HydraVM: Extracting Parallelism from Legacy Sequential Code Using STM

Mohamed M. Saad
Mohamed Mohamedin, and
Binoy Ravindran

Hot Topics in Parallelism (HotPar '12), Berkeley, CA
Outline

- Motivation & Objectives
- Background
- Architecture
- Program Reconstruction
- Implementation
Motivations

- Multicore Era
  - More cores, not faster CPUs
  - Are we ready?

- Legacy Software Refactoring
  - Designed to use few threads
  - Multi-threading programming headache

- General Purpose Parallelization
Contribution

- General Purpose Parallelization

- Automated refactoring at Virtual Machine level (no source code required)

- **HydraVM**: Java Virtual Machine Prototype based on Jikes RVM and targets utilizing large number of cores by automatically detecting possible parallel portions of code
Related Work

- Non-speculative Parallelization [4, 16, 27, 13]
  - Compiler-based
  - Exploit loop-level parallelism
  - Limited data dependency support
- Speculative Parallelization [26, 33, 15, 20, 32, 10, 11, 24, 34, 18, 27, 13, 12, 9]
  - Program constructs
  - HW, SW or both
  - Requires source code
  - Online, offline or both
- Coarse-grain parallelism vs. fine-grain parallelism
Concurrency Control Abstraction
Like database transactions
ACID properties
Easier to program
Fine-grained performance
Composable
HTM (e.g., TCC, UTM), STM (e.g., RSTM, Deuce), and HyTM (e.g., SpHT, VTM)
Limitations

```java
public boolean add(int item) {
    Node pred, curr;
    atomic {
        pred = head;
        curr = pred.next;
        while (curr.val < item) {
            pred = curr;
            curr = curr.next;
        }
        if (item == curr.val) {
            return false;
        } else {
            Node node = new Node(item);
            node.next = curr;
            pred.next = node;
            return true;
        }
    }
}
```
How TM works?

- Optimistic concurrency
- Example: Adding 9 & 15 concurrently
Thread A adds 9 & Thread B adds 15

Thread A
Read-set: 8
Write-set:

Thread B
Read-set: 8
Write-set:
Thread A adds 9 & Thread B adds 15

Thread A
Read-set: 8, 10
Write-set:

Thread B
Read-set: 8, 10
Write-set:
Thread A adds 9 & Thread B adds 15

**Thread A**
Read-set: 8, 10
Write-set: 10 (left child pointer)

**Thread B**
Read-set: 8, 10, 14
Write-set:
Thread A adds 9 & Thread B adds 15

Thread A
Read-set: 8, 10
Write-set: 10 (left child pointer)
Committed successfully

Thread B
Read-set: 8, 10, 14
Write-set: 14 (right child pointer)
Thread A adds 9 & Thread B adds 15

Thread A
Read-set: 8, 10
Write-set: 10 (left child pointer)
Committed successfully

Thread B
Read-set: 8, 10, 14
Write-set: 14 (right child pointer)
Committed successfully
Object-based granularity

Thread A
Read-set: 8, 10
Write-set: 10
Committed successfully

Thread B
Read-set: 8, 10, 14
Write-set: 14
Conflict ➔ Abort

WAR
Comparison (STM vs. locking) *

STM versus locking on Hashtable

Comparison of STM and locking on B-Tree

Transactionsal Memory

- How it can help?
  - Course-grain parallelism
  - Simpler data dependency analysis
- Overhead
- Past efforts
  - Loop-based [22, 30]
  - Recursive programs [6]
HydraVM

Execution

Program Analysis
- Online/Offline
- Instrumentation
- Profiling based
- Slow

Program Reconstruction
- Basic Blocks Analysis
- Job Based execution

Tuning
- Re-layout Basic Blocks
- Minimize conflicts
Basic blocks
Execution graph
Profiling code injection
  Execution frequency
  Data dependency
  I/O

int C = 0;
for(int J=0; J<3; J++)
  if(Math.random() > 0.3)
    C++;
  else
    C--;

0: iconst_0
1: istore_1
2: iconst_0
3: istore_2
4: goto 29
7: invokestatic #13;
10: ldc2_w #19;
13: dcmpl
14: ifle 23
17: iinc 1, 1
20: goto 26
23: iinc 1, -1
26: iinc 2, 1
29: iload_2
30: bipush 3
32: if_icmplt 7
35: return

AFBDEFBDEFBCEFG
Superblock Detection

- Execute the profiled bytecode
  - Hot spot detection
  - Data dependency
  - Parallel execution patterns detection (Superblocks)
Superblock Detection

Example

```
1 for (Integer i = 0; i < DIMx; i++)
2    for (Integer j = 0; j < DIMx; j++)
3       for (Integer k = 0; k < DIMy; k++)
4          X[i][j] += A[i][k] * B[k][j];
```

- String factorization
  - Main’s algorithm \([21, 28]\)
    - \(ab(jb(hcfefg)^2 hi)^2\) jk

- Problems?
  - Data dependency
  - I/O
Code Reconstruction

- Job-based execution
- Producer-Consumer pattern
- Jobs run as transactions
Code Reconstruction (2)

- Collector
  - Provides **Executor** with suitable blocks as **Tasks** to execute according to flow up-to time
- Executor
  - Allocates core threads
  - Assign tasks to threads
  - Requests **Collector** for more blocks based on program flow, after all threads complete
- Maintain program consistency & correctness
  - Each transaction has a chronological order
  - A transaction commits iff
    - It is reachable
    - The program reached its chronological order
    - No conflict with older concurrent transactions
Reconstruction Tuning

- Store conflicting jobs (knowledge repository)
- Monitor conflict rate
- Tune through combining conflicting superblocks
Detecting Real Memory Dependencies

- Static Single Assignment (SSA)
- Superblock has multiple entries & multiple exits
Implementation (2)

- Method Inlining
  - Automatic
  - Partial recursion unrolling

- ByteSTM
  - VM-level STM
  - Based on RingSTM algorithm [31]
    - The Ring
    - Bloom filter [3]
    - Ordering (conflict resolution & commit postponing)
  - ByteSTM implementation is publicly available at [www.hydravm.org/bytestm]
Parallelizing Nested Loops

Implemented as parallel closed nesting transactions

Figure from: A. Turcu and B. Ravindran. On open nesting in distributed transactional memory. In Systor, 2012.
Example: Matrix Multiplication

\[ ab(jb(hcfefg)^2 hi)^2 jk \]
Experimental Evaluation

- **Testbed**
  - 8-core machine (2 Intel Xeon (E5520), each with 4 cores @ 2.27GHz)
  - Ubuntu Linux Server 10.04 LTS 64-bit
  - JikesRVM version 3.1.0

- **Benchmarks**
  - Matrix multiplication
  - JOlden benchmark suite [7] (MST, TreeAdd, TSP, BiSort)
Evaluation

- Speedup
  - 2x-5x
Summary

- HydraVM
  - JVM with automatic parallelization (refactoring)
  - At the bytecode-level
  - Exploits data-level & execution-flow parallelism
  - STM is used for program consistency
  - Promising speedup (2x-5x)
Questions

- Thank You
- Please visit us at
  - [www.hydravm.org](http://www.hydravm.org)
- Questions?