X10: The Big Picture
http://x10.sf.net
x10-users@lists.sourceforge.net

Vijay Saraswat
vijay@saraswat.org
July 2007
IBM Research

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

- **X10 Core Team**
  - Rajkishore Barik, Ganesh Bikshandi, Chris Donawa, Allan Kielstra, Sreedhar Kodali, Nathaniel Nystrom, Igor Peshansky, Christoph von Praun, Vijay Saraswat, Vivek Sarkar, Lex Spoon, Pradeep Varma, Krishna Venkat, Tong Wen

- **X10 Tools**
  - Philippe Charles, Julian Dolby, Robert Fuhrer, Frank Tip, Mandana Vaziri

- **Emeritus**
  - Kemal Ebcioglu, Christian Grothoff, Vincent Cave

- **Research colleagues**
  - Ras Bodik, Guang Gao, Radha Jagadeesan, Jens Palsberg, Rodric Rabbah, Jan Vitek
  - Vinod Tipparaju, Jarek Nieplocha (PNNL)
  - Kathy Yellick, Dan Bonachea (Berkeley)
  - Doug Lea (SUNY Oswego)
  - Several others at IBM

- **Recent Publications**

- **Tutorials**
  - Graduate course on X10 at U Pisa (07/07)
A new era of mainstream parallel processing

**The Challenge**
Parallelism scaling replaces frequency scaling as foundation for increased performance ➔ Profound impact on future software

**Multi-core chips**  **Heterogeneous Parallelism**  **Cluster Parallelism**

Our response:
Use X10 as a new language for parallel hardware that builds on existing tools, compilers, runtimes, virtual machines and libraries
A strategy – radical incrementalism

- One (mainstream) programming model to rule them all.
  - Multicore, Clusters, Combos
  - Sequential Java + concurrency
  - places, async, finish, clock, atomic, annotations

- Keep it simple, stupid
  - Look to aggressively parallelize sequential programs
  - Concurrency as a means to an end

- Address all approaches
  - VM extensions
  - Libraries
  - Annotations + extensive static analysis
  - New languages
  - Domain-specific parallelism
Themes

- Target productivity, without sacrificing performance.
- Build on what we understand – core sequential OO base.
- Manage concurrency and distribution explicitly.
- Target the “mainstream” programmer; do not ignore the expert.
- Rule out large classes of errors by design – statically as far as possible, else dynamically.
- Keep the language small, orthogonal.
- Make easy things easy, hard things possible.
- Address the entire tool chain: development environment, analysis tools, debugging, …
Programming Model -- The Big Picture

- **Places**
  - Partition data and activities that operate on the data
  - But keep address space global.
  - *Locality central to heterogeneity, clustering.*

- **Asynchrony**
  - Express fine-grained parallelism
  - Exploit fork-join structure, recursive partitioning.

- **Atomicity**
  - Specify granularity of atomicity.
  - *Focus on the desired property, not the mechanism.*

- **Ordering**
  - Termination detection
  - Quiescence detection
X10 Programming Model – The Big Picture

- Dynamic parallelism with a *Partitioned Global Address Space*
- *Places* encapsulate binding of activities and globally addressable data
  - Number of places currently fixed at launch time
  - All concurrency is expressed as *asynchronous activities* – subsumes threads, structured parallelism, messaging, DMA transfers, etc.
- *Atomic blocks* enforce mutual exclusion of co-located data
  - No place-remote accesses permitted in atomic section
- *Immutable* data offers opportunity for single-assignment parallelism

**Storage classes:**
- Activity-local
- Place-local
- Partitioned global
- Immutable
X10 v1.01 Cheat sheet

Stm:
- \texttt{async} \ [ ( \texttt{Place} ) ] \ [ \texttt{clocked} \ \texttt{ClockList} ] \ \texttt{Stm}
- \texttt{atomic} \ \texttt{Stm}
- \texttt{when} \ ( \texttt{SimpleExpr} ) \ \texttt{Stm}
- \texttt{finish} \ \texttt{Stm}
- \texttt{next;} \ \texttt{c.resume()} \ \texttt{c.drop()}
- \texttt{for(} \ i : \texttt{Region} \ \texttt{)} \ \texttt{Stm}
- \texttt{foreach(} \ i : \texttt{Region} \ \texttt{)} \ \texttt{Stm}
- \texttt{ateach(} \ i : \texttt{Distribution} \ \texttt{)} \ \texttt{Stm}

Expr:
- \texttt{ArrayExpr}

ClassModifier : Kind
- \texttt{atomic}
- \texttt{nonblocking}
- \texttt{sequential}
- \texttt{local}
- \texttt{safe}

MethodModifier:
- \texttt{atomic}
- \texttt{nonblocking}
- \texttt{sequential}

Type:
- \texttt{DataType}
- \texttt{nullable}<\texttt{Type}>
- \texttt{future}<\texttt{Type}>

Kind:
- \texttt{value} \ | \ \texttt{reference}

ClassType, InterfaceType:
- \texttt{TypeName}

Annotations: \texttt{@} \texttt{InterfaceType}

Annotations suffix types and prefix almost all other syntactic elements.

\texttt{x10.lang} has the following classes (among others)
- point, range, region, distribution, clock, array

Some of these are supported by special syntax.

x10.lang has the following classes (among others)
- point, range, region, distribution, clock, array

Some of these are supported by special syntax.
X10 v1.01 Cheat sheet: Array support

ArrayExpr:
- new ArrayType ( Formal ) { Stm }
- Distribution Expr -- Lifting
- ArrayExpr [ Region ] -- Section
- ArrayExpr / Distribution -- Restriction
- ArrayExpr // ArrayExpr -- Union
- ArrayExpr.overlay(ArrayExpr) -- Update
- ArrayExpr. scan( [fun [, ArgList]]
- ArrayExpr. reduce( [fun [, ArgList]]
- ArrayExpr.lift( [fun [, ArgList]]

ArrayType:
- Type [Kind] ][
- Type [Kind] [ region(N) ]
- Type [Kind] [ Region ]
- Type [Kind] [ Distribution ]

Region:
- Expr : Expr -- 1-D region
- [ Range, ..., Range ] -- Multidimensional Region
- Region && Region -- Intersection
- Region || Region -- Union
- Region – Region -- Set difference
- BuiltinRegion

Dist:
- Region -> Place -- Constant distribution
- Distribution / Place -- Restriction
- Distribution / Region -- Restriction
- Distribution || Distribution -- Union
- Distribution – Distribution -- Set difference
- Distribution.overlay ( Distribution )
- BuiltinDistribution

Language supports type safety, memory safety, place safety, clock safety.
X10 availability

- **X10 is an open source project (Eclipse Public License).**
- **Website:** [http://x10.sf.net](http://x10.sf.net)
- **Reference implementation in Java, runs on any Java 5 VM.**
  - Windows/Intel, Linux/Intel
  - AIX/PPC, Linux/PPC
  - Runs on multiprocessors

- **Website contains**
  - Tutorial material
  - Presentations
  - Download instructions
  - Copies of some papers
  - Pointers to mailing list
Multi-core SMP Implementation

- **X10 Parser**
- **DOMO Static Analyzer**
- **Analysis passes**
- **Java code emitter**
- **Java compiler**

**Common components w/ SAFARI**

- **X10 Grammar**
- **Code Generation Templates**
- **Target Java**

**X10 Front End**

- **Place**
  - Ready Activities
  - Executing Activities
  - Blocked Activities
  - Clock
  - Future

**X10 Runtime**

- **JCU thread pool**
- **External Interface**
  - High Performance JRE
    - IBM J9 VM
    - Testarossa JIT Compiler
    - Modified for X10 on PPC/AIX
  - Portable Standard Java 5 Runtime Environment
    - (Runs on multiple Platforms)

**Java Concurrency Utilities (JCU)**

- **STM library**

**X10 classfiles**

- (Java classfiles with special annotations for X10 analysis info)

**Outbound activities**

- **Inbound activities**
- **Outbound replies**
- **Inbound replies**

**Atomic sections do not have blocking semantics**

**Activity can only access its stack, place-local mutable data, or global immutable data**
PERCS Programming Model, Tools

- **Productivity Measurements**
- **Refactoring for Concurrency**
- **Performance Exploration**
- **Parallel Tools Platform (PTP)**

**X10 Components**

**X10 runtime**

**Java Components**

**Fast extern interface**

**C/C++ Components**

**Fortran components**

**Eclipse platform**

**Dynamic Compilation + Continuous Program Optimization**

**Integrated Parallel Runtime: MPI + LAPI + RDMA + OpenMP + threads**
X10DT: Enhancing productivity

- Code editing
- Refactoring
- Code visualization

Vision: State-of-the-art IDE for a modern OO language for HPC
**X10 Compiler (06/2007)**

**Structure**
- Translator based on Polyglot (Java compiler framework)
- X10 extensions are modular.
- Uses Jikes parser generator.

**Code metrics**
- **Parser**: ~45/14K*
- **Translator**: ~112/9K
- **RTS**: ~190/10K – revised for JUC
- **Polyglot base**: ~517/80K
- **Approx 280 test cases.**

(*) classes+interfaces/LOC

**New features 6/07**
- Annotations
- X10lib v 1

**Operational X10 implementation (since 02/2005)**
X10Flash

- Distributed runtime
  - In C/C++
  - On top of messaging library (GASNet, ARMCI, LAPI)
  - Targeted for high-performance clusters of SMPs.

- X10lib
  - Runtime also made available as a standalone library.
  - Supporting global address space, places, asyncs, clocks, futures etc.

- Performance goal
  - To be competitive with MPI

- Release schedule
  - Internal demonstration 12/07
  - External release 2008
Conclusion

- X10 is intended to span multicore, heterogeneous systems and clusters.

- X10 offers a simple programming model for concurrency and distribution
  - Places
  - Asynchrony
  - Atomicity
  - Ordering

- X10 is being developed as an open source project
  - Reference implementation available on http://x10.sf.net

- A clustered implementation is being worked on
  - Compiles to C/C++
  - Uses LAPI for inter-process messaging
  - Uses algorithmic scheduling for intra-process scheduling
Backup
Comparison with MPI

Model of parallelism
- MPI: data parallel, SPMD
- X10: structured multithreading, SPMD is a special case.

Synchronization model
- MPI: clear distinction between control synchronization (barriers) and communication (or combine them in a collective).
  - synchronous: send / receive
  - asynchronous: put/get
- X10: shared memory paradigm
  - address space partitioned,
    - locality of access visible to the programmer
  - remote access are asynchronous (but may be made synchronous)
  - Supports “active messages”.
Comparison with UPC

X10 is similar to UPC in ...
- shared partitioned global address space
- barrier synchronization:
  - UPC split barrier (upc_notify, upc_wait)
  - X10 clocks: resume(), next
- parallel loops with affinity
  - UPC affinity clause in upc forall
  - X10 distribution in ateach

X10 extends UPC in ...
- dynamic structured multithreading (not SPMD)
- safety properties (managed runtime)
- distributed computation and data, concept of places
- array language, multidimensional arrays
- critical section synchronization
  - simple atomic blocks, not locks
  - conditional atomic blocks
- type system exposes access locality
- multiple, custom distributions
Comparison with CILK

**X10 is similar to CILK in ...**
- structured concurrency
  - Cilk: spawn, sync
  - X10: async, finish (block scoped, rooted exception model), method body not forced to finish spawned asyncs.
- serial elision only for non-blocking activities in X10

**X10 extends CILK in ...**
- distributed computation and data, concept of places
  - intra/inter-place work-stealing scheduler?
- array language, multidimensional arrays
- critical sections
  - simple atomic blocks, not locks
  - X10 has condition synchronization
- X10 managed runtime and type system: safety properties
Comparison with Java™

X10 language builds on the Java language

*Shared underlying philosophy: shared syntactic and semantic tradition, simple, small, easy to use, efficiently implementable, machine independent*

X10 does not have:
- Dynamic class loading
- Java’s concurrency features
  - thread library, volatile, synchronized, wait, notify

X10 restricts:
- Class variables and static initialization

X10 adds to Java:
- **value types, nullable**
- **Array language**
  - Multi-dimensional arrays, aggregate operations
- **New concurrency features**
  - activities (async, future), atomic blocks, clocks
- **Distribution**
  - places
  - distributed arrays
Overview of green field work

- **Compilation for Combos**
  - Cell + Opteron
  - Graphics cards – NVidia (CUDA)

- **Implement specific annotations and supporting static analyses**
  - Integrate with WALA ([http://wala.sf.net](http://wala.sf.net))

- **Support Implicit Parallelism**
  - tryasync, dependency annotations, flow annotations, ...
  - Static/dynamic support

- **Runtime**
  - x10lib – an implementation of the X10 runtime in C/C++ for high-performance clusters
  - Algorithmic scheduling
    - Design extensions to handle X10 control constructs.
    - Evaluate different scheduling strategies (work-stealing, parallel depth-first scheduling, ...)
    - Implement
Algorithmic Scheduling

- Centralized task-queue is a bottleneck
  - Each worker must synchronize in order to get new work
- Thread pool may grow unboundedly
  - when causes thread to suspend.
- Observation: finish/async programs have series/parallel dependence graphs.

- Solution: use Cilk-style work-stealing scheduler
  - Guaranteed $O(T_1/p + T_\infty)$ runtime, and $p*S_1$ space.
- Todo:
  - Investigate alternate schedulers with better space utilization/cache behavior (cf parallel depth-first scheduler)
  - Consider hierarchical, multi-place scheduler
  - Handle all X10 control constructs.

Preliminary results: very encouraging. Good scaling, absolute performance.
Speedup for spanning tree on a 32-core Niagara

Runtime ~ 0.5s for p=24, V=5M, E=20M
Speedup for spanning tree on 8-proc Opteron/Solaris

Runtime ~ 1.19s for p=8, V=5M, E=20M