X10: New opportunities for Compiler-Driven Performance via a new Programming Model

Kemal Ebcioglu
Vijay Saraswat
Vivek Sarkar
IBM T.J. Watson Research Center
{kemal,vsaraswat,vsarkar}@us.ibm.com

Compiler-Driven Performance Workshop --- CASCON 2004

Oct 6, 2004

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

• IBM PERCS Team members
  - Research
  - Systems & Technology Group
  - Software Group
  - PI: Mootaz Elnozahy

• University partners:
  - Cornell
  - LANL
  - MIT
  - Purdue University
  - RPI
  - UC Berkeley
  - U. Delaware
  - U. Illinois
  - U. New Mexico
  - U. Pittsburgh
  - UT Austin
  - Vanderbilt University

• Contributors to X10 design & implementation ideas:
  - David Bacon
  - Bob Blainey
  - Philippe Charles
  - Perry Cheng
  - Julian Dolby
  - Kemal Ebcioglu
  - Guang Gao (U Delaware)
  - Allan Kielstra
  - Robert O'Callahan
  - Filip Pizlo (Purdue)
  - Christoph von Praun
  - V.T.Rajan
  - Lawrence Rauchwerger (Texas A&M)
  - Vijay Saraswat (contact for lang. spec.)
  - Vivek Sarkar
  - Mandana Vaziri
  - Jan Vitek (Purdue)
Performance and Productivity Challenges facing Future Large-Scale Systems

1) **Memory wall**: Severe *non-uniformities* in bandwidth & latency in memory hierarchy

   ![Diagram of memory hierarchy with different cache levels and cores]

2) **Frequency wall**: Multiple layers of *hierarchical heterogeneous parallelism* to compensate for slowdown in frequency scaling

   - Clusters (scale-out)
   - SMP
   - Multiple cores on a chip
   - Coprocessors (SPUs)
   - SMTs
   - SIMD
   - ILP

3) **Scalability wall**: Software will need to deliver ~ $10^5$-way *parallelism* to utilize large-scale parallel systems

V. Sarkar  CDP 2004 Workshop  IBM
IBM PERCS Project
(Productive Easy-to-use Reliable Computing Systems)

Increase overall productivity

Increase number of applications written

Increase development productivity

PERCS Programming Tools
performance-guided parallelization and transformation, static & dynamic checking, separation of concerns --- all integrated into a single development environment (Eclipse)

PERCS Programming Model

OpenMP

MPI

Static and Dynamic Compilers for base language w/ programming model extensions
Mature languages: C/C++, Fortran, Java
Experimental languages: X10, UPC, StreamIt, HTA/Matlab

Language Runtime + Dynamic Compilation + Continuous Optimization

PERCS System Software (K42)

PERCS System Hardware
Limitations in exploiting Compiler-Driven Performance in Current Parallel Programming Models

- **MPI**: Local memories + message-passing
  - Parallelism, locality, and “global view” are completely managed by programmer
  - Communication, synchronization, consistency operations specified at low level of abstraction
  ➔ Limited *opportunities* for compiler optimizations

- **Java threads, OpenMP**: shared-memory parallel programming model
  - Uniform symmetric view of all shared data
  - Non-transparent performance --- programmer cannot manage data locality and thread affinity at different hierarchy levels (cluster, SMT, …)
  ➔ Limited *effectiveness* of compiler optimizations

- **HPF, UPC**: partitioned global address space + SPMD execution model
  - User specifies data distribution & parallelism, compiler generates communications using owner-computes rule
  - Large overheads in accessing shared data; compiler optimizations can help applications with simple data access patterns
  ➔ Limited *applicability* of compiler optimizations
X10 Design Guidelines: Design for Productivity & Compiler/Runtime-driven Performance

• Start with state-of-the-art OO language primitives as foundation
  − No gratuitous changes
  − Build on existing skills

• Raise level of abstraction for constructs that should be amenable to optimized implementation
  − Monitors → atomic sections
  − Threads → async activities
  − Barriers → clocks

• Introduce new constructs to model hierarchical parallelism and non-uniform data access
  − Places
  − Distributions

• Support common parallel programming idioms
  − Data parallelism
  − Control parallelism
  − Divide-and-conquer
  − Producer-consumer / streaming
  − Message-passing

• Ensure that every program has a well-defined semantics
  − Independent of implementation
  − Simple concurrency model & memory model

• Defer fault tolerance and reliability issues to lower levels of system
  − Assume tightly-coupled system with dedicated interconnect
Logical View of X10 Programming Model
(Work in progress)

Granularity of place can range from single h/w thread to an entire scale-up system

- **Place** = collection of resident activities and data
  - Maps to a data-coherent unit in a large scale system

- Four storage classes:
  - Partitioned global
  - Place-local
  - Activity-local
  - Value class instances
    - Can be copied/migrated freely

- Activities can be created by
  - **async statements** (one-way msgs)
  - **future expressions**
  - **foreach & ateach** constructs

- Activities are coordinated by
  - **Unconditional atomic sections**
  - **Conditional atomic sections**
  - **Clocks** (generalization of barriers)
  - **Force** (for result of future)
Async activities: abstraction of threads

• Async statement
  - async (P) {S}: run S at place P
  - async (D) {S}: run S at place containing datum D
  - S may contain local atomic operations or additional async activities for same/different places.
• Example: percolate process to data.

```java
public void put(K key, V value) {
    int hash = key.hashCode()% D.size;
    async (D[hash]) {
        for (_ b = buckets[hash]; b != null; b = b.next) {
            if (b.k.equals(key)) {
                b.v = value;
                return;
            }
        }
    }
    buckets[hash] =
        new Bucket<K,V>(key, value, buckets[hash]);
}
```

• Async expression (future)
  - F = future(P) {E}, or F = future(D) {E}: Return the value of expression E, evaluated in place P (or the place containing datum D)
  - force F or !F: suspend until value is known
• Example: percolate data to process.

```java
public ^V get(K key) {
    int hash = key.hashCode()% D.size;
    return future (D[hash]) {
        for (_ b = buckets[hash]; b != null; b = b.next) {
            if (b.k.equals(key)) {
                return b.v;
            }
        }
    }
    return new V();
}
```

Distributed hash-table example
RandomAccess (GUPS) example

```java
public void run(int a[], int seed[], int cyclic,
                int value smallTable[]) {
    ateach (start : seed clocked c) {
        int ran = start;
        for (int count : 1.. N_UPDATES/place.MAX_PLACES) {
            ran = Math.random(ran);
            int j = F(ran); // function F() can be in C/Fortran
            int k = smallTable[g(ran)];
            async (a[j]) atomic {a[j]^=k;}
        } // for
    } // ateach
    next c;
}
```
Regions and Distributions

- **Regions**
  - The domain of some array; a collection of array indices
  - region \( R = [0..99]; \)
  - region \( R2 = [0..99,0..199]; \)

- **Region operators**
  - region Intersect = \( R3 \&\& R4; \)
  - region Union = \( R3 \| R4; \)
  - Etc.

- **Distributions**
  - Map region elements to places
    - distribution \( D = \text{cyclic}(R); \)
  - Domain and range restriction:
    - distribution \( D2 = D \mid R; \)
    - distribution \( D3 = D \mid P; \)

- Regions/Distributions can be used like type and place parameters
  - \langle region R, distribution D\rangle
  - void m(...)
ArrayCopy example: example of high-level optimizations of async activities

**Version 1 (original):**

```java
<value T, D, E> public static void arrayCopy( T[D] a, T[E] b) {
    // Spawn an activity for each index to fetch and copy the value
    ateach (i : D.region)
        a[i] = async b[i];
    next c; // Advance clock
}
```

**Version 2 (optimized):**

```java
<value T, D, E> public static void arrayCopy( T[D] a, T[E] b) {
    // Spawn one activity per place
    ateach ( D.places )
        for ( j : D | here )
            a[i] = async b[i];
    next c; // Advance clock
}
```

**Version 3 (further optimized):**

```java
<value T, D, E> public static void arrayCopy( T[D] a, T[E] b) {
    // Spawn one activity per D-place and one future per place p to which E maps an index in (D | here).
    ateach ( D.places ) {
        region LocalD = (D | here).region;
        ateach ( p : E[LocalD] ) {
            region RemoteE = (E | p).region;
            region Common = LocalD && RemoteE;
            a[Common] = async b[Common];
        }
    }
    next c; // Advance clock
}
```
Uniform treatment of Arrays & Loops and Collections & Iterators

• **Arrays**
  - Map region elements to values (therefore multidimensional)
  - Declared with a given distribution
  - int[D] array;

• **Loops**
  - foreach (D[R]) { ... }
  - foreach (array) { ... }
  - foreach (i : R) { ... }
  - foreach (i : D) { ... }
  - foreach (i : array) { ... }
  - sequential variants of foreach are available as for loops

• **Distributed Collections**
  - Map collection elements to places
  - Collection<D,E> identifies a collection with distribution D and element type E

• **Parallel iterators**
  - foreach (e : C) { ... }
  - foreach (array) { ... here ... }

• **Sequential iterator**
  - for (e : C)
Clocks: abstraction of barriers

- **Operations:**
  
  ```
  clock c = new clock();
  now(c){S}
  • Require S to terminate before clock can progress.
  continue c;
  • Signals completion of work by activity in this clock phase.
  next c_1,...,c_n ;
  • Suspend until clocks can advance. Implicitly continues all clocks. c_1,...,c_n names all clocks for activity.
  drop c;
  • No further operations on c..
  ```

- **Semantics**
  - Clock c can advance only when all activities registered with the clock have executed `continue c`.

- **Clocked final**
  - `clocked(c)` final int l = r;
  - Variables is “final” (immutable) until next phase
Unstructured Mesh Transport Example (UMT2K)

- 3D, deterministic, multi-group, photon transport code
- Solves 1\textsuperscript{st} order form of steady-state Boltzmann equation
- Represented by an unstructured mesh
  - Partitioning strives to maintain load balance, reduce communicate/compute ratio

Figure source: Modified from Mathis and Kerbyson, IPDPS 2004
Communication Structure

- Nearest neighbor communication in graph domain
- Communication can be minimized via judicious mapping of graph to system nodes

Figure source: Modified from Mathis and Kerbyson, IPDPS 2004
do {
    now (c) {
        ateach (n : nodes) { // Cluster-level parallelism
            foreach (s : Sweeps) { // SMP parallelism
                // receive inputs
                flows = new Flux[R](k) { // SMT parallelism
                    async(...) inputs[s][k].receive();
                }
                // Choice of using clock or force to synchronize on flows[*]
                // Thread-local with vector & co-processor parallelism
                flux = compute(s, flows);
                // send outputs
                ...
            } // foreach
        } // ateach
    } // now
    // use clock c to wait for all sweeps to complete
    next c;
    ...
} while (err > MAX_ERROR);
int n;
double *A, *Tmp;
const double epsilon = 0.000001;
int main(int argc, char* argv[]) {
    int i, iters;
double delta;
    int numprocs, rank, mysize;
double sum;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (argc != 2) {
        printf("usage: fixedpt n\n");
        exit(1);
    }
    n = atoi(argv[1]);
    mysize = n * (rank+1)/numprocs - n * rank / numprocs;
    A = malloc((mysize+2)*sizeof(double));
    for (i = 0; i <= mysize; i++) A[i] = 0.0;
    if (rank == numprocs - 1)  A[mysize+1] = n + 1.0;
    Tmp = malloc((mysize+2)*sizeof(double));
    iters = 0;
do {
    iters++;
    if (rank < numprocs -1)
        MPI_Send(&A[mysize], 1, MPI_DOUBLE, rank+1, 1,
                MPI_COMM_WORLD);
    if (rank > 0)
        MPI_Recv(&A[0], 1, MPI_DOUBLE, rank-1, 1,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    if (rank > 0)
        MPI_Send(&A[1], 1, MPI_DOUBLE, rank-1, 1,
                MPI_COMM_WORLD);
    if (rank < numprocs-1)
        MPI_Recv(&A[mysize+1], 1, MPI_DOUBLE, rank+1, 1,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    for (i=1; i <=mysize; i++)  Tmp[i] = (A[i-1]+A[i+1])/2.0;
    delta = 0.0;
    for (i = 1; i <= mysize; i++)  delta +=fabs(A[i]-Tmp[i]);
    MPI_Allreduce(&delta, &sum, 1, MPI_DOUBLE, MPI_SUM,
                  MPI_COMM_WORLD);
    delta = sum;
    for (i = 1; i <= mysize; i++)  A[i]=Tmp[i];
    } while (delta > epsilon);
    if (rank == 0)  printf("Iterations: %d\n", iters);
    MPI_Finalize();
}
Reduction and Scan Operators

• Reduction operator over type T
  – Static method with signature: T(T,T)
  – Virtual method in class T with signature T(T)
  – Operator is expected to be associative and commutative

• Reduction operation: A >> foo() returns value of type T, where
  – A is an array over base type T
  – A>>foo() performs reductions over all elements of A to obtain a single result of type T

• Scan operation: A || foo() returns array, B, of base type T, where
  – B[i] = A[0..i]>foo()
Example of Unconditional Atomic Sections
SPECjbb2000: Java vs. X10 versions

**Java version:**
```java
public class Stock extends Entity {
    private float  ytd;
    private short orderCount; ...

    public synchronized void incrementYTD(short ol_quantity) {
        ytd += ol_quantity; ...
    }

    public synchronized void incrementOrderCount() {
        ++orderCount; ...
    }
}
```

**X10 version (w/ atomic section):**
```java
public class Stock extends Entity {
    private float  ytd;
    private short orderCount; ...

    public atomic void incrementYTD(short ol_quantity) {
        ytd += ol_quantity; ...
    }

    public atomic void incrementOrderCount() {
        ++orderCount; ...
    }
}
```

These two methods cannot be executed simultaneously because they use the same lock.

Atomic Sections are deadlock-free!

V. Sarkar  CDP 2004 Workshop
Example of Conditional Atomic Section

- Conditional Atomic Sections are similar to Conditional Critical Regions (CCRs)
  - Powerful construct, misuse can lead to deadlock
  - Need to identify special cases that are most useful in practice

```java
class OneBuffer<value T> {
    ?Box<T> datum = null;
    public void send(T v) {
        when (this.datum == null) {
            this.datum := new Box<T>(datum);
        }
    }
    public T receive() {
        when (this.datum !=null) {
            T v = datum.datum;
            value := null;
            return v;
        }
    }
}
```
Memory Model

• X10 focus is on data-race-free applications

• Programmer uses atomic / clock / force operations to avoid data races
  - X10 programming environment also includes data race detection tool

• Weak memory model for defining consistency of unsynchronized accesses
  - Based on Location Consistency memory mode
  - Akin to weak ordering guarantees of messages in MPI
X10 Type System: Features relevant to Compiler Optimization

- Unified type system
  - All data items are objects

- Value classes and clocked final
  - Immutable --- no updatable fields
  - However, target of object reference in a field can be mutable (if it’s not itself a value class instance)

- Type parameters
  - Places, distributions,

- Nullable
  - All types are non-null by default, need to explicitly declare a variable as nullable
  - For any type T, the type ?T (read: “nullable T”) contains all the values of type T, and a special null value, unless T already contains null.

- Support for both rectangular multidimensional arrays (matrices) and nested arrays
X10 Compilation and Runtime Environment

X10 source code

X10 Front end

X10 Classfiles

X10 static high-level optimizer

X10 Virtual Machine w/ PERCS CPO

OS

Clusters (scale-out)
- SMP
- Multiple cores on a chip
- Coprocessors (SPUs)
- SMTs
- Vector (VMX)
- ILP

Hardware parameters

Profile Feedback

V. Sarkar
## 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>
Some Challenges in Optimization of X10 programs

- Analysis and optimization of explicitly parallel programs
  - Proposed approach: use Parallel Program Graph (PPG) representation

- Analysis and optimization of remote data accesses
  - Proposed approach: perform data access aggregation and elimination using Array SSA framework

- Optimized implementation of Atomic Sections
  - Simple cases that can be supported by hardware e.g., reductions
  - Analyzable atomic sections
  - General case

- Load-balancing
  - Dynamic, adaptive migration of places

- Continuous optimization
  - Efficient implementation of scan/reduce

- Efficient invocation of components in foreign languages
  - C, Fortran

Garbage collection across multiple places
X10 Status and Plans

• Draft Language Design Report available internally w/ set of sample programs

• Implementation begun on Prototype #1 for 1/2005
  – Functional reference implementation of language subset, not optimized for performance
  – Support for calls to single-threaded native code (C, Fortran)

• Productivity experiments planned for 7/2005
  – Use prototype #1 to compare X10 w/ MPI, UPC
  – Revise language based on feedback from productivity experiments

• Prototype #2 planned for 12/2005
  – Includes design & prototype implementation of selected optimizations for parallelism, synchronization and locality in X10 programs
  – Revise language based on feedback from design evaluation

• Next phase of PERCS project planned for 7/2006 – 6/2010 timeframe
Conclusions and Future Work

- Future Large-scale Parallel Systems will be accompanied by severe productivity and performance challenges
- Summarized X10 language approach in PERCS project, with a focus on next steps:
  - Use applications and productivity studies to refine design decisions in X10
  - Prototype solutions to address implementation challenges
- Future work (beyond 2005)
  - Explore integration of X10 with other language efforts in IBM
    - XML (XJ), BPEL, …
  - Community effort to build consensus on standardized “high productivity” languages for HPC systems in the 2010 timeframe