

JESSICA2: A Distributed Java Virtual Machine with Transparent Thread Migration Support



Wenzhang Zhu, Cho-Li Wang, and Francis C. M. Lau Department of Computer Science and Information Systems The University of Hong Kong {wzzhu+clwang+fcmlau}@csis.hku.hk

Outline

Motivations
Related works
JESSICA2 features
Experimental results
Conclusion & Future works

Motivation

Why Java?

- The dominant language for server-side programming
- Platform independent
- Built-in multithreading support at language level
- High-performance with Just-in-Time compilation

Why cluster?

 A cluster provides a scalable parallel hardware platform for high performance computing



Parallel/Distributed Computing using Java

RMI, Cobra ?

- Application level
- Complex programming model
- Can't take advantage of Java's multithreading features

Java Multithreading

- Running a multithreaded Java application on a cluster
- A Distributed Java Virtual Machine (DJVM) Approach

Distributed Java Virtual Machine (DJVM)

A distributed Java Virtual Machine (DJVM) spanning multiple cluster nodes can provide a true parallel execution environment for multithreaded Java applications with a Single System Image illusion to Java threads.



Sept. 26, 2002

An abstract view of (Distributed) JVM

T: Thread System E: Execution Engine M: Memory Space



Μ

Wenzhang Zhu, Cho-Li Wang, Francis C.M. Lau-CSIS, HKU

Ξ

Cluster 2002 Chicago, Illinois Sept. 26, 2002

Design issues of DJVM

- Extend TEM to distributed environment
 - T -> thread creation and migration mechanisms
 - E -> execution engine should be aware of the cluster environments
 - M -> provide a global object space in a distributed environment



Problems in existing DJVM's

Can't preserve Java's merits

- Static compilation (Hyperion, Jackal)=> No dynamic class loading
- Interpreters(cJVM,Java/DSM,JESSICA) => Can not support JIT compilation
- Manual distribution (Java/DSM)=>Need to re-write programs
- Layered design using DSM can't be tightly coupled with JVM
 - JVM runtime information can't be channeled to DSM
 - False sharing problem if page-based DSM is employed



Our strategies

- Preemptive transparent Java thread migration in JIT mode
 - No source code modification or bytecode instrumenting
 - Runtime Capturing and Restoring of thread execution context at bytecode boundary
 - Able to be executed in JIT compilation mode
 - Enable dynamic load balancing on clusters
- Embedded Global Object Space layer
 - Take advantage of JVM runtime supports to reduce object access overheads

JESSICA2 Architecture



Transparent thread migration in JIT mode?

Simple for interpreters (e.g. JESSICA)

- Interpreter sits in the bytecode decoding loop which can be stopped upon a migration flag checking
- The full state of a thread is available in the data structure of interpreter
- No register allocation
- JIT mode execution makes things complex (JESSICA2)
 - No clear bytecode boundary
 - How to deal with machine registers?
 - How to organize the stack frames?
 - How to restore an execution of native codes?

What are those functions?

Migration Points Selection

At the head of loop basic block + method

Register Context Handler

- Nondestructive register spilling: spill dirty registers at migration point without invalidation so that native codes can continue the use of registers
- Register rebuild: use register recovering stub at restoring phase
- Variable Type Deducing
 - Spill type in stacks using compression
- Java Frames Detection
 - Discover consecutive Java frames



Details of Transparent Java thread migration inside JIT compiler



Global Object Space (GOS)

Provide global heap abstraction for DJVM Home-based object coherence protocol, compliant with JVM Memory Model OO-based to reduce false sharing Non-blocking communication Use threaded I/O interface inside JVM for communication to hide the latency Adaptive object home migration mechanism Take advantage of JVM runtime information for optimization



Overview of GOS



Adaptive object home migration

Definition

 "home" of an object = the JVM that holds the master copy of an object

Problems

 cache objects need to be flushed and re-fetched from the home whenever synchronization happens

Adaptive object home migration

 if # of accesses from a thread dominates the total # of accesses to an object, the object home will be migrated to the node where the thread is running



Experimental Setting



 Pentium II 540MHz, 128MB
 Linux 2.2.1 kernel
 Connected by Fast Ethernet
 Kaffe 1.0.6

> Cluster 2002 Chicago, Illinois Sept. 26, 2002

Wenzhang Zhu, Cho-Li Wang, Francis C.M. Lau CSIS. HKU

Microbenchmarks(I)

CPI breakdown

	Frame number	1 frame	2 frames	11 frames
Capture time		(475Bytes)	(482Bytes)	(3,049Bytes)
Pasring time	Stack capturing	232	437	12,993
	Frame parsing	166	328	1,383
resolution of methods	Resolution	3,431	13,747	227,587
□ frame setup time	Frame setup	9	13	49
	Overall time	3,838	14,525	242,012

Cluster 2002 Chicago, Illinois Sept. 26, 2002

19

Microbenchmark(II)

(Execution time in microseconds)



JESSICA2 vs JESSICA

CPI(50,000,000iterations)



2

Application benchmark



C

Parallel Ray Tracing on JESSICA2 (Running at 8-node P-III cluster)

wzzhu's X desktop (ostrich01.csis.hku.hk:1)	
(wzzhu@ – – – MatPanel	
wzzhu@	Help
• (wzzhu@	70;
• (wzzhu@	🚾 wzzhu's X desktop (ostrich01.csis.hku.hk:1)
• wzzhu@o	🔟 (wzzhu@ 🔟 🛶 wz: — 🐳 MatPanel 🗙 🖌 📥
(wzzhu@	Wzzhu@ File Set File Help
MatPanel	(wzzhu@ Main.clas Main.java
wzzhu@	(wzzhu@ MainFrame [wzzhu@os [wzzhu@os [s]
wzznu@o	wzzhu@o c[wzzhu@o g8sguare, at
• ostrich03	(wzzhu@ AABox.cla AABox.iav
• ostrich04	MatPanel BoundingV BoundingV
• ostrich05	
• ostrich07	• ostrich02 Light-Sour
• ostrich08 ess	• ostrich03 Main.clas 114 ODJECTS pet
• ostrich09 bss	ostrich04 MainFrame Imzzbußos at
	ostrich05 Osequare, ostrich05
off process	ostrich07 TestCase0 Image: Second Seco
process process	ostrich08 usztku@codbieb01.coio.b/m.b/c.duordu/zabu/booob/multipocod_22
process process	ostrich09 essing i/o_iob_0x84902b8: id=54(net PlainSocketImpl socketWrite), fd=24
flor process process	• ostrich10 essing i/o job 0x84902b8; id=54(net_PlainSocketImpl_socketWrite), fd=24 • ostrich10 essing i/o job 0x84902b8; id=50(net_PlainSocketImpl_socketVrite), fd=24
process process	The wave of the second
www.rec process	processing i/o job 0x8490338: id=54(het_PlainSocketImpl_socketWrite), fd=23
process process processing i/o ich 0/8/25/28-id=5/(pet PlaipSocketion) w7751/805171ChP	processing 1/0 job 0x8490398; id=94(net_PlainSocketImpl_socketWrite), fd=2/ processing i/o job 0x8490398; id=54(net_PlainSocketImpl_socketWrite), fd=26 seconds (15 minutes)
processing i/o job vvccazic, id-svitec_i landocketimpi.covzi ladocketimpi.covzi ladocketi	processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=28 processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=28
	processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=22 processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=25 Take more than 10
	processing i/o job 0x8490398; id=54(net_PlainSocketImpl_socketWrite), fd=29
	processing i/o job 0.42490338: id=54(net_PlainSocketImpl_socketWrite), fd=26 processing i/o job 043490338: id=54(net_PlainSocketImpl_socketWrite), fd=28
	processing i/o job 0x8490398; id=54(net_PlainSocketImpl_socketWrite), fd=23
	processing i/o job 0x8490398: id=54(net_PlainSocket/mpl_socket/write), id=25
	processing 1/0 job 0x8490338; id=54(net_PlainSocketImpl_socketWrite), fd=29 processing i/o job 0x8490338; id=54(net_PlainSocketImpl_socketWrite), fd=27
	processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=26 processing i/o job 0x8490398: id=54(net_PlainSocketImpl_socketWrite), fd=28 wzzhu 6271 May 9 1998 wncCanvas.class
Wenzhang Zhu, Cho-Li Wang, Francis C	
CSIS, HKU	

Effect of Adaptive object home migration (SOR)



Cluster 2002 Chicago, Illinois Sept. 26, 2002

24

Conclusions

Transparent Java thread migration in JIT compiler enables the highperformance execution of multithreaded Java application on clusters An embedded GOS layer can take advantage of the JVM runtime information to reduce communication overhead



Works in Progress

Exploit new optimization techniques on GOS
 Incremental Distributed GC
 Add load balancing module
 Enhanced Single I/O Space to benefit more real-life applications
 Parallel I/O Support



Thanks

● Q & A

Wenzhang Zhu, Cho-Li Wang, Francis C.M. Lau CSIS, HKU

