X10 and APGAS at Petascale

Olivier Tardieu[1], Benjamin Herta[1], David Cunningham[2],
David Grove[1], Prabhanjan Kambadur[1], Vijay Saraswat[1],
Avraham Shinnar[1], Mikio Takeuchi[3], Mandana Vaziri[1]

[1]IBM T.J. Watson Research Center
[2]Google Inc.
[3]IBM Research – Tokyo

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

Background

- X10 tackles the challenge of programming at *scale*
  - HPC, cluster, cloud
  - scale out: run across many distributed nodes ➔ this talk & PPAA talk
  - scale up: exploit multi-core and accelerators ➔ CGO tutorial
  - resilience and elasticity ➔ next talk

- X10 is
  - a programming language
    - imperative object-oriented strongly-typed garbage-collected (like Java)
    - concurrent and distributed: Asynchronous Partitioned Global Address Space model
  - an open-source tool chain developed at IBM Research ➔ X10 2.4.2 just released
  - a growing community
    - X10 workshop at PLDI'14 ➔ CFP at http://x10-lang.org

- Double goal: *productivity* and *performance*

Outline

- X10
  - programming model: Asynchronous Partitioned Global Address Space

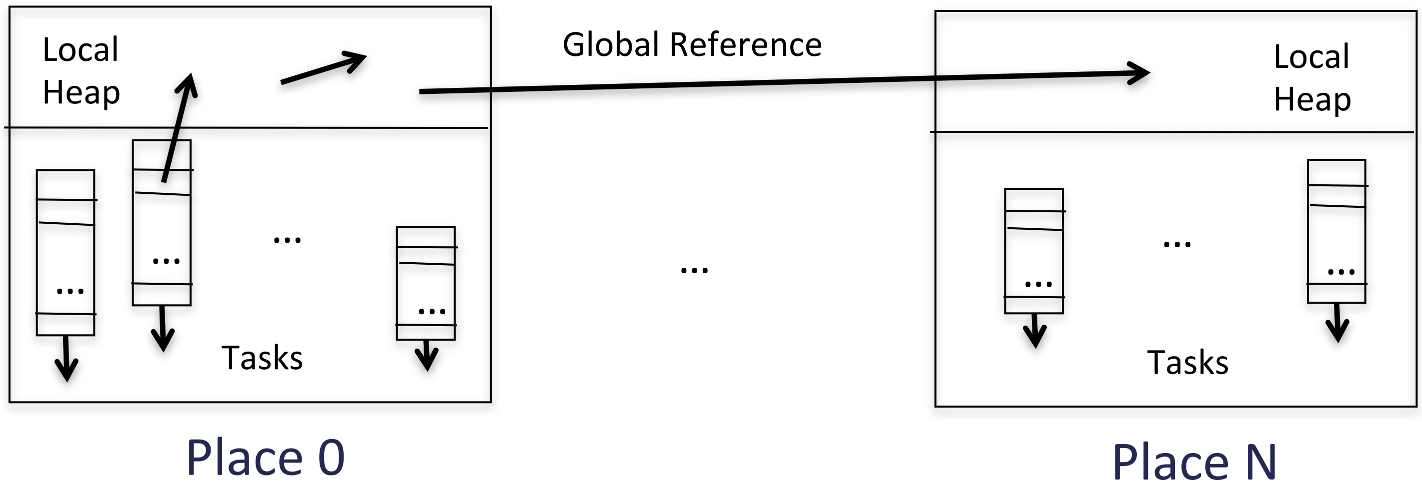
- Optimizations for scale out
  - distributed termination detection
  - high-performance networks
  - memory management

- Performance results
  - Power 775 architecture
  - benchmarks

- Global load balancing
  - Unbalanced Tree Search at scale

X10 Overview

APGAS 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]`

APGAS Idioms

- Remote procedure call

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

- Active message

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

- SPMD

```
finish for(p in Place.places()) {
  at(p) async {
    for(i in 1..n) {
      async doWork(i);
    }
  }
}
```

- Atomic remote update

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

- Divide-and-conquer parallelism

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

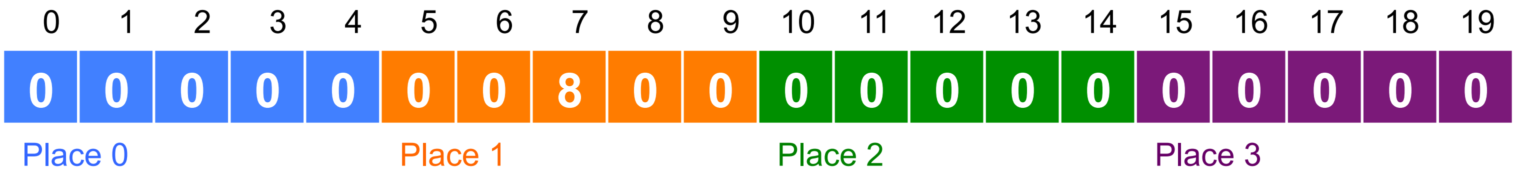
- *finish* construct is transitive and can cross place boundaries

Example: 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));
  }
}
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0

Optimizations for Scale Out

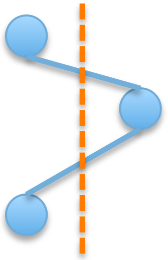
Distributed Termination Detection

- Local finish is easy
  - synchronized counter: increment when task is spawned, decrement when task ends
- Distributed finish is non-trivial
  - network can reorder increment and decrement messages
- X10 algorithm: disambiguation in space → space overhead
  - one row of *n* counters per place with *n* places
  - when place *p* spawns task at place *q* increment counter *q* at place *p*
  - when task terminates at place *p* decrement counter *p* at place *p*
  - finish triggered when sum of each column is zero
- Charm++ algorithm: disambiguation in time → communication overhead
  - successive non-overlapping waves of termination detections

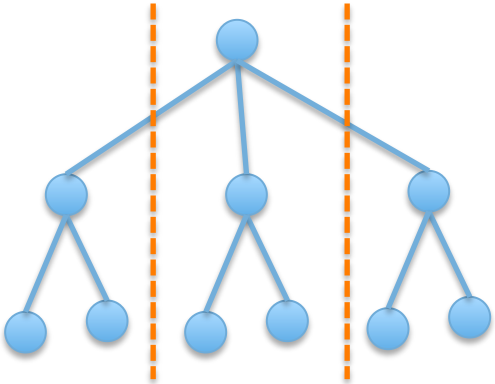
Optimized Distributed Termination Detection

- Source optimizations
  - aggregate messages at source
  - compress

- Software routing
  - aggregate messages at intermediate nodes



“get”



SPMD finish

- Pattern-based specialization
  - “put”: a finish governing a single task → wait for one ack
  - “get”: a finish governing round trip → wait for return task
  - local finish: a finish with no remote tasks → single counter
  - SPMD finish: a finish with no nested remote task → single counter
  - irregular/dense finish: a finish with lots of links → software routing

- Runtime optimizations + static analysis + pragmas → **scalable finish**

High-Performance Networks

- RDMAs
  - efficient remote memory operations
  - asynchronous semantics → **good fit for APGAS**
    - *just another task*

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

- Collectives
  - multi-point coordination and communication
  - networks/APIs biased towards SPMD today → **poor fit for APGAS today**

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

-
  - future: MPI-3 and beyond → **good fit for APGAS**
    - one-sided collectives, endpoints, etc.

Memory Management

- Garbage collector
  - problem: distributed heap
    - distributed garbage collection is impractical
  - solution: segregate local/remote refs → **issue is contained**
    - only local refs are automatically collected

- Congruent memory allocator
  - problem: low-level requirements
    - large pages required to minimize TLB misses
    - registered pages required for RDMAs
    - congruent addresses required for RDMAs at scale
  - solution: dedicated memory allocator → **issue is contained**
    - congruent registered pages
    - large pages if available
    - only used for performance-critical arrays
    - only impacts allocation & deallocation

Performance Results

DARPA HPCS/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

DARPA HPCS/PERCS Benchmarks

- HPC Challenge benchmarks

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

- Machine learning kernels

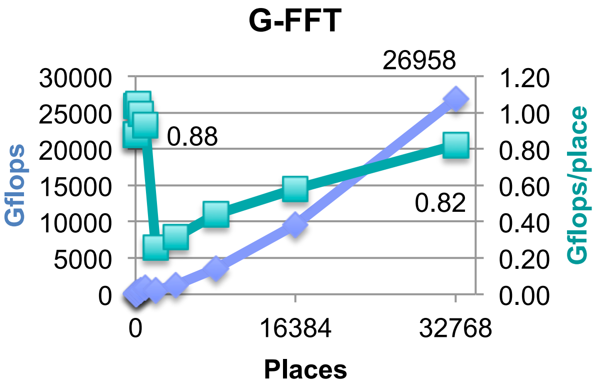
KMeans	graph clustering
SSCA1	pattern matching
SSCA2	irregular graph traversal
UTS	unbalanced tree traversal

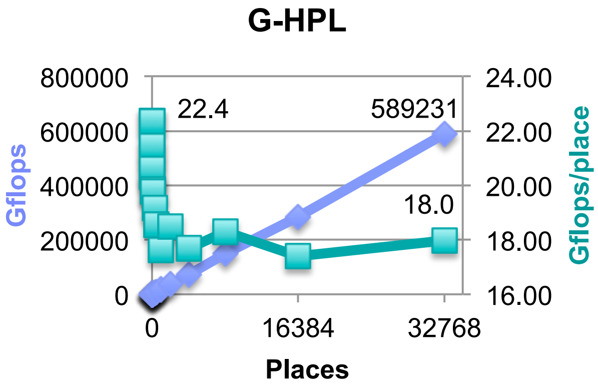
- *Implemented in X10 as pure scale out tests*
  - *one core = one place = one main task*
  - *native libraries for sequential math kernels: ESSL, FFTE, SHA1*

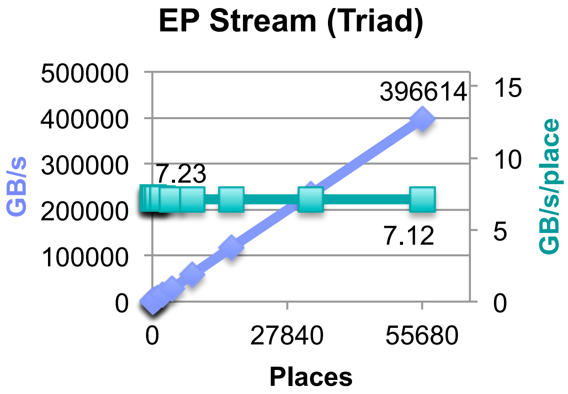
Performance at Scale (Weak Scaling)

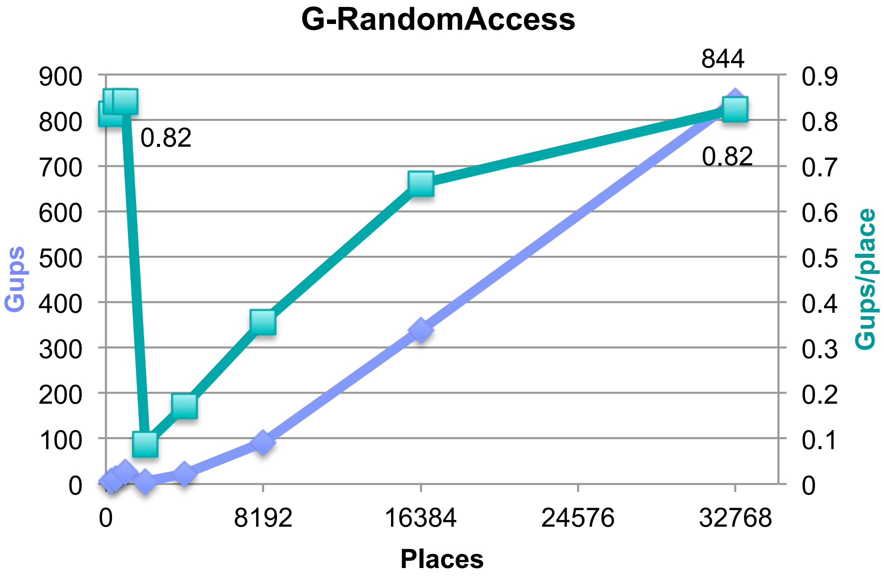
	number of cores at scale	absolute performance at scale	relative efficiency compared to single host (weak scaling)	performance at scale relative to best implementation available
Stream	55,680	397 TB/s	98%	87%
FFT	32,768	28.7 Tflop/s	100%	41% (no tuning)
Linpack	32,768	589 Tflop/s	87%	85%
RandomAccess	32,768	843 Gup/s	100%	81%
KMeans	47,040		98%	?
SSCA1	47,040		98%	?
SSCA2	47,040	245 B edges/s	*see paper for details*	?
UTS (geometric)	55,680	596 B nodes/s	98%	prior impl. do not scale

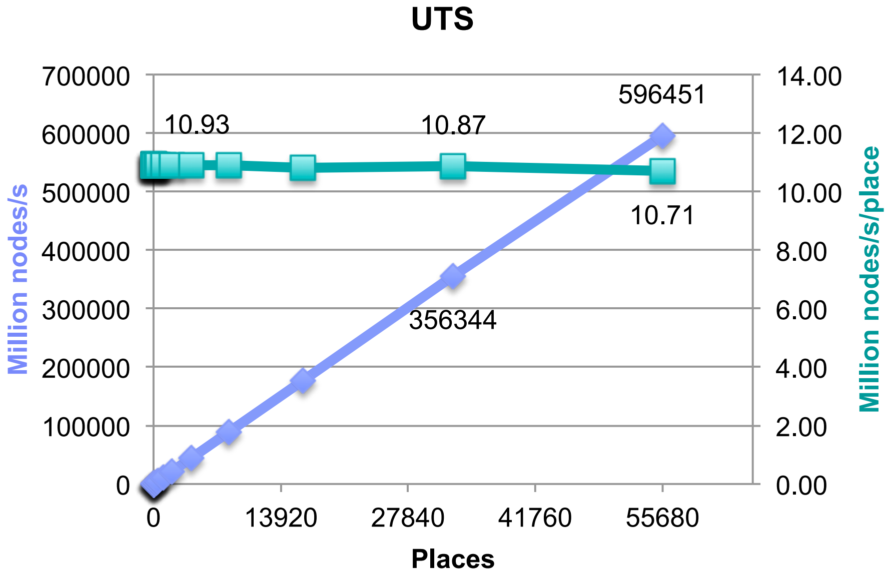
HPCC Class 2 Competition 2012: Best Performance Award











Global Load Balancing

Unbalanced Tree Search at Scale

- Problem
  - count nodes in randomly generated tree ➔ unbalanced & unpredictable
  - separable random number generator ➔ no locality constraint

- Lifeline-based global work stealing [PPoPP'11]
  - *n* random victims then *p* lifelines (hypercube)
  - steal (synchronous) then deal (asynchronous)

- Novel optimizations
  - use of nested finish scopes ➔ scalable finish
    - use of "dense" finish pattern for root finish
    - use of "get" finish pattern for random steal attempt
  - pseudo random steals ➔ software routing
  - compact work queue encoding (for shallow trees) ➔ less state, smaller messages
    - lazy expansion of intervals of nodes (siblings)

Conclusions

- Performance
  - X10 and APGAS can scale to Petaflop systems

- Productivity
  - X10 and APGAS can implement legacy algorithms
    - such as statically scheduled and distributed codes
  - X10 and APGAS can ease the development of novel scalable codes
    - including irregular and unbalanced workloads

- APGAS constructs deliver productivity and performance gains at scale

- Follow-up work presented PPAA’14
  - APGAS global load balancing library derived from UTS
  - application to SSCA2