X10 for Productivity and Performance at Scale

Olivier Tardieu, David Grove, Bard Bloom, David Cunningham,
Benjamin Herta, Prabhanjan Kambadur, Vijay Saraswat,
Avraham Shinnar, Mikio Takeuchi, Mandana Vaziri

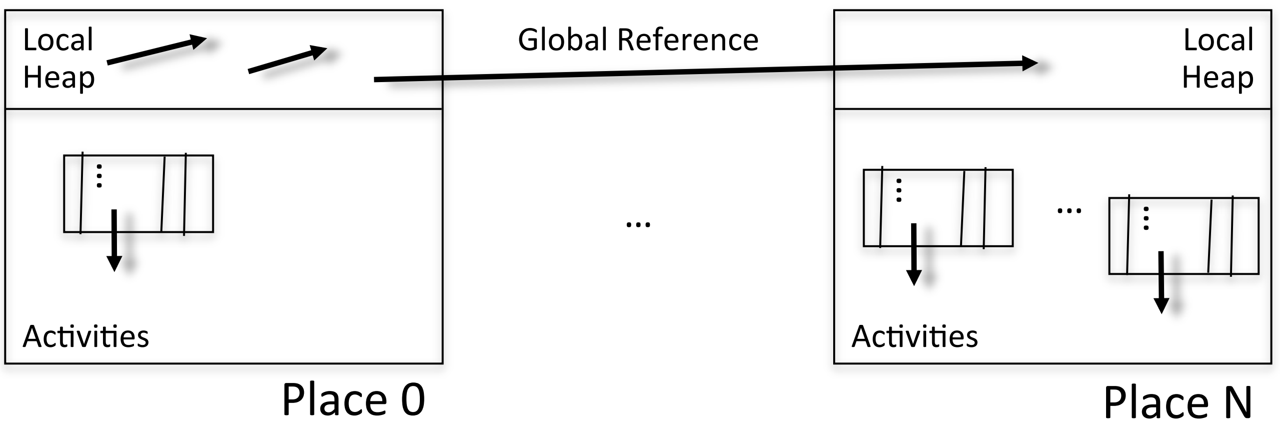
This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002.

X10

- Programming language
  - 8 years of R&D by IBM Research with support from DARPA/HPCS (PERCS)
  - modern object-oriented language
    - evolution of Java
    - strongly typed, memory safe (garbage collected), pre/postconditions, invariants
    - constructs for concurrency and distribution
- Focus on scale
  - HPC and Big Data
  - scalable productivity and performance
- Open source
  - specification and implementation: http://x10-lang.org
  - doc: “A Brief Introduction to X10 for the High Performance Programmer”
- Portable and interoperable
  - backend compilers: C++, Java, CUDA
  - transports: sockets (IP & IB), PAMI, MPI, DCMF
  - architectures: Power (Linux, AIX), x86 (Linux, OSX, Windows), BlueGene/P

Programming Model: APGAS
Asynchronous Partitioned Global Address Space

- Two basic ideas: places and asynchrony



Concurrency

- `async S`
- `finish S`

Distribution

- `at(P) S`
- `at(P) e`

Scalable Productivity and Performance

- Scalable APGAS
  - many activities
    - work-stealing scheduler
  - many places
    - finish specialization via static analysis, dynamic analysis & pragmas

- High-performance interconnects
  - RDMAs
    - /* finish */ Array.asyncCopy(srcArray, dstArray);
  - collectives
    - Team.WORLD.barrier(role);

X10 delivers high productivity and high performance at *peta* scale

PERCS Prototype (Power 775)

- Compute Node
  - 32 Power7 cores 3.84 GHz
  - 128 GB memory
  - peak: 982 Gflops
  - *Torrent* interconnect
- Drawer
  - 8 nodes
- Rack
  - 8 to 12 drawers
- Full System
  - up to 1740 compute nodes
    - from 1470 in initial submission
  - up to 55680 cores
  - up to 1.7 Petaflops

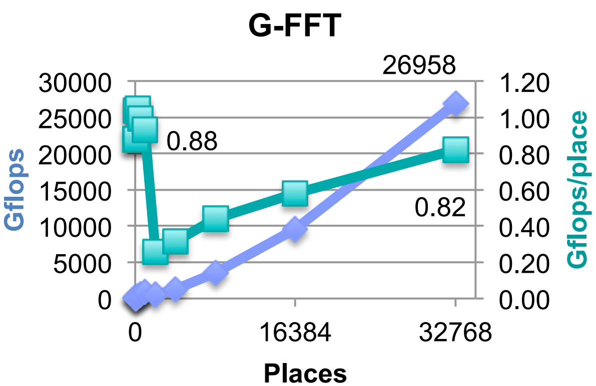
(1 Petaflops with 1024 compute nodes)

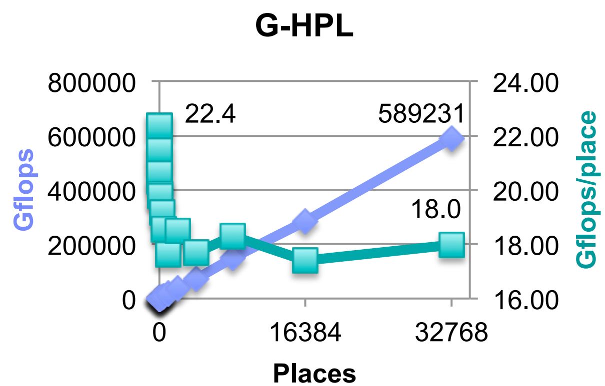
Performance at Scale

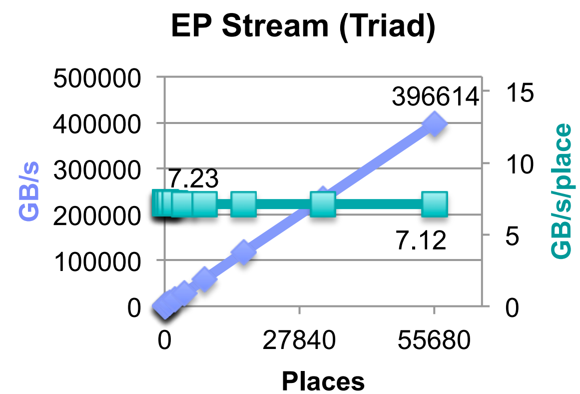
	cores	perf. at scale	parallel efficiency		relative perf.		X10 LOCs	C LOCs
STREAM	55,680	397 TB/s	98%	vs. 1 node	85%	vs. class 1	60	0
FFT	32,768	27/35 Tflops	93%	vs. 1 node	42%	vs. class 1	213	23+FFTE
HPL	32,768	589 Tflops	80%	vs. 1 core	80%	vs. class 1	588	120+BLAS
RA	32,768	844 Gups	100%	vs. 1 drawer	76%	vs. class 1	143	0
UTS	**55,680**	**596 B nodes/s**	**98%**	**vs. 1 core**	**>>**	**MPI, UPC**	**493**	**137+SHA1**

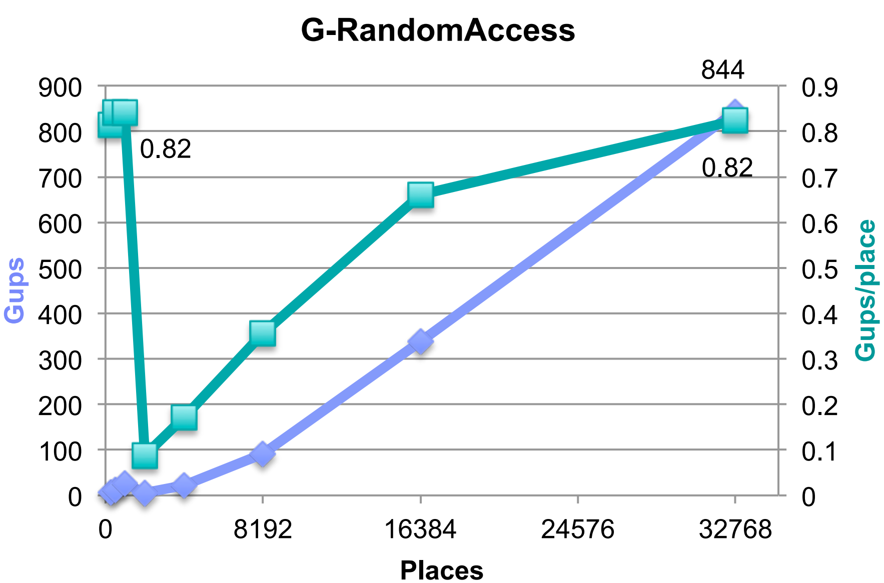
- FFT
  - 27 TFlops with 1024 hosts
  - 35 TFlops with 32,768 cores spread across 1740 hosts
  - FFTE: 1094 C LOCs
- UTS: Unbalanced Tree Search
  - law: fixed geometric
  - at scale: 69,312,400,211,958 nodes in 116s.
  - SHA1: 505 C LOCS implementation

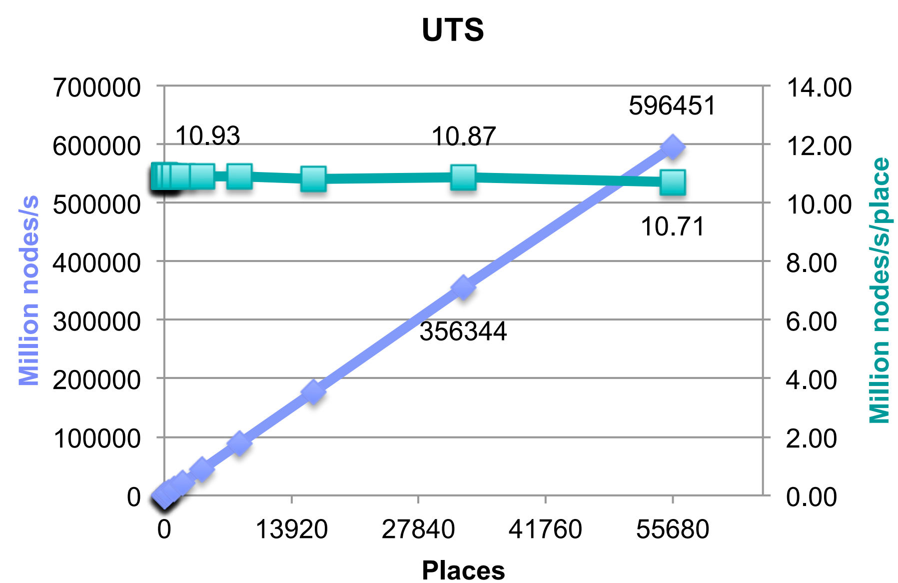
Performance Graphs











Unbalanced Tree Search

- Problem statement
  - count nodes in randomly generated tree
  - separable random number generator => work can be relocated
  - highly unbalanced => need for distributed load balancing

- Key insights
  - lifeline-based global work stealing [PPoPP'11]
    - n random victims then p lifelines (hypercube)
  - compact work queue (for wide but 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

X10 enables the rapid exploration of the design space leading to a scalable algorithm