X10: a High-Productivity Approach to High Performance Programming

www.research.ibm.com/x10

Vivek Sarkar
vsarkar@us.ibm.com
Senior Manager, Programming Technologies
IBM T.J. Watson Research Center

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCH30390004.
Acknowledgments

Recent Publications


Upcoming events

- OOPSLA 2006 tutorial, SC 2006 demo
Future System Trends: a new Era of Mainstream & HPC Parallel Processing

Hardware building blocks for mainstream and high-performance systems are varied and proliferating …

Homogeneous Multi-core

Heterogeneous Accelerators

Clusters

Challenge: Develop new language, compiler and tools technologies to support portable parallel abstractions for current and future hardware.
PERCS Programming Model, Tools and Compilers
(Productive Easy-to-use Reliable Computer System)

Eclipse platform

Java™ source code (w/ threads & conc util)
X10 source code
C/C++ source code (w/ MPI, OpenMP, UPC)
Fortran source code (w/ MPI, OpenMP)

Productivity Measurements
Rational PurifyPlus
Rational Team Platform
Remote System Explorer
Java Development Toolkit
X10 Development Toolkit
C/C++ Development Toolkit + MPI & OpenMP extensions
Fortran Development Toolkit

Java Components
X10 components
Fast extern interface
C/C++ components
Fortran components

Java runtime
X10 runtime
C/C++ runtime
Fortran runtime

HPC Toolkit + pSigma + Performance Tuning Automation
Dynamic Compilation + Continuous Program Optimization
Integrated Parallel Runtime: MPI + LAPI + RDMA + OpenMP + threads

Text in blue identifies PERCS contributions

Productive Easy-to-use Reliable Computer System
X10 Productivity Goals
(or why are MPI, OpenMP, UPC, … not good enough?)

- **Core Sequential Language**
  - Modern OO features -- strong typing, exceptions
  - Memory safety -- null check, bounds check, pointer safety
  - Single-assignment data -- value arrays, value types
  - Invariants -- dependent types, method annotations
  - Interoperability -- fast extern interface

- **Concurrency**
  - Beyond SPMD -- uniform model for task parallelism, data parallelism, messaging, data transfers
  - Global view -- one-sided accesses, with more general active messages / function shipping
  - High-level locality model -- places
  - Deadlock-free activity coordination -- finish, atomic, clocks
  - Datarace-free storage classes
    - Activity-local
    - Immutable (final, value types)
    - Clocked final
X10 Productivity Goals (contd.)

- **Loops and Arrays**
  - Region = set of multidimensional indices (points)
  - Distribution = map from region to places
  - Rich algebra on regions and distributions
  - for, foreach, ateach loops
  - Rank-independent programming of array computations

- **Analyzeability by compilers and tools**
  - Concurrency structure evident in program structure
  - Immutability of regions and distributions

- **Performance transparency**
  - Allows programmer-directed mapping of computation and communication
  - Use of dependent types, method annotations, and dynamic casts to provide additional invariants for compiler
X10 Programming Model

Storage classes:
- Activity-local
- Place-local
- Partitioned global
- Immutable

• 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 sections* enforce mutual exclusion of co-located data
  - No place-remote accesses permitted in atomic section
• *Immutable* data offers opportunity for single-assignment parallelism
X10 Language

- **async [(Place)] [clocked(c…)] Stm**
  - Run Stm asynchronously at Place
- **finish Stm**
  - Execute s, wait for all asyncs to terminate (generalizes join)
- **foreach (point P : Reg) Stm**
  - Run Stm asynchronously for each point in region
- **ateach (point P : Dist) Stm**
  - Run Stm asynchronously for each point in dist, in its place.
- **atomic Stm**
  - Execute Stm atomically
- **new T**
  - Allocate object at this place (here)
- **new T[d] / new T value [d]**
  - Array of base type T and distribution d
- **Region**
  - Collection of index points, e.g.
    - region r = [1:N,1:M];
- **Distribution**
  - Mapping from region to places, e.g.
    - dist d = block(r);
- **next**
  - suspend till all clocks that the current activity is registered with can advance
  - Clocks are a generalization of barriers and MPI communicators
- **future [(Place)] [clocked(c…)] Expr**
  - Compute Expr asynchronously at Place
- **F. force()**
  - Block until future F has been computed
- **extern**
  - Lightweight interface to native code

Deadlock safety: any X10 program written with above constructs (excluding future) can never deadlock
- Can be extended to restricted cases of using future
- Can be extended to restricted cases of using future
X10 Arrays, Regions, Distributions

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.
Example 1: All Reduce in Matrix-Vector Multiplication (CG)

```java
final int logN = … ; final int N = 2^logN;
final region R = [0:N];
final dist D = dist.factory.block(R);
final double [] A=new double [D] (point[i]) {return …;};
f final double [] B=new double [D];

d for (int j=0 ; j<phases ; j++ ) {
  shift=Factor/2;
  final int destId = (i+shift) % Factor + (i/Factor)*Factor;
  else A[i] = B[i] + B[destId];
  next;
  Factor = Factor / 2;  red = ! red;
}
if (!red) A[i]=B[i];
```
Example 2: Local Sequence Alignment

Goal: find the best matching subregions in a pair of sequences (e.g., DNA, RNA, sequence) so as to narrow down set of candidates for identifying biological relationships

\[
e[i, j] = \min (e[i-1,j] + \text{iGapPen}, \ e[i,j-1] + \text{iGapPen}, \ e[i-1,j-1] + (c1[i] == c2[j] \ ? \ \text{iMatch} : \text{iMisMatch}));
\]
void computeMatrix(int[] A, value char[] c1, value char[] c2,
       int firstCol, int lastCol) {

// Dynamic programming algorithm
    for ( point[i,j] : [1:N,firstCol:lastCol] )
                      A[i-1,j-1] + (c1[i]==c2[j] ? Match : MisMatch));
}

// Main program
const int N = c1.length, M = c2.length;

    A = new int[[0:N,0:M]];
    computeMatrix(A, c1, c2, 1, M);

    ...
Parallelization Algorithm

Each place redundantly computes columns overlapStart..myLow-1 in warmup array as input for computing columns myLow..myHigh
Distributed Parallel Version in X10

// Allocate A with a [* , block] distribution
int[.] A = new int[dist.blockColumns([0:N,0:M])];
final int overlap = ceilFrac(N*(-Match),Gap) + N;
// SPMD computation at each place
ateach(point [i] : dist.unique()) {
    final dist myD = A.distribution | here; // sub-distribution for this place
    final int myLow = myD.region.rank(1).low();
    final int myHigh = myD.region.rank(1).high();
    final int overlapStart = max(0,myLow-overlap);
    final dist warmupD = [0:N,overlapStart:myLow]->here;
    final int [.] W = new int[warmupD]; // W = local warmup array
    computeMatrix(W, c1, c2, overlapStart+1, myLow);
    foreach (point[i]:[0:N]) A[i,myLow] = W[i,myLow]; // Copy col myLow
    // Compute my section of global array A
    computeMatrix(A, c1, c2, myLow+1, myHigh);
}
void refine(final int n, final int l, final int nmax) {
    left = new Tree(this, 2.0*l);
    right = new Tree(this, 2.0*l+1);
    final nullable Tree ll = left, rr=right;
    if (n < (nmax-1)) {
        async { ll.refine(n+1, 2*l, nmax); }
        async { rr.refine(n+1, 2*l+1, nmax); }
    }
    if (n < nmax) data = null;
}
X10 Deployment

X10 language defines mapping from X10 objects & activities to X10 places

X10 deployment defines mapping from virtual X10 places to physical nodes

• Though the notion of places may make you think of X10 deployments on clusters, other granularities of deployment are possible as well …
X10 Deployment on a SMP with Multi-Core chips

- Basic Approach -- partition X10 heap into multiple place-local heaps
- Each X10 object is allocated in a designated place
- Each X10 activity is created at a designated place
- Use shared memory to support inter-place remote accesses
- Places serve as affinity hints for intra-SMP locality
**Possible X10 Deployment for Cell**

- **Basic Approach:**
  - Map place 0 on to GPP
  - Map place 1 on to PPE
  - Map places 2 to 9 on SPEs
  - Use finish & async’s as portable high-level constructs for DMAs

- **Challenges:**
  - Weak PPE
  - 128-bit alignment is critical
  - Limited memory on SPE's
  - Limited performance of code with frequent conditional or indirect branches
  - Different ISA's for PPE and SPE.
Current X10 Reference Implementation (Multi-Core Deployment)

**X10 Front End**

- **X10 source**
- **X10 Grammar** → **AST**
- **DOMO Static Analyzer** → **Annotated AST**
- **Code Generation Templates** → **Target Java**
- **Java code emitter** → **Java compiler**

**Common components w/ SAFARI**

- **X10 Parser** → **Analysis passes**

**X10 Runtime**

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

- **Place 0**
- **Place 1**

- **JCU thread pool**
- **X10 libraries**
- **STM library**
- **Java Concurrency Utilities (JCU)**

**Outputs:** 1) Program output, 2) Abstract Performance Metrics
X10 VM: the Big Picture

Java  X10  HPLS

J9 Execution Engine

Testa Rossa JIT

machine code

UPC  C, C++  FTN

library  W-Code  PDF

Machine Independent Optimizer

DLL / a.out
Preliminary Speedup Results:
JGF Section 2 benchmarks (Size C) on 16-way Power5+ SMP
extern: inter-operability with other languages

class Daxpy {

static { System.loadLibrary("blas"); }
extern static void daxpy(int n, double da, double[] dx,
                          int incx, double[] dy, int incy);

declaration

public static void main(String args[]) {
    final int N = 10;
    double da = …;
    double[] dx = double [N];
    double[] dy = double [N];
    int incx = 1, incy = 1;

    for (int i = 0; i < N; i++) {
        dx[i] = …;
        dy[i] = …;
    }

call
daxpy (n, da, dx, incx, dy, incy);
}
## Relating optimizations for past programming paradigms to X10 optimizations

<table>
<thead>
<tr>
<th>Programming paradigm</th>
<th>Activities</th>
<th>Storage classes</th>
<th>Important optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message-passing e.g., MPI</td>
<td>Single activity per place</td>
<td>Place local</td>
<td>Message aggregation, optimization of barriers &amp; reductions</td>
</tr>
<tr>
<td>Data parallel e.g., HPF</td>
<td>Single global program</td>
<td>Partitioned global</td>
<td>SPMDization, synchronization &amp; communication optimizations</td>
</tr>
<tr>
<td>PGAS e.g., Titanium, UPC</td>
<td>Single activity per place</td>
<td>Partitioned global, place local</td>
<td>Localization, SPMDization, synchronization &amp; communication optimizations</td>
</tr>
<tr>
<td>DSM e.g., TreadMarks</td>
<td>Multiple</td>
<td>Partitioned global, activity local</td>
<td>Data layout optimizations, page locality optimizations</td>
</tr>
<tr>
<td>NUMA</td>
<td>Single activity per place</td>
<td>Partitioned global, activity local</td>
<td>Data distribution, synchronization &amp; communication optimizations</td>
</tr>
<tr>
<td>Co-processor e.g., STI Cell</td>
<td>Single activity per place</td>
<td>Partitioned-global, place-local</td>
<td>Data communication, consistency, &amp; synchronization optimizations</td>
</tr>
<tr>
<td>Futures / active messages</td>
<td>Multiple</td>
<td>Place-local, activity local</td>
<td>Message aggregation, synchronization optimization</td>
</tr>
<tr>
<td>Full X10</td>
<td>Multiple activities in multiple places</td>
<td>Partitioned-global, place-local, activity-local</td>
<td>All of the above</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- X10 programming model provides core concurrency and distribution constructs for new era of mainstream parallel processing
- X10 language is our preferred embodiment of the X10 programming model
  - Other embodiments are possible
    - C/C++ libraries
    - Integration in C, C++, or Fortran
    - Domain-specific application frameworks
- Two primary opportunities for adoption of X10 ideas:
  - DARPA High Productivity Programming Language Systems
  - Mainstream language for multi-core
- We’d welcome collaboration on X10