

# Table of Contents

Biographical Sketch	iii
Acknowledgements	v
Table of Contents	vii
List of Tables	xi
List of Figures	xii
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis Contributions	4
1.1.1 Jbufs: Safe and Explicit Management of Buffers	4
1.1.2 Jstreams: Optimizing Serialization for Cluster Applications	5
1.1.3 Overview of Related Approaches	6
1.2 Thesis Overview	7
<b>2 Interfacing Java with Network Interfaces</b>	<b>9</b>
2.1 Background	10
2.1.1 The Virtual Interface Architecture	10
2.1.2 Gigaset cLAN™ GNN-1000 Cluster	13
2.1.3 Explicit Buffer Mapping: A Case for Buffer Re-Use	13
2.1.4 Java: A Safe Language	16
2.1.5 Separation between Garbage-Collected and Native Heaps	19
2.2 Java-Native Interfaces	21

2.2.1	The Marmot System	22
2.2.2	J/Direct	23
2.2.3	Java Native Interface (JNI)	23
2.2.4	Performance Comparison	24
2.2.5	Summary	27
2.3	Javia-I: Interfacing Java to the VI Architecture	27
2.3.1	Basic Architecture	27
2.3.2	Example: Ping-Pong	30
2.3.3	Implementation Status	31
2.3.4	Performance	31
2.4	Summary	33
2.5	Related Work	34
<b>3</b>	<b>Safe and Explicit Memory Management</b>	<b>36</b>
3.1	Jbufs	37
3.1.2	Example: A Typical Jbuf Lifetime	40
3.1.3	Runtime Safety Checks	41
3.1.4	Explicit De-allocation	43
3.1.5	Implementing Jbufs with a Semi-Space Copying Collector	43
3.1.6	Performance	44
3.1.7	Implications on Other Garbage-Collection Schemes	46
3.1.8	Proposed JNI Support	46
3.2	Javia-II	48
3.2.1	Basic Architecture	48
3.2.2	Example: Ping-Pong	50
3.2.3	Performance	50
3.3	pMM: Parallel Matrix Multiplication in Java	51

3.3.1	Single Processor Performance	54
3.3.2	Cluster Performance	56
3.4	Jam: Active Messages for Java	60
3.4.1	Basic Architecture	61
3.4.2	Bulk Transfers: Re-Using Jbufs	62
3.4.3	Implementation Status	63
3.4.4	Performance	64
3.5	Summary	65
3.6	Related Work	67
3.6.1	Pinned Java Objects	67
3.6.2	Safe Memory Management	69
<b>4</b>	<b>Object Serialization: A Case for Specialization</b>	<b>71</b>
4.1	Object Serialization	72
4.1.1	Performance	73
4.2	Impact of Serialization on RMI	76
4.2.1	Overview of RMI	76
4.2.2	An Implementation of Javia-I/II	77
4.2.3	Performance	79
4.3	Impact of Serialization on Applications	81
4.3.1	RMI Benchmark Suite	81
4.3.2	Performance	85
4.3.3	Estimated Impact of Serialization	89
4.4	Summary	90
4.5	Related Work	90
4.5.1	Java Serialization and RMI	90
4.5.2	High Performance Java Dialects	91

4.5.3. Compiler-Support for Serialization	92
<b>5 Optimizing Object Serialization</b>	<b>93</b>
5.1 In-Place Object De-serialization	94
5.2 Jstreams	95
5.2.1 Runtime Safety Checks	98
5.2.2 Serialization	99
5.2.3 De-Serialization	101
5.2.4 Implementing Jstreams in Marmot	101
5.2.5 Performance	102
5.2.6 Enhancements to Javia-II	104
5.2.7 Proposed JNI Support	104
5.3 Impact on RMI and Applications	105
5.3.1 "Polymorphic" RMI over Javia-I/II	106
5.3.2 Zero-Copy Array Serialization	107
5.3.3 RMI Performance	107
5.3.4 Impact on Applications	108
5.4 Summary	109
5.5 Related Work	111
5.5.1 RPC Specialization	111
5.5.2 Optimizing Data Representation	112
5.5.3 Zero-Copy RPC	113
5.5.4 Persistent Object Systems	114
<b>6 Conclusions</b>	<b>115</b>
<b>Bibliography</b>	<b>119</b>

# List of Tables

2.1 Marmot, J/Direct, and JNI's GC-related features.	24
2.2 Cost of Java-to-C downcalls.	25
2.3 Cost of C-to-Java upcalls.	25
2.4 Cost of accessing Java fields from C.	25
2.5 Cost of crossing the GC/Native separation.	26
2.6 Javia-I 4-byte round-trip latencies and per-byte overhead.	32
3.1 Jbufs overheads in Marmot.	45
3.2 Javia-II 4-byte round-trip latencies and per-byte overhead.	51
4.1 Impact of Marmot optimizations in serialization.	76
4.2 Impact of Marmot optimizations in de-serialization.	76
4.3 RMI 4-byte round-trip latencies.	81
4.4 Summary of RMI benchmark suite.	82
4.5. Communication profile of structured RMI applications.	88
4.6 Estimated impact of serialization on application performance.	88
5.1 Measured impact of jstreams on application performance.	109

# List of Figures

2.1 Virtual Interface data structures.	11
2.2 Typical in-memory representation of a Buffer object.	19
2.3 The hard separation between garbage-collected and native heaps.	20
2.4 JaviaI per-endpoint data structures.	29
2.5 Javia-I round-trip latencies.	33
2.6 Javia-I effective bandwidth.	34
3.1 Typical lifetime of a jbuf with a copying garbage collector.	39
3.2 Jbufs state diagram for runtime safety checks.	42
3.3 Javia-II per-endpoint data structures.	49
3.4 Javia-II round-trip latencies	51
3.5 Javia-II effective bandwidth	52
3.6 Performance of MM on a single 450Mhz Pentium-II.	55
3.7 Impact of safety checks on MM.	55
3.8 Communication time in pMM (64x64 matrices, 8 processors).	58
3.9 Communication time in pMM (256x256 matrices, 8 processors).	58
3.10 Overall performance of pMM (64x64 matrices, 8 processors).	59
3.11 Overall performance of pMM (256x256 matrices, 8 processors)	59
3.12 Jam round-trip latencies.	64
3.13 Jam effective bandwidth.	65

4.1 Object Serialization and De-serialization.	72
4.2 Serialization costs in three implementations of JOS.	74
4.3 De-serialization costs in three implementations of JOS.	75
4.4 RMI round-trip latencies.	80
4.5 RMI effective bandwidth.	80
4.6 Speedups of TSP and IDA.	86
4.7 Speedup of SOR.	86
4.8 Performance of EM3D on 8 processors.	87
4.9 Performance of FFT on 8 processors.	87
4.10 Performance of pMM on 8 processors.	88
5.1 Serialization with jstreams.	96
5.2 De-serialization with jstreams.	97
5.3 Jstreams state diagram for runtime safety checks.	98
5.4 Jstreams wire protocol in Marmot.	100
5.5 .Serialization overheads of jstreams in Marmot.	103
5.6 De-serialization overheads of jstreams in Marmot.	103
5.7 RMI round-trip latencies with jstreams.	107
5.8 RMI effective bandwidth with jstreams.	108