Compiler Challenges for Multicore Parallel Systems

Vivek Sarkar
Rice University

vsarkar@rice.edu

Workshop on Architectures and Compilers for Multithreading
Acknowledgments

- IBM X10 project (x10.sf.net)
- IBM Jalapeno project and Jikes Research Virtual Machine (jikesrvm.org)
- IBM ASTI project
- IBM PTRAN Project
- Java Concurrency Utilities open source project (gee.cs.oswego.edu/dl/concurrency-interest)
- Rice Co-Array Fortran project (www.hipersoft.rice.edu/caf)
- Rice COMP 635 Seminar on Heterogeneous Processors (www.cs.rice.edu/~vsarkar/comp635)
- Rice Habanero project team members
Future Architecture Trends: a new Era of Parallel Processing

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

Homogeneous Multi-core

Heterogeneous Accelerators

High Performance Clusters

Challenge: Develop new compiler technologies to support portable parallel abstractions for future hardware
X10 Background (x10.sf.net)

- X10 language developed as part of IBM PERCS project in DARPA High Productivity Computing Systems program
- Unified abstractions of asynchrony and concurrency for use in
  - Multi-core SMP Parallelism
  - Messaging and Cluster Parallelism
- Productivity
  - High Level Language designed for portability and safety
    - Build on sequential subset of Java language
    - Target adoption by mainstream developers with Java/C/C++ skills
  - X10 Development Toolkit for Eclipse
- Performance
  - Lightweight threads (activities) and synchronization
  - Transparency – expert programmer should have controls to tune optimizations and deployment
  - Efficient foreign function interfaces for libraries written in Fortran and C/C++
- “X10: An Object-Oriented Approach to Non-Uniform Cluster Computing”, OOPSLA 2005
X10 Language and Computation Model

Activity creation and termination
- async \[\text{clocked(...)}\] \(Stm\)
  - Run \(Stm\) asynchronously at Place
- foreach \[\text{clocked(...)}\] \(\text{(point } P : \text{Reg}) \ Stm\)
  - Run \(Stm\) asynchronously for each point in region
- finish \(Stm\)
  - Execute \(Stm\), wait for all asyncs to terminate

Activity coordination
- atomic \(Stm\)
  - Execute \(Stm\) atomically
- next
  - suspend till all clocks that the current activity is registered with can advance

Places
- Region — set of points, e.g., region \(r = [1:N,1:M]\);
- Distribution — mapping from region to places, e.g., dist \(d = \text{block}(r)\);
- Activity alignment
  - here — place at which current activity is executing
  - async \((\text{Place}) \ [\text{clocked(...)}] \ Stm\)
    - Run \(Stm\) asynchronously at Place
  - ateach \[\text{clocked(...)}\] \(\text{(point } P : \text{Dist}) \ Stm\)
    - Run \(Stm\) asynchronously for each point in Dist, at its place

Data Alignment
- new — Allocate object at this place (here)
- new \(T[d]\) — allocate array with distribution \(d\)

Deadlock safety: any X10 program written with async, atomic, finish, foreach, ateach, and next can never deadlock
X10 places

- Dynamic parallelism with a Partitioned Global Address Space
- Places encapsulate binding of activities and globally addressable mutable data
  - Number of places currently fixed at launch time
- All concurrency is expressed as asynchronous activities — subsumes threads, structured parallelism, messaging, DMA transfers, etc.
- Locality rule: all accesses to mutable data in an atomic section must be place-local
- Immutable data is place-independent and offers opportunity for single-assignment parallelism

Storage classes:
- Activity-local
- Place-local
- Partitioned global
- Immutable
Comparison with other languages

- Single Program Multiple Data (SPMD) languages with Partition Global Address Space (PGAS)
  - Unified Parallel C, Co-Array Fortran, Titanium
  - X10 generalizes PGAS to a "threaded-PGAS" model (beyond SPMD)
- Hierarchical fork-join parallelism
  - Cilk (ultra-lightweight threads, work-stealing scheduling, …)
  - X10 generalizes Cilk by adding places, distributions, finish-async, …
- X10 has similarities with other languages in DARPA HPCS program — Chapel (Cray) and Fortress (Sun) — but there are also key differences
  - Chapel allows object migration and data redistribution, which could makes it harder to use for scalable parallelism
  - Fortress is advancing the underlying sequential language in novel ways that are orthogonal to parallelism
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Place-local analysis & transformation
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
// X10 pseudo code
main() { // implicit finish
    Activity A0 (Part 1);
    async {A1; async A2;}
    try {
        finish {
            Activity A0 (Part 2);
            async A3;
            async A4;
        }
        catch (...) { ... }
        Activity A0 (Part 3);
    }
}
X10 Program Structure Tree (Static Representation)

- The PST for an X10 procedure is a rooted tree with six types of nodes:
  - Root node --- represents entire procedure
  - Async node --- represents an async statement
    - Async node is annotated with its destination place expression
  - Finish node --- represents a finish statement
  - Atomic node --- represents an atomic statement
  - Loop node --- represents a sequential loop statement
    - A parallel loop is modeled as a sequential loop with an async body
  - Other statement --- represents a leaf node in the PST
- Parent relation in PST is determined by program structure
  - PST.parent(N) is the node that represents the closest enclosing async/finish/atomic/loop statement (or root node if none)
Optimization of Finish and Async Operations

Research Problems

- Use PDG-style dependence/interference analysis to coarsen scope of finish nodes and insert additional async nodes (additional automatic parallelization)
- Program partitioning to extract useful parallelism from ideal parallelism
- Reorder nodes to further increase finish-async parallelism e.g., “Instruction Reordering for Fork-Join Parallelism”, PLDI 1990

Without reordering:

```plaintext
A;
finish {
    async B; async C;
}
```
```plaintext
D;
F;
// Completion time = 202
```

Without reordering:

```plaintext
A;
C;
finish {
    async B; async E;
}
```
```plaintext
D;
F;
// Completion time = 104
```
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Place-local analysis & transformation
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
Optimization Opportunities for Atomic Blocks

- Move as much code out of atomic as possible
- Use nonblocking operations for selected atomic statements
- Optimization of Transactional Memory operations for atomic
  - TX_BEGIN, TX_COMMIT, TX_ABORT, TX_VALIDATE, TX_ACCESS, TX_ACQUIRE, TX_READ, TX_WRITE, TX_EXPIRE
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Place-local analysis & transformation
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
X10 Clock Extensions

- Allow activity to be registered on clock for wait, notify, or both
  - Clock can now be used for one-way synchronizations and for barriers
  - next = wait for wait-only activity, notify for notify-only activity, and wait + notify for both
- Allow next statement to have an optional “single” statement
Optimization Opportunities for Clocks

- Split-phase barriers
  - Replace next by notify & wait, and insert local computations in between them
- Distributed barriers
  - Replace single clock by multiple clocks with fine-grained synchronization
- Use of single with next
  - e.g., replace two next operations in the following code fragment ...

```c
next;
if (id == 0 )
    if (l != k) { t = col_k[l]; col_k[l] = col_k[k]; col_k[k] = t; }
next;
```
- ... by one next operation with a single computation

```c
next { if (l != k) { t = col_k[l]; col_k[l] = col_k[k]; col_k[k] = t; } }
```
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Place-local analysis & transformation
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
Place-local analysis & transformation

- Locality Analysis
  - Augment with place type declarations by user, and augment with type inference
    - “Type Inference for Locality Analysis of Distributed Data Structures”, PPoPP 2008 (to appear)

- Automatic selection of data and computation mappings e.g.,

```c
// Implicit and explicit versions of remote fetch-and-op
a) a.x = f(a.x, b.y) ;
b) async (b) {
    final int v = b.y;
    async (a) a.x = f(a.x,v);
}
```
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Place-local analysis & transformation
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
X10 language defines mapping from X10 objects & activities to X10 places.

X10 deployment defines mapping from virtual X10 places to physical processing elements.

**Homogeneous Multi-core**

**X10 Data Structures**

**X10 Places**

**Physical PEs**

**Clusters**

**Interconnect**
Possible X10 Deployment on Nvidia G80
(with extensions to support hierarchies of places)
**Possible X10 Deployment for Cell**

**Basic Approach:**
- Map 9 places on to PPE + eight SPEs
- Use finish & async's as high-level representation of DMAs

**Challenges:**
- Weak PPE
- SIMDization is critical
- Lack of hardware support for coherence
- Limited memory on SPE's
- Limited performance of code with frequent conditional or indirect branches
- Different ISA's for PPE and SPE.
Selected Compiler Challenges

- Optimization of parallel operations
  - Activity creativity and termination — Async, Finish
  - Activity coordination — Atomic, Clocks
- Data and Computation Alignment — Places
- Deployment and Code Generation — Homogeneous & Heterogeneous Multicore
- Language Extensions in support of Automatic Compiler Parallelization
- Parallel Intermediate Representation (PIR)
Language Extensions in Support of Compiler Parallelization (LCPC 2007)

Language features that aid in automatic parallelization of high productivity languages:

- **Already in X10**
  - multidimensional arrays, points, regions, dependent types

- **Proposed extensions**
  - array views
  - parameter intents
  - retained (non-escaping) arrays and objects
  - pure methods
  - exception-free code regions
  - gather/reduce computations

All declarations are annotations are checked for safety e.g.,

- Compiler inserts dynamic check for “m != 0” in “j / m”
- Programmer inserts dynamic check using a type cast operator
  - `int (:nonzero) m = (int(:nonzero)) n;` // Cast to nonzero
- Compiler performs static checks of dependent types
  - `int (:nonzero) m = n;` // Need to declare n as nonzero
## Case Study: Java Grande Forum Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Series</th>
<th>Sparse*</th>
<th>SOR</th>
<th>Crypt</th>
<th>LUFact</th>
<th>FFT</th>
<th>Euler</th>
<th>MolDyn</th>
<th>Ray*</th>
<th>Monte*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-dim arrays</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Regions, Points</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array views</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In/Out/InOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disjoint</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retained</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pure method</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NonNull</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Region Dep-type</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonzero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exception free</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Reduction</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

* Sparse: SparseMatmult, Ray: RayTracer, Monte: MonteCarlo
Performance of X10 Serial and X10 Hand-Parallel relative to JGF Serial

- X10 serial is 1.2x faster than JGF serial on average
- X10 hand-parallel is 11.9x faster than X10 serial and 14.3x faster than JGF serial on average
- Future work: build a compiler that can automatically achieve X10 hand-parallel performance
Rice Habanero Multicore Software Project

Parallel Applications

1) Habanero Programming Language
2) Habanero Static Compiler
3) Habanero Virtual Machine
4) Habanero Concurrency Library
5) Habanero Toolkit

Seq Java, C, Fortran, ...

Habanero Foreign Function Interface

Eclipse Platform

Vendor tools
Java standard libraries
Vendor Platform Compilers & Libraries

Multicore OS
Multicore Hardware

(subsets)
X10  Fortress  ...

2), 3), 4) will be developed first for X10 subset
Habanero Research Topics

1) **Language Research (builds on X10)**
   - Explicit parallelism: hierarchical places for multicore
   - Implicit deterministic parallelism: array views, parameter intents, HPF-style forall, Sisal-style loops and arrays
   - Implicit non-deterministic parallelism: unordered iterators, partially ordered statement blocks

2) **Compiler research (focus of this talk!)**
   - Parallel Intermediate Representation (PIR)
   - Optimization of parallel operations
     - Activity creativity and termination
     - Activity coordination
     - Data and Computation Alignment
     - Deployment and Code Generation
   - Language Extensions in support of Automatic Compiler Parallelization

3) **Virtual machine research (builds on Jikes RVM)**
   - VM support for work-stealing scheduling algorithms with extensions for places, transactions, task groups
   - Integration and exploitation of lightweight profiling in VM scheduler and memory management system

4) **Concurrency library (builds on JUC and DSTM2 libraries)**
   - Fine-grained signal/wait, efficient transactions, new nonblocking data structures

5) **Toolkit research (builds on Rice HPCtoolkit & Eclipse PTP)**
   - Program analysis for common parallel software errors
   - Performance attribution of loops and inlined code using static and dynamic calling context
Habanero Target Applications and Platforms

Applications:

1) Parallel Benchmarks
   - SSCA's #1, #2, #3 from DARPA HPCS program
   - NAS Parallel Benchmarks
   - Java Grande Forum benchmarks
2) Signal Processing and Medical Imaging
   - Back-end processing for Compressive Sensing
     (www.dsp.ece.rice.edu/cs)
   - Contacts: Rich Baraniuk (Rice), Jason Cong (UCLA)
3) Seismic Data Processing
   - Rice Inversion project (www.trip.caam.rice.edu)
   - Contact: Bill Symes (Rice)
4) Computer Graphics and Visualization
   - Mathematical modeling and smoothing of meshes
   - Contact: Joe Warren (Rice)
5) Fock Matrix Construction
   - Contacts: David Bernholdt, Wael Elwasif, Robert Harrison, Annirudha Shet (ORNL)

Platforms:

- AMD Barcelona Quad-Core Opteron
- Clearspeed Advance X620
- DRC Coprocessor Module w/ Xilinx Virtex FPGA
- IBM Power6
- nVidia GeForce 8800GTX
- STI Cell
- Sun Niagara 2

Additional suggestions welcome!
Habanero Team

Send email to Vivek Sarkar (vsarkar@rice.edu) if you are interested in a PhD, postdoc, research scientist, or programmer position in the Habanero project, or in collaborating with us!
Advances in compilers are necessary to address the programming challenges of mainstream computing.