Compiling X10 for Scalable High Performance

David Cunningham, David Grove, Igor Peshansky, Vijay Saraswat, Olivier Tardieu

IBM Watson
8th Workshop on Compiler-Driven Performance

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

• What is X10? Why should I care?
• X10 in a Nutshell
• HPC Challenge (Class 2) Results
• Compilation Challenges & Opportunities
• Conclusions
What is X10?

- X10 is a new language developed in the IBM PERCS project as part of the DARPA program on High Productivity Computing Systems (HPCS)
- X10 is an instance of the APGAS programming model in the Java family of languages
- X10 is an open-source project (http://x10-lang.org)
The current architectural landscape

Multi-core processors, with accelerators
  e.g. Sun Niagara
  e.g. Intel multicore, IXP
  e.g. IBM Cell
  e.g. GPGPUs

Road Runner: Cell-accelerated Opteron
The current architectural landscape

- Substantial architectural innovation is anticipated over the next ten years.
  - Hardware situation remains murky, but programmers need stable interfaces to develop applications.

- Heterogenous accelerator-based systems will exist, raising serious programmability challenges.
  - Programmers must choreograph interactions between heterogenous processors, memory subsystems.

- Multicore systems will dramatically raise the number of cores available to applications.
  - Programmers must understand concurrent structure of their applications.

- Applications seeking to leverage these architectures will need to go beyond data-parallel, globally synchronizing MPI model.

- These changes, while most profound for HPC now, will change the face of commercial computing over time.
Fundamental Challenge

◆ What is a good Programming Model for these machines?
  - How do we migrate existing users beyond MPI so that they can productively use these machines, specifically for HPC, at scale?
  - How do we make it easy for new classes of users to program such machines?

⇒ The need for a common programming model has never been more urgent.
Storage classes:
- Activity-local
- Place-local
- Partitioned global
- Immutable

Locality Rule:
Any access to a mutable datum must be performed by a local activity ➔ remote data accesses can be performed by creating remote activities

Locally Synchronous:
Guaranteed coherence for local heap ➔ sequential consistency

Globally Asynchronous:
No ordering of inter-place activities ➔ use explicit synchronization for coherence

Activity = sequential computation that runs at a place

Place = collection of resident activities and objects

PGAS:
- Replicated Data
- Local Heap
- Remote Heap
X10 Constructs

Fine grained concurrency
• async $S$

Atomicity
• atomic $S$
• when (c) $S$

Place-shifting operations
• at (P) $S$

Ordering
• finish $S$
• clock

Global data-structures
• points, regions, distributions, arrays

Two basic ideas: Places and Asynchrony
import x10.io.Console;

class HelloWorldPar {
    public static def main(args: Rail[String]): void {
        finish at each (p in Dist.makeUnique()) {
            Console.OUT.println("Hello World from Place" + p);
        }
    }
}

(1) x10c++ -o HelloWorldPar -O HelloWorldPar.x10

(2) mpirun -n 4 HelloWorldPar
Hello World from Place(0)  
Hello World from Place(2)  
Hello World from Place(3)  
Hello World from Place(1)

(3)
public class Fib {
    /**
     * Used as an in-out parameter to the computation.
     * When the Fib object is created, r indicates the number to compute.
     * After the computation has completed, r holds the result (Fib(r)).
     */
    var r:int;

    public def run() {
        if (r<2) return; // r already contains Fib(r)

        val f1 = new Fib(r-1);
        val f2 = new Fib(r-2);
        finish {
            async f1.run();
            f2.run();
        }
        r = f1.r + f2.r;
    }
}
Overview of Features

◆ Many sequential features of Java inherited unchanged
  □ Classes (w/ single inheritance)
  □ Interfaces, (w/ multiple inheritance)
  □ Instance and static fields
  □ Constructors, (static) initializers
  □ Overloaded, over-rideable methods
  □ Garbage collection

◆ Structs

◆ Closures

◆ Points, Regions, Distributions, Arrays

◆ Substantial extensions to the type system
  □ Dependent types
  □ Generic types
  □ Function types
  □ Type definitions, inference

◆ Concurrency
  □ Fine-grained concurrency:
    □ async (p,l) S
  □ Atomicity
    □ atomic (s)
  □ Ordering
    □ L: finish S
  □ Data-dependent synchronization
    □ when (c) S
X10 Project Status

- X10 is an open source project (Eclipse Public License)
  - Documentation, releases, mailing lists, code, etc. all publicly available via http://x10-lang.org

- XRX: X10 Runtime in X10 (14kloc and growing)

- X10 1.7.x releases throughout 2009 (Java & C++)

- X10 2.0 will be released this week (rc1 available now)
  - Java: any platform with Java 5
    - Single process (all places in 1 JVM)
  - C++:
    - Multi-process (1 place per process)
      - aix, linux, cygwin, macos, solaris
      - x86, x86_64, PowerPC, Sparc
      - x10rt: APGAS runtime (binary only) or MPI (open source)
HPC Challenge Benchmarks

• Data taken from X10/UPC HPCC'09 submission
  – (full details: http://www.x10-lang.org/hpcc09)

• Used Power 5+ Cluster at POK (v20)
  – P575+, 1.9GHz, 16CPUs/node; 64GB DDR2 memory/node; 32 compute nodes, 28 dedicated, 4 shared; gpfs
  – Dual plane HPS switch
  – Rated performance: 7.6GFlops/s per CPU

• In the process of gathering final data for SC'09 BOF
X10 Compilation Challenges

• All of the usual issues with OO languages
  – Virtual/interface dispatch
  – Small methods, class libraries & frameworks
  – ...
  – plus closures and higher-order functions

• Concurrency/Communication
  – Recognize idiomatic async/finish patterns
    reduce async termination traffic
  – Optimize message traffic
    hoist “loop invariant” messages
    eliminate unused object fields from messages
Random Access

static def runBenchmark(rails: ValRail[Rail[Long]],
  logLocalTableSize: Int, numUpdates: Long) {
  val mask = (1<<logLocalTableSize)-1;
  val local_updates = numUpdates / Place.MAX_PLACES;
  finish for ((p) in 0..Place.MAX_PLACES-1) {
    async (Place.places(p))
      @Immediate finish {
        var ran:Long = HPCC_starts(p*(numUpdates/Place.MAX_PLACES));

        for (var i:Long=0 ; i<local_updates ; ++i) {
          val place_id = ((ran>>logLocalTableSize) & (Place.MAX_PLACES-1)) as Int;
          val index = (ran & mask as Int);
          val update = ran;

          val dest = Place.places(place_id);
          val rail = rails(place_id) as Rail[Long]{self.at(dest)};
          @Immediate async (dest) {
            rail(index) ^= update;
          }
          ran = (ran << 1) ^ (ran<0L ? POLY : 0L);
        }
      }
  }
}
X10 Compilation Opportunities

• Exploiting dependent types
  Drive method specialization and loop versioning

• User directed concurrency refactoring, annotation-driven loop transformations, use IDE tooling to enable iterative loop between user & compiler.

• X10 compiled to both C++ and Java
  Neither is always the best choice. Are there interesting things to be learned by studying together?
Conclusions

• X10/APGAS: a programming language/model for multi-core, clusters, accelerators

• Abundance of interesting compilation challenges

• X10 Innovation Grants
  Short timeline: due 11/25, awarded late 2009/early 2010
course materials, applications/frameworks/DSLs, tools

• More information on X10: http://x10-lang.org