The Software Challenge for Multicore Processors

Vivek Sarkar
Parallel Computing Group, Rice University

vsarkar@rice.edu
Acknowledgments

- X10 open source project (x10.sf.net)
- Jikes RVM open source project (jikesrvm.org)
- Java Concurrency Utilities open source project (gee.cs.oswego.edu/dl/concurrency-interest)
- Co-Array Fortran project (www.hipersoft.rice.edu/caf)
- HPCToolkit project (www.hipersoft.rice.edu/hpctoolkit)
- Habanero project
Multi-Core & Many-Core Systems: a new Era of Mainstream Parallel Processing

Parallelism scaling replaces frequency scaling as foundation for improved performance and power/performance ➔ Profound impact on future software

Homogeneous Multi-core

Heterogeneous Multi-core

Multi-Core Cluster

Interconnect
Major Challenge: the Productivity Crisis for Parallel Software

Complexity of Current Parallel Software Stack requires high level of expertise

Only a small fraction of application developers are experts in parallelism
Outline

- Experiences from past work
  - Languages & Compilers -- X10, Co-Array Fortran
  - Runtime -- Jikes RVM, Java Concurrency Utilities
  - Tools -- HPCToolkit

- New research project at Rice on Multicore Software -- Habanero
X10 Approach

- 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
  - X10 Development Toolkit for Eclipse
- Performance
  - Extend VM+JIT model for high performance
    - 80% of new code is written for execution on VMs and managed runtimes
  - Performance transparency – don’t lock out the performance expert!
    - expert programmer should have controls to tune optimizations and tailor distributions & communications to actual deployment
- Build on sequential subset of Java language
  - Retain core values of Java --- productivity, ubiquity, maturity, security
  - Target adoption by mainstream developers with Java/C/C++ skills
- Dynamic parallelism with a Partitioned Global Address Space
- Places encapsulate binding of activities and globally addressable data
- All concurrency is expressed as asynchronous activities – subsumes threads, structured parallelism, messaging, DMA transfers (beyond SPMD)
- 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

Deadlock safety: any X10 program written with async, atomic, finish, foreach, ateach, and clocks can never deadlock
X10 Language Summary

- **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
// 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);
    }
}

IndexOutOfBoundsException
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, futures, …
- 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 make it harder to use for scalable parallelism (compared to X10)
  - Fortress is advancing the underlying sequential language in novel ways that are orthogonal to parallelism
Explicit language concurrency simplifies threading related constructs

**Single-Threaded Java**

```java
initTasks() { tasks = new ToTask[nRunsMC]; … }
public void runSerial() {
    results = new Vector(nRunsMC);
    // Now do the computation.
    PriceStock ps;
    for( int iRun=0; iRun < nRunsMC; iRun++ ) {
        ps = new PriceStock();
        ps.setInitAllTasks(initAllTasks);
        ps.setTask(tasks[iRun]);
        ps.run();
        results.addElement(ps.getResult());
    }
}
```

**Multi-Threaded Java**

```java
initTasks() { tasks = new ToTask[nRunsMC]; … }
public void runDistributed() {
    results = new x10Vector(nRunsMC);
    // Now do the computation
    PriceStock ps;
    finish at each (point[iRun] : tasks.distribution) {
        ps = new PriceStock();
        ps.setInitAllTasks((ToInitAllTasks)initAllTasks);
        ps.setTask(tasks[iRun]);
        ps.run();
        final ToResult r = ps.getResult(); // ToResult is a value type
        async(results) atomic results.v.addElement(r);
    }
}
```

**Multi-Threaded X10**

```java
initTasks() { tasks = new ToTask[dist.block([0:nRunsMC-1])]; … }
public void runDistributed() {
    results = new x10Vector(nRunsMC);
    // Now do the computation
    PriceStock ps;
    for( int iRun=0; iRun < nRunsMC; iRun++ ) {
        ps = new PriceStock();
        ps.setInitAllTasks((ToInitAllTasks)initAllTasks);
        ps.setTask(tasks[iRun]);
        ps.run();
        final ToResult r = ps.getResult(); // ToResult is a value type
        async(results) atomic results.v.addElement(r);
    }
}
```

*Source: http://www.epcc.ed.ac.uk/javagrande/javag.html - The Java Grande Forum Benchmark Suite*
Portable Parallel Programming via X10 Deployments

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

Heterogeneous Accelerators

Clusters

Interconnect
X10 Deployment on a Multicore SMP

- Basic Approach -- partition X10 heap into multiple place-local heaps
- Each X10 object is allocated in a designated place
- Each X10 activity is created (and pinned) at a designated place
- Allow an X10 activity to synchronously access data at remote places outside of atomic sections

Theory:
- Places serve as affinity hints for intra-SMP locality
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.
Human Productivity Study
(Comparison of MPI, UPC, X10)

- **Goals**
  - Contrast productivity of X10, UPC, and MPI for a statistically significant subject sample on a programming task relevant to HPCS Mission Partners
  - Validate the PERCS Productivity Methodology to obtain quantitative results that, given specific populations and computational domains, will be of immediate and direct relevance to HPCS.

- **Overview**
  - 4.5 days: May 23-27, 2005 at the Pittsburgh Supercomputing Center (PSC)
  - Pool of 27 comparable student subjects
  - Programming task: Parallelizing the alignment portion of Smith-Waterman algorithm (SSCA#1)
  - 3 language programming model combinations (X10, UPC, or C + MPI)
  - Equal environment as near as possible (e.g. pick of 3 editors, simple println stmts for debugging)
  - Provided expert training and support for each language
Data Summary

- Each thin vertical bar depicts 5 minutes of development time, colored by the distribution of activities within the interval.

- Development milestones bound intervals for statistical analysis:
  - begin/end task
  - begin/end development
  - first correct parallel output

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>UPC</th>
<th>X10</th>
</tr>
</thead>
<tbody>
<tr>
<td>obtained correct parallel output</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>did not obtain correct parallel output</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>dropped out</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

May 24  May 25  May 26  May 27
Comparing average development times between languages, several observations are clear:

- Average development time for subjects using X10 was significantly lower than that for subjects using UPC and MPI.
- The relative time debugging was approximately the same for all languages.
- X10 programmers spent relatively more time executing code and relatively less time authoring and tidying code.
- Subjects using MPI spent more time accessing documentation (tutorials were online; more documentation is available).
- A batch environment was used in this study --- use of an interactive environment will probably have a significant impact on development time results.
X10 Performance Results on a Multicore SMP (LCPC 2007)

- Compare four versions of each Java Grande benchmark
  - Sequential Java -- original version
  - Parallel Java -- original version, may have different algorithm than sequential version
  - Sequential X10 -- X10 port of Sequential Java
  - Parallel X10 -- hand-coded emulation of expected compiler parallelization

- Environment details
  - X10 v1.0 with JUC
  - Target system: p570+ 16-way Power5+ 2.2GHz SMP server
  - JVM: J9 VM (build 2.4, J2RE 1.6.0)
  - Options for Java runs:
    - -Xjit:count=0, optLevel=veryHot, ignoreIEEE -Xms1000M -Xmx1000M.
  - X10 runs used the following additional options:
    - -Xjit:x10Features, x10SkipChecks -MAX_NUMBER_OF_PLACES=1
  - X10 runtime augmented with special “one-way” synchronization mechanism to improve performance of finish and next operations
  - “Lightweight” X10 è use Java arrays instead of X10 arrays
  - For all runs, the main program was extended with a three-iteration loop, and the best of the three times was reported (to reduce perturbation of JIT compiler overheads)
Performance of Sequential and Parallel X10 (relative to Sequential Java)

- Crypt: 16.9
- FFT: 11.0
- LUFact: 14.3
- Series: 12.9
- SOR: 22.5
- SparseMat: 16.0
- Euler: 1.2
- MolDyn: 14.6
- MonteCarlo: 12.5
- RayTracer: 5.2
- Average: 12.7
Performance of Sequential + Parallel X10 & Parallel Java (relative to Sequential Java)

JGF Parallel versions are based on different sequential algorithms than Sequential Java for SOR and SparseMatMult
Performance of Sequential + Parallel X10 & Parallel Java (relative to Sequential Java)

X10 Parallel versions rewritten to match algorithms of Parallel Java versions for SOR and SparseMat
Impact of Multicore on Parallel Programming Models for High End Supercomputers

- **Existing languages**
  - MPI: de facto standard, difficult to program
  - OpenMP: lack of locality control limits scalability
  - HPF: limited class of applications, and need for heroic compilers

- **Emerging languages**
  - Co-array Fortran (CAF), Unified Parallel C (UPC), Titanium
  - Scalable performance demonstrated on high end supercomputers, but SPMD execution model is not well suited for multicore

- **HPCS languages**
  - X10, Chapel, Fortress
  - Dynamic parallelism models that are well suited for multicore, but production-strength implementations will not be available before 2010

- **Interim opportunity**
  - *Extend an emerging language (CAF) to support intra-node multithreading for multicore*
  - *Rice Co-Array Fortran project: John Mellor-Crummey, Fengmei Zhao, Bill Scherer, Guohua Jin, Yuri Dotsenko, ...*
Distributed Multithreading (DMT) Execution Model

- Classical CAF
  - Image 1
  - Image 2
  - Image N

- Multithreaded CAF
  - Image is a *locality domain*
  - Several threads per image spawned locally or remotely
    - Image 1
    - Image 2
    - Image N
Extending CAF with DMT

- Explicit co-function/co-subroutine declaration and interface
  - Blocking spawn
    - call foo(...)[p]
    - \( a = \text{bar1}(...)[p] + \text{bar2}(...)[q] + 3*\text{bar3}(...) \)
  - Non-blocking spawn
    - explicit handle
      - \( h = \text{spawn foo}(...)[p] \)
      - await_reply(h)
    - implicit handle
      - spawn foo(...)[p]
      - sync
  - Remote return: reply
  - Intra-node synchronization: lock, conditional variables
Maxval: GET vs. Co-function

Time to find the maximum of a co-array section

Normalized to local average time vs. Size of a double precision co-array section

- Local
- GET
- CF

Itanium2+Myrinet

Smaller is better
Outline

- Experiences from past work
  - Languages & Compilers -- X10, Co-Array Fortran
  - Runtime -- Jikes RVM, Java Concurrency Utilities
  - Tools -- HPCToolkit

- New research project at Rice on Multicore Software -- Habanero
Jikes Research Virtual Machine – Summary

- Key Innovations
  - Implement Jikes RVM in Java
    - JVM itself may be dynamically optimized; also helps in porting RVM to multiple platforms
  - Integrated two-compiler execution strategy with selective adaptive optimization
    - Baseline/Quick compiler + Optimizing compiler w/ three optimization levels
  - Modular high-performing parallel memory management system (MMTk)
    - Supports a range of type-accurate parallel garbage collectors: copying/noncopying, generational/nongenerational, real-time
  - Lightweight m:n implementation of Java threads on OS threads
    - Important for scalability; thread scheduler is tightly integrated with garbage collector
- Developed in Jalapeno project at IBM Research since 1998, and released as open source in 2001
- Community impact of open source release since 2001
  - 90+ universities using Jikes RVM for research and teaching
  - 155+ research publications using Jikes RVM
  - 27+ PhD dissertations based on Jikes RVM
  - 20+ courses taught using Jikes RVM as underlying infrastructure
  - In top 1% active projects on sourceforge
Jikes RVM Adaptive Optimization System
Jikes RVM scalability: pBOB performance from Feb 2000

Transactions/sec

(on 12-way AIX PPC/SMP)

Jalapeno (Feb 2000)  IBM JDK 1.1.8 (AIX)
Structure of the Jikes RVM Optimizing Compiler

- Full optimizing back-end for Java classfiles
- Optimizer and IR designed specifically for Java
- Dynamic optimization, adaptive optimization
- Table-driven instruction selection for ease in retargetability
- Optimization Levels
  - 0: simple on-the-fly opts + Linear Scan reg alloc
  - 1: + flow-insensitive opts
  - 2: + SSA-based opts (including array & pointer alias analysis)

HIR = High-level Intermediate Representation
LIR = Low-level Intermediate Representation
MIR = Machine-specific Intermediate Representation
BURS = Bottom-Up Rewrite System
Efficient & Scalable Implementation of Synchronous Queues in the Java Concurrency Utilities (Bill Scherer, Doug Lea, Michael Scott)

- **Motivation:**
  - Synchronous queues are the foundation for task-parallel runtime libraries e.g., JUC ThreadPoolExecutor
  - Original Java 5 implementation used locks

- **This work:**
  - Nonblocking algorithm for synchronous queues
  - Optional FIFO fairness
    - Fair mode extends dual queue
    - Unfair mode extends dual stack
  - Up to 14x performance gain in queue operations
    - Translates to 10x gain for ThreadPoolExecutor
  - Adopted in Java 6
Traditional Synchronization: Mutual Exclusion (Locks)

Object obj;
Lock mutex;
...
mutex.acquire();
obj.modify();
mutex.release();
...

Acquire
Critical Section
Release
Remainder
Problems with Locks

- Conceptual
  - Coarse-grained: poor scalability
  - Fine-grained: hard to write
- Semantic
  - Deadlock
  - Priority inversion
- Performance
  - Convoying
  - Intolerance of page faults and preemption
Nonblocking Synchronization

- Resilient to failure or delay of any thread
- *Optimistic* update pattern:
  1) Set-up operation (invisible to other threads)
  2) Effect all at once (atomic)
  3) Clean-up if needed (can be done by any thread)
- Atomic compare-and-swap (CAS)

```c
bool CAS(word *ptr, word e, word n) {
    if (*ptr != e) return false;
    *ptr = n; return true;
}
```
Synchronous Queue Microbenchmark Performance
(Java SE 5.0 Hotspot JVM on 16-way SunFire 6800)
ThreadPoolExecutor Microbenchmark Performance (Java SE 5.0 Hotspot JVM on 16-way SunFire 6800)

![Graph showing performance comparison of ThreadPoolExecutor with different queue configurations. The graph indicates a significant improvement, marked by a 10x increase in performance at higher thread counts.]
Outline

- Experiences from past work
  - Languages & Compilers -- X10, Co-Array Fortran
  - Runtime -- Jikes RVM, Java Concurrency Utilities
  - Tools -- HPCToolkit
- New research project at Rice on Multicore Software -- Habanero
# Multi-core Tools: Examples of Current State of the Art

<table>
<thead>
<tr>
<th>Tool</th>
<th>Approach</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel VTune</td>
<td>Trace/Timeline visualization</td>
<td>Workload imbalance, synchronization stalls</td>
</tr>
<tr>
<td></td>
<td>Thread critical path view</td>
<td></td>
</tr>
<tr>
<td>Sun Studio</td>
<td>Trace/timeline visualization</td>
<td>Synchronization stalls</td>
</tr>
<tr>
<td>Performance Analyzer</td>
<td>Hardware counter, memory allocation</td>
<td>Resource utilization</td>
</tr>
<tr>
<td>Paraver (University of Barcelona)</td>
<td>Trace/Timeline visualization</td>
<td>Automatic analysis engines</td>
</tr>
<tr>
<td></td>
<td>Profile, histogram</td>
<td></td>
</tr>
<tr>
<td>Eclipse PTP/PE</td>
<td>Trace/timeline visualization</td>
<td>Analysis engines in progress</td>
</tr>
<tr>
<td>Intel Thread Checker</td>
<td>Trace analysis</td>
<td>Data race detection, thread conflicts</td>
</tr>
<tr>
<td>TotalView Debugger (Etnus)</td>
<td>Trace/profile analysis</td>
<td>Memory leaks, allocation errors</td>
</tr>
<tr>
<td>Microsoft Visual Studio</td>
<td>Debugging of multithreaded .Net applications</td>
<td>Set breakpoints, examine variables</td>
</tr>
<tr>
<td>Eclipse Java Development Toolkit</td>
<td>Debugging of multithreaded Java applications</td>
<td>Set breakpoints, examine variables</td>
</tr>
</tbody>
</table>
Application Performance Analysis with Rice HPCToolkit

(John Mellor-Crummey, Mike Fagan, Mark Krentel, Nathan Tallent)
HPCToolkit User Interface

Integrates static and dynamic calling context to attribute performance measurements to loops and inlined code

Call path profile of Chroma:
C++ program for Lattice Quantum Chromodynamics
Outline

- Experiences from past work
  - Languages & Compilers -- X10, Co-Array Fortran
  - Runtime -- Jikes RVM, Java Concurrency Utilities
  - Tools -- HPCToolkit

- New research project at Rice on Multicore Software -- Habanero
Habanero: the Big Picture

Multicore Applications

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

Vendor Platform Compilers & Libraries

Eclipse Platform

Java standard libraries

Vendor tools

Seq Java, C, Fortran, ...

Habanero Foreign Function Interface

Multicore OS

Multicore Hardware

X10  Fortress  Sequoia  ...

(subsets)
1) Habanero Programming Language

- Use Java subset as sequential core language
- Explore extensions in three orthogonal directions
  1. Value-oriented extensions --- implicit deterministic parallelism
     - Value types, value (pure) methods
     - Type annotations, dependent types, type casts, type inference
     - Annotations and contracts on code blocks & methods
     - Multidimensional points, regions, iterators, arrays, reductions
     - ...
  2. “Sequential nondeterminism” --- implicit nondeterministic parallelism
     - Unordered iterations and collections
     - Event handling frameworks
     - ...
  3. Explicit parallelism with safety guarantees (can also be targeted by compiler)
     - Lightweight tasks (activities) and places
     - Parallel control constructs: async, finish, atomic, par loops, barriers/clocks
     - Parallel data constructs: distributions, futures, atomic
     - ...
2) Habanero Static Parallelizing & Optimizing Compiler

Research Opportunities:
- Optimization of value-oriented extensions
- Program analysis foundation for explicit parallelism
  - Extensions of data flow analysis, SSA analysis, etc.
- Integration of program analyses with specification and verification of annotations and contracts

Front End

AST

IRGen

Parallel IR (PIR)

PIR Analysis & Optimization

Interprocedural Analysis

Annotated Classfiles

C / Fortran

(restricted code regions for targeting accelerators & high-end computing)
3) Habanero Virtual Machine

- Classfile Loader
- Parallel Dynamic Compiler
- Parallel Allocators & Collectors
- Activity & Data Movement Scheduler
- Profiling and Adaptive Optimization System
- Synchronization Runtime
- Boot Image Writer (Quasi Static Compilation)

Research Opportunities:
- Back-end optimizations for manycore
- Support for runtime code specialization
- Integrating static & dynamic compilation
- Multicore garbage collection
- Extensions of work stealing algorithms:
  - Affinity
  - Transactions
  - Gang-scheduled collections of activities

4) Habanero Concurrency Library

- Affinity-based Thread Pools
- Concurrent Queues
- Nonblocking Sync. Primitives
- HW Abstraction

- Transactions
- Locks

Research Opportunities:
- Fine grained signal/wait
- New nonblocking data structures
- Efficient locks
- Efficient transactions
Opportunities for Broader Impact

- **Education**
  - Influence how parallelism is taught in introductory Computer Science courses

- **Open Source**
  - Build an open source testbed to grow ecosystem for researchers in Multicore Software area

- **Industry standards**
  - Our research results can be used as proofs of concept for new features being standardized
  - Infrastructure can provide foundation for reference implementations
Opportunities for Collaboration

- Any level of the Habanero stack --- language, compiler, virtual machine, concurrency library, tools
- Applications
  - Suggestions welcome on which applications/workloads we should target in this project
  - Some initial candidates
    - High Performance Computing for the mainstream
    - Bayesian network learning
    - Computational structural biology & Bioinformatics
- Hardware platforms
  - Suggestions welcome on what hardware we should target in this project
  - Some initial candidates
    - Homogeneous multi-core/many-core
      - Quad-core, Oct-core, ...
    - Heterogeneous processors
      - GPGPU’s, FPGA’s, accelerators, ...
Conclusion

Advances in multiple levels of software stack are necessary to address the parallel programming productivity challenge for multicore processors.