Portable Parallel Programming for Multicore Computing

Vivek Sarkar
Rice University
vsarkar@rice.edu
Acknowledgments

- Rice Habanero Multicore Software project
  - http://habanero.rice.edu
- COMP 635 Seminar on Heterogeneous Processors
- X10 open source project
  - http://x10.sf.net
- IBM Research study on “Java on Cell”
Future System Trends: a new Era of Mainstream & High End Parallel Processing

Challenge: Develop new programming technologies to support portable parallel abstractions for future hardware
Outline

1. Habanero Multicore Software Project
2. Portable Parallel Programming for Heterogeneous Processors
3. Legacy Transformation for Automatic Parallelization
Habanero Project (habanero.rice.edu)

Parallel Applications

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

Sequential C, Fortran, Java, …
Habanero Foreign Function Interface

Multicore OS
Multicore Hardware

1) will be based on an X10 subset
2), 3), 4), 5) will be developed first for 1), and then extended to support other languages

X10  F#  …
Habanero Target Applications and Platforms

Applications:

**Parallel Benchmarks**
- SSCA’s #1, #2, #3 from DARPA HPCS program
- NAS Parallel Benchmarks
- JGF, JUC, SciMark benchmarks

**Medical Imaging**
- Back-end processing for Compressive Sensing ([www.dsp.ece.rice.edu/cs](http://www.dsp.ece.rice.edu/cs))
- Contacts: Rich Baraniuk (Rice), Jason Cong (UCLA)

**Seismic Data Processing**
- Rice Inversion project ([www.trip.caam.rice.edu](http://www.trip.caam.rice.edu))
- Contact: Bill Symes (Rice)

**Computer Graphics and Visualization**
- Mathematical modeling and smoothing of meshes
- Contact: Joe Warren (Rice)

**Computational Chemistry**
- Fock Matrix Construction
- Contacts: David Bernholdt, Wael Elwasif, Robert Harrison, Annirudha Shet (ORNL)

**Habanero Compiler**
- Implement Habanero compiler in Habanero so as to exploit multicore parallelism within the compiler

Platforms:

- AMD Opteron Quad-Core
- Clearspeed Advance X620
- DRC Coprocessor Module w/ Xilinx Virtex FPGA
- IBM Cyclops-64 (C-64)
- IBM Power5+, Power6
- Intel Xeon Quad-Core
- NVIDIA Tesla S870
- STI Cell
- Sun UltraSparc T1, T2
- . . .

*Additional suggestions welcome!*
2) Habanero Static Parallelizing & Optimizing Compiler

- **Front End**
  - **Interprocedural Analysis**
  - **Habanero Language**

- **IRGen**
  - **AST**

- **PIR Analysis & Optimization**

- **Parallel IR (PIR)**

- **Annotated Classfiles**

- **C / Fortran** (restricted code regions for targeting accelerators & high-end computing)

- **Portable Managed Runtime**

- **Platform-specific static compiler**

- **Partitioned Code**

- **Front End**
  - **Habanero Foreign Function Interface**
  - **Sequential C, Fortran, Java, ...**
Evaluating “Java on Cell” on a Streaming Microbenchmark (Rajesh Bordawekar, IBM Research, 1Q2007)

Streaming integer vector add (b[j] = a[j] + c) for 32M vector size on 2.99 GHz P4 and 2.1 GHz Cell blade. Pentium version uses C code. Cell version uses Java on PPE and C on SPE.
Outline

1. Habanero Multicore Software Project
2. Portable Parallel Programming for Heterogeneous Processors
3. Legacy Transformation for Automatic Parallelization
Heterogeneous Processor Spectrum

**Dimension 1:**
Distance of accelerator from main processor (decreasing latency & bandwidth)

**Dimension 2:**
Hardware customization in accelerator (decreasing energy per operation)
Portable Parallel Programming via X10 Places

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

X10 deployment defines mapping from virtual X10 places to physical processing elements
Places (contd.)

Examples

1) \textbf{finish} \{ // Inter-place parallelism
   final int x = \ldots, y = \ldots;
   async (a) a.foo(x); // Execute at a’s place
   async (b[j]) b[j].bar(y); // Execute at b[j]’s place
\}

2) // Implicit and explicit versions of remote fetch-and-op
   a) a.x = foo(a.x, b.y);
   b) async (b) {
      final double v = b.y; // Can be any value type
      async (a) \textbf{atomic} a.x = foo(a.x, v);
   }

\[12\]
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

Thus, places serve as affinity hints for intra-SMP locality
Extending X10 Places for Cell Deployments (Habanero)

- 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.
Extending X10 Places for GPU Deployments (Habanero)
Outline

1. Habanero Multicore Software Project
2. Portable Parallel Programming for Heterogeneous Processors
3. Legacy Transformation for Automatic Parallelization
Automatic Parallelization revisited:
let’s target shiny decks instead of dusty decks!

Legacy Code → Sequential Java → Parallel X10 → X10 Runtime → Sequential Habanero

Language extensions + Parallel constructs

Automatic Parallelization → Habanero Runtime

Fine grained Synchronization (Phasers)

Language Extensions to aid Compiler Parallelization

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

- **Proposed in Habanero project**
  - 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>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array views</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In/Out/InOut</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disjoint</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retained</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>x</td>
</tr>
<tr>
<td>Region Dep-type</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduction</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Annotations are checked for safety, and are consistent with best practices in software engineering.
Experimental Results

- **Target system**
  - p570 16-way Power6 4.7GHz SMP
    - Main memory: 186GB
    - L3 cache: 32MB/chip
    - L1 cache: 128KB
  - Page size: 16GB
  - L2 cache: 4MB/core
  - SMT-off, AIX5.3J
  - IBM J9 JVM (Build 2.4, J2RE 1.6.0) used with following options in all runs
    - `-Xjit:count=0, optLevel=veryHot, ignoreIEEE -Xms1000M -Xmx1000M`

- **Benchmarks**
  - Java Grande Forum Benchmarks (Section 2 and Section 3)
    - **Java serial**: v2.0 of the JGF benchmarks, sequential Java
    - **Habanero serial**: Sequential Java with language extensions, same algorithm as JGF serial, annotations enable JVM optimization of null pointer and bounds checks
    - **Habanero parallel**: Annotations enable parallelization of Habanero serial version (hand-simulated in this study)
Performance Results on 16-core Power6 SMP (8p x 2c)

- Habanero Serial is 1.2x faster than JGF Serial on average
- Habanero Parallel (hand-simulated) is 11.9x faster than Habanero serial and 14.3x faster than JGF serial on average
Conclusion

Advances in parallel languages, compilers, and runtimes are necessary to address the programming challenges of multicore computing.
Habanero Team (Nov 2007)

Send email to Vivek Sarkar (vsarkar@rice.edu) if you are interested in the Habanero project, or in collaborating with us!