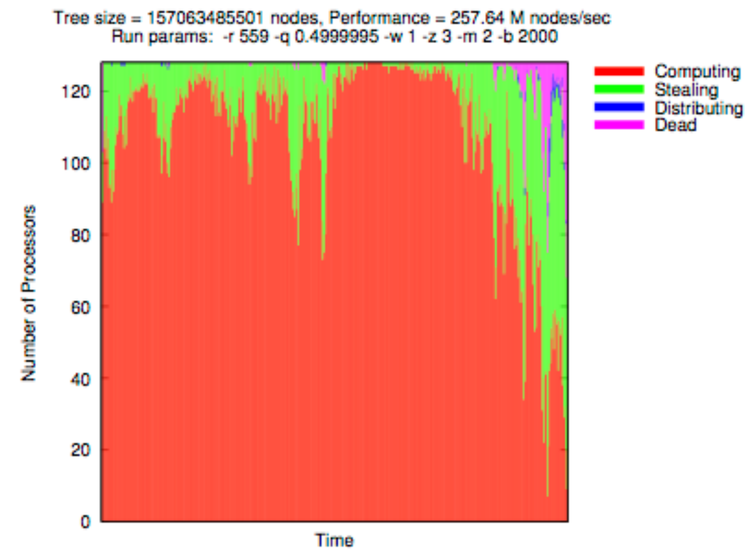
- Many more activities than execution units (hardware threads)
- Non-preemptive work-stealing schedulers
  - pool of worker threads, per-worker deque of pending jobs
  - worker first serves own deque then steals from other
- Production scheduler
  - job = async body
  - pure runtime scheduler
- Research scheduler [PPoPP'12,OOPSLA'12]
  - job = continuation
  - requires compiler hooks or JVM hooks
  - fixed-size thread pool

➡ **Cilk-like performance**

# Distributed Load Balancing: Unbalanced Tree Search

- Problem statement
  - count nodes in randomly generated tree
  - separable random number generator
  - cryptographic & highly unbalanced

- Key insights
  - lifeline-based global work stealing [PPoPP'11]
    - *n* random victims then *p* lifelines (hypercube)
  - compact work queue (for shallow trees)
    - thief steals half of each work item
  - finish only accounts for lifelines
  - sparse communication graph
    - bounded list of potential random victims
    - finish trades contention for latency




**genuine APGAS algorithm**

# Distributed Termination

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

# High-Performance Interconnects

- RDMAs
  - 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

# Memory Management

- Garbage collection
  - problem: risk of overhead and jitter
  - solution: mitigation techniques
    - maximize memory reuse
    - GC hints *(not always beneficial)*
    - X10 runtime structures are freed explicitly

➔ **not an issue in practice**

- Low-level constraints
  - problem: not all pages are created equal
    - large pages required to minimize TLB misses
    - registered pages required for RDMAs
    - congruent addresses required for RDMAs at scale
  - solution: congruent memory allocator
    - configurable congruent registered memory region
      - backed by large pages if available
      - only used for performance critical arrays

➔ **issue is contained**

# Adaptability

From 256 cores in January 2011 to 7,936 in March 2012 to 47,040 in July 2012
Delivery in August 2012

?

**good abstractions**

# Wrap-Up

# Future Developments

- Funding from US Dept. of Energy (X-Stack, part of D-TEC project -> 2015)
  - develop APGAS runtime based on X10 runtime to enable usage of APGAS programming model (finish, async, at, places) from C/C++/Fortran code
  - integrate X10 compiler front-end with ROSE compiler infrastructure
  - enhance X10 language support for Domain Specific Languages (DSL)
- Funding from US Air Force Research Lab (Resilient and Elastic X10 -> 2014)
  - add support for place failure and dynamic place creation to X10 runtime & language
- X10 for Big Data
  - enhance Managed X10 (X10 on JVMs) to support development of IBM middleware
- X10 for HPC
  - support porting of X10 to new systems (BlueGene/Q, K Computer, Tsubame)
  - enhance MPI backend and interoperability

# Selected Application Projects

IBM

- Main Memory Map Reduce (M3R)
  - map/Reduce engine in X10 optimized for in-memory workloads
- Global Matrix Library (open source)
  - matrix (sparse & dense) library supporting parallel execution on multiple places
- SAT-X10
  - X10 control program to join existing SAT solvers into parallel, distributed solver

Community

- ANUChem
  - computational chemistry library developed by Australia National University
- ScaleGraph
  - scalable graph library developed by Tokyo Institute of Technology
- XAXIS
  - large-scale agent simulation platform developed by Tokyo Institute of Technology

# Final Thoughts

Give X10 a try!

- Language definition is stable
- Tool chain is good enough, generated code is good

- Main X10 website
  http://x10-lang.org

- "A Brief Introduction to X10 (for the HPC Programmer)"
  http://x10.sourceforge.net/documentation/intro/intro-223.pdf

- X10 2012 HPC challenge submission
  http://hpcchallenge.org
  http://x10.sourceforge.net/documentation/hpcc/x10-hpcc2012-paper.pdf
  http://x10.sourceforge.net/documentation/hpcc/x10-hpcc2012-slides.pdf