X10 at Petascale

Olivier Tardieu, Benjamin Herta, David Cunningham, David Grove, Prabhanjan Kambadur, Vijay A. Saraswat, Avraham Shinnar, Mikio Takeuchi, Mandana Vaziri *@ IBM Research*

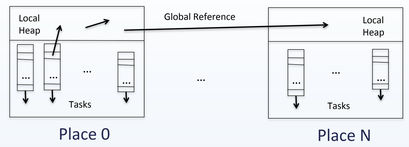
Overview

X10 is an evolution of Java for HPC and Big Data
- >8 years of R&D by IBM Research supported by DARPA (HPCS/PERCS)
- class-based imperative OO language, type safe and garbage collected
- *Asynchronous Partitioned Global Address Space (APGAS)* programming model
- open-source compilers (to C++ or Java), runtime, standard library, IDE

How to scale X10 and APGAS to petascale supercomputers and beyond?
- we add distributed termination detection optimizations (1)
- we add support for RDMAs and collectives (2)
- we add distributed load balancing (3)

We report performance results on a 1.7 PFLOPS Power 775 system
- 1740 nodes (32 Power7 cores 3.84 GHz, 128 GB DRAM), Torrent interconnect
- X10 2.2.3 implementations of 8 kernels: HPCC benchmarks, K-Means, Smith-Waterman, Betweenness Centrality, Unbalanced Tree Search

APGAS Programming Model: Places and Tasks



Task parallelism
- async S
- finish S

Concurrency control within a place
- when(c) S
- atomic S

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

Distributed heap
- GlobalRef[T]
- PlaceLocalHandle[T]

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 n;
 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;
```

Overlap of computation and communication
```
val seq = new Sequence(…);
while(seq.hasValue()) finish {
  val current = seq.getValue();
  at(ref) async ref() = current;
  seq.computeNextValue();
}
```

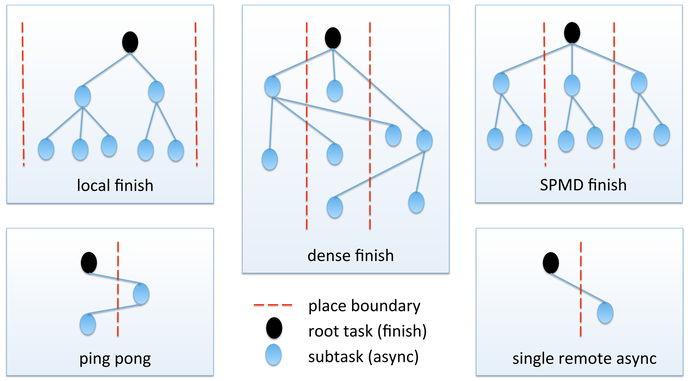
Performance at Scale – X10 Codes Compared to Optimized HPC Challenge Class 1 Codes

Benchmark	X10 performance with a single host	Hosts used at scale	X10 performance at scale (weak scaling)	Relative efficiency at scale (vs. 1 host)	Fraction of HPC class 1 optimized implementation performance (*)
Global HPL	20.62 Gflop/s/core	1024	17.98 Gflop/s/core	**87%**	**83%**
Global RandomAccess	0.82 Gup/s/host	1024	0.82 Gup/s/host	**100%**	**81%**
Global FFT	0.88 Gflop/s/core	1024	0.88 Gflop/s/core	**100%**	**41% (@)**
EP Stream (Triad)	7.23 GB/s/core	1740	7.12 GB/s/core	**98%**	**87%**
Unbalanced Tree Search	10.90 M nodes/s/core	1740	10.71 M nodes/s/core	**98%**	*not available*
K-Means	6.16s run time	1470	6.27s run time	**98%**	*not available*
Smith-Waterman	12.68s run time	1470	12.87s run time	**98%**	*not available*
Betweenness Centrality	11.59 M edges/s/core	1470	5.21 M edges/s/core	**45%**	*not available*

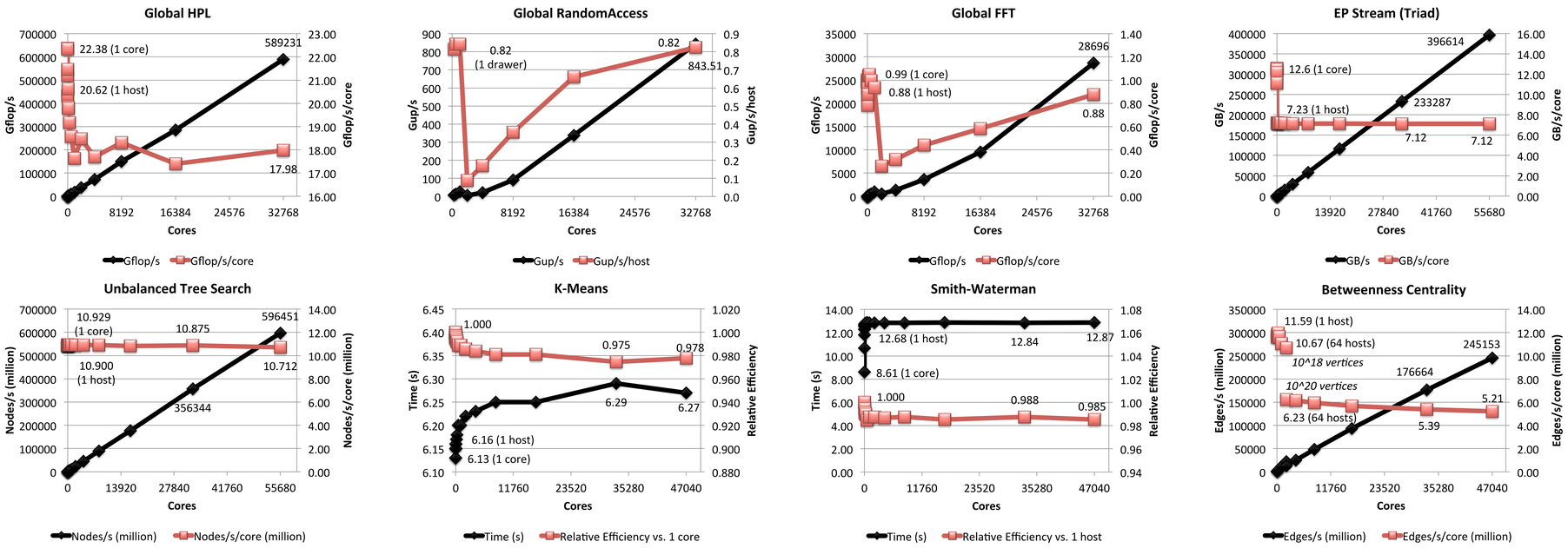
(*) as reported at http://icl.cs.utk.edu/hpcc/hpcc_record.cgi?id=495

(@) due to untuned sequential 1D FFT code

Finish Patterns with Optimized Implementations



Scaling Curves – X10 Codes – Weak Scaling – Aggregate Performance (black curve) – Performance Per Core (red curve)



(1) Scalable Distributed Termination Detection

Base algorithm is not scalable
- must handle arbitrary message reordering
- must handle arbitrary distributed task graphs

We add optimizations and specialized algorithms
- local aggregation and message batching (up to local quiescence)
- pattern-based specialization of the finish construct
  - local finish, SPMD finish, ping pong, single remote async, dense finish
- software routing (indirect routes)
- uncounted asyncs
- runtime optimizations + static analysis + pragmas

(2) Exploitation of Network Accelerators

We add copy-less RDMA-backed remote data transfer APIs
- fundamentally asynchronous (async semantics)
  - Array.asyncCopy[Double](src, srcIndex, dst, dstIndex, size);

We add teams for collective communications (~ MPI communicators)
- with restrictions today but bright future (MPI-3 and beyond)
  - Team.WORLD.barrier(here.id);
  - columnTeam.addReduce(columnRole, localMax, Team.MAX);

We add a congruent memory allocator
- enable symmetric allocation, pinning & registration, large pages

(3) Scalable Distributed Load Balancing

Unbalanced Tree Search Benchmark
- count nodes in randomly generated tree on the fly
- separable cryptographic random number generator
  - childCount = rand(nodeId); childId = SHA1(nodeId, childIndex)
- highly unbalanced, unpredictable trees
- tree traversal can be relocated (no data dependencies, no locality)

We add a dynamic distributed load balancing kernel
- built on top of lifeline-based work-stealing [Saraswat et al. PPoPP'11]
- new work queue, finish, and communication optimizations
- *scalability improved by an order of magnitude*

Take Away

X10 and APGAS scale to petaflop systems
- using the optimizations we identified and implemented

X10 and APGAS support traditional SPMD-style kernels well
- approaching the performance of optimized low-level implementations

X10 and APGAS enable novel algorithms to be developed
- taking advantage of task-based asynchronous concurrency
- to deliver performance at scale for irregular, unbalanced workloads

The X10 distribution and benchmark source codes are available at

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.