

Contents

List of Algorithms	xv
List of Figures	xix
List of Tables	xxi
Preface	xxiii
Acknowledgements	xxvii
Authors	xxix
1 Introduction	1
1.1 Explicit deallocation	2
1.2 Automatic dynamic memory management	3
1.3 Comparing garbage collection algorithms	5
Safety	6
Throughput	6
Completeness and promptness	6
Pause time	7
Space overhead	8
Optimisations for specific languages	8
Scalability and portability	9
1.4 A performance disadvantage?	9
1.5 Experimental methodology	10
1.6 Terminology and notation	11
The heap	11
The mutator and the collector	12
The mutator roots	12
References, fields and addresses	13
Liveness, correctness and reachability	13
Pseudo-code	14
The allocator	14
Mutator read and write operations	14
Atomic operations	15
Sets, multisets, sequences and tuples	15

2	Mark-sweep garbage collection	17
2.1	The mark-sweep algorithm	18
2.2	The tricolour abstraction	20
2.3	Improving mark-sweep	21
2.4	Bitmap marking	22
2.5	Lazy sweeping	24
2.6	Cache misses in the marking loop	27
2.7	Issues to consider	29
	Mutator overhead	29
	Throughput	29
	Space usage	29
	To move or not to move?	30
3	Mark-compact garbage collection	31
3.1	Two-finger compaction	32
3.2	The Lisp 2 algorithm	34
3.3	Threaded compaction	36
3.4	One-pass algorithms	38
3.5	Issues to consider	40
	Is compaction necessary?	40
	Throughput costs of compaction	41
	Long-lived data	41
	Locality	41
	Limitations of mark-compact algorithms	42
4	Copying garbage collection	43
4.1	Semispace copying collection	43
	Work list implementations	44
	An example	46
4.2	Traversal order and locality	46
4.3	Issues to consider	53
	Allocation	53
	Space and locality	54
	Moving objects	55
5	Reference counting	57
5.1	Advantages and disadvantages of reference counting	58
5.2	Improving efficiency	60
5.3	Deferred reference counting	61
5.4	Coalesced reference counting	63
5.5	Cyclic reference counting	66
5.6	Limited-field reference counting	72
5.7	Issues to consider	73
	The environment	73
	Advanced solutions	74
6	Comparing garbage collectors	77
6.1	Throughput	77
6.2	Pause time	78
6.3	Space	78
6.4	Implementation	79

6.5	Adaptive systems	80
6.6	A unified theory of garbage collection	80
	Abstract garbage collection	81
	Tracing garbage collection	81
	Reference counting garbage collection	82
7	Allocation	87
7.1	Sequential allocation	87
7.2	Free-list allocation	88
	First-fit allocation	89
	Next-fit allocation	90
	Best-fit allocation	90
	Speeding free-list allocation	92
7.3	Fragmentation	93
7.4	Segregated-fits allocation	93
	Fragmentation	95
	Populating size classes	95
7.5	Combining segregated-fits with first-, best- and next-fit	96
7.6	Additional considerations	97
	Alignment	97
	Size constraints	98
	Boundary tags	98
	Heap parsability	98
	Locality	100
	Wilderness preservation	100
	Crossing maps	101
7.7	Allocation in concurrent systems	101
7.8	Issues to consider	102
8	Partitioning the heap	103
8.1	Terminology	103
8.2	Why to partition	103
	Partitioning by mobility	104
	Partitioning by size	104
	Partitioning for space	104
	Partitioning by kind	105
	Partitioning for yield	105
	Partitioning to reduce pause time	106
	Partitioning for locality	106
	Partitioning by thread	107
	Partitioning by availability	107
	Partitioning by mutability	108
8.3	How to partition	108
8.4	When to partition	109
9	Generational garbage collection	111
9.1	Example	112
9.2	Measuring time	113
9.3	Generational hypotheses	113
9.4	Generations and heap layout	114
9.5	Multiple generations	115

9.6	Age recording	116
	En masse promotion	116
	Aging semispaces	116
	Survivor spaces and flexibility	119
9.7	Adapting to program behaviour	121
	Appel-style garbage collection	121
	Feedback controlled promotion	123
9.8	Inter-generational pointers	123
	Remembered sets	124
	Pointer direction	125
9.9	Space management	126
9.10	Older-first garbage collection	127
9.11	Beltway	130
9.12	Analytic support for generational collection	132
9.13	Issues to consider	133
9.14	Abstract generational garbage collection	134
10	Other partitioned schemes	137
10.1	Large object spaces	137
	The Treadmill garbage collector	138
	Moving objects with operating system support	139
	Pointer-free objects	140
10.2	Topological collectors	140
	Mature object space garbage collection	140
	Connectivity-based garbage collection	143
	Thread-local garbage collection	144
	Stack allocation	147
	Region inferencing	148
10.3	Hybrid mark-sweep, copying collectors	149
	Garbage-First	150
	Immix and others	151
	Copying collection in a constrained memory space	154
10.4	Bookmarking garbage collection	156
10.5	Ulterior reference counting	157
10.6	Issues to consider	158
11	Run-time interface	161
11.1	Interface to allocation	161
	Speeding allocation	164
	Zeroing	165
11.2	Finding pointers	166
	Conservative pointer finding	166
	Accurate pointer finding using tagged values	168
	Accurate pointer finding in objects	169
	Accurate pointer finding in global roots	171
	Accurate pointer finding in stacks and registers	171
	Accurate pointer finding in code	181
	Handling interior pointers	182
	Handling derived pointers	183
11.3	Object tables	184
11.4	References from external code	185

11.5	Stack barriers	186
11.6	GC-safe points and mutator suspension	187
11.7	Garbage collecting code	190
11.8	Read and write barriers	191
	Engineering	191
	Precision of write barriers	192
	Hash tables	194
	Sequential store buffers	195
	Overflow action	196
	Card tables	197
	Crossing maps	199
	Summarising cards	201
	Hardware and virtual memory techniques	202
	Write barrier mechanisms: in summary	202
	Chunked lists	203
11.9	Managing address space	203
11.10	Applications of virtual memory page protection	205
	Double mapping	206
	Applications of no-access pages	206
11.11	Choosing heap size	208
11.12	Issues to consider	210
12	Language-specific concerns	213
12.1	Finalisation	213
	When do finalisers run?	214
	Which thread runs a finaliser?	215
	Can finalisers run concurrently with each other?	216
	Can finalisers access the object that became unreachable?	216
	When are finalised objects reclaimed?	216
	What happens if there is an error in a finaliser?	217
	Is there any guaranteed order to finalisation?	217
	The finalisation race problem	218
	Finalisers and locks	219
	Finalisation in particular languages	219
	For further study	221
12.2	Weak references	221
	Additional motivations	222
	Supporting multiple pointer strengths	223
	Using Phantom objects to control finalisation order	225
	Race in weak pointer clearing	226
	Notification of weak pointer clearing	226
	Weak pointers in other languages	226
12.3	Issues to consider	228
13	Concurrency preliminaries	229
13.1	Hardware	229
	Processors and threads	229
	Interconnect	230
	Memory	231
	Caches	231
	Coherence	232

	Cache coherence performance example: spin locks	232
13.2	Hardware memory consistency	234
	Fences and happens-before	236
	Consistency models	236
13.3	Hardware primitives	237
	Compare-and-swap	237
	Load-linked/store-conditionally	238
	Atomic arithmetic primitives	240
	Test then test-and-set	240
	More powerful primitives	240
	Overheads of atomic primitives	242
13.4	Progress guarantees	243
	Progress guarantees and concurrent collection	244
13.5	Notation used for concurrent algorithms	245
13.6	Mutual exclusion	246
13.7	Work sharing and termination detection	248
	Rendezvous barriers	251
13.8	Concurrent data structures	253
	Concurrent stacks	256
	Concurrent queue implemented with singly linked list	256
	Concurrent queue implemented with array	261
	A concurrent deque for work stealing	267
13.9	Transactional memory	267
	What is transactional memory?	267
	Using transactional memory to help implement collection	270
	Supporting transactional memory in the presence of garbage collection	272
13.10	Issues to consider	273
14	Parallel garbage collection	275
14.1	Is there sufficient work to parallelise?	276
14.2	Load balancing	277
14.3	Synchronisation	278
14.4	Taxonomy	279
14.5	Parallel marking	279
	Processor-centric techniques	280
14.6	Parallel copying	289
	Processor-centric techniques	289
	Memory-centric techniques	294
14.7	Parallel sweeping	299
14.8	Parallel compaction	299
14.9	Issues to consider	302
	Terminology	302
	Is parallel collection worthwhile?	303
	Strategies for balancing loads	303
	Managing tracing	303
	Low-level synchronisation	305
	Sweeping and compaction	305
	Termination	306

15	Concurrent garbage collection	307
15.1	Correctness of concurrent collection	309
	The tricolour abstraction, revisited	309
	The lost object problem	310
	The strong and weak tricolour invariants	312
	Precision	313
	Mutator colour	313
	Allocation colour	314
	Incremental update solutions	314
	Snapshot-at-the-beginning solutions	314
15.2	Barrier techniques for concurrent collection	315
	Grey mutator techniques	315
	Black mutator techniques	317
	Completeness of barrier techniques	317
	Concurrent write barrier mechanisms	318
	One-level card tables	319
	Two-level card tables	319
	Reducing work	320
15.3	Issues to consider	321
16	Concurrent mark-sweep	323
16.1	Initialisation	323
16.2	Termination	324
16.3	Allocation	325
16.4	Concurrent marking and sweeping	326
16.5	On-the-fly marking	328
	Write barriers for on-the-fly collection	328
	Doligez-Leroy-Gonthier	329
	Doligez-Leroy-Gonthier for Java	330
	Sliding views	331
16.6	Abstract concurrent collection	331
	The collector wavefront	334
	Adding origins	334
	Mutator barriers	334
	Precision	334
	Instantiating collectors	335
16.7	Issues to consider	335
17	Concurrent copying & compaction	337
17.1	Mostly-concurrent copying: Baker's algorithm	337
	Mostly-concurrent, mostly-copying collection	338
17.2	Brooks's indirection barrier	340
17.3	Self-erasing read barriers	340
17.4	Replication copying	341
17.5	Multi-version copying	342
	Extensions to avoid copy-on-write	344
17.6	Sapphire	345
	Collector phases	346
	Merging phases	351
	Volatile fields	351
17.7	Concurrent compaction	351

Compressor	352
Pauseless	355
17.8 Issues to consider	361
18 Concurrent reference counting	363
18.1 Simple reference counting revisited	363
18.2 Buffered reference counting	366
18.3 Concurrent, cyclic reference counting	366
18.4 Taking a snapshot of the heap	368
18.5 Sliding views reference counting	369
Age-oriented collection	370
The algorithm	370
Sliding views cycle reclamation	372
Memory consistency	373
18.6 Issues to consider	374
19 Real-time garbage collection	375
19.1 Real-time systems	375
19.2 Scheduling real-time collection	376
19.3 Work-based real-time collection	377
Parallel, concurrent replication	377
Uneven work and its impact on work-based scheduling	384
19.4 Slack-based real-time collection	386
Scheduling the collector work	389
Execution overheads	390
Programmer input	391
19.5 Time-based real-time collection: Metronome	391
Mutator utilisation	391
Supporting predictability	393
Analysis	395
Robustness	399
19.6 Combining scheduling approaches: Tax-and-Spend	399
Tax-and-Spend scheduling	400
Tax-and-Spend prerequisites	401
19.7 Controlling fragmentation	403
Incremental compaction in Metronome	404
Incremental replication on uniprocessors	405
Stopless: lock-free garbage collection	406
Staccato: best-effort compaction with mutator wait-freedom	407
Chicken: best-effort compaction with mutator wait-freedom for x86	410
Clover: guaranteed compaction with probabilistic mutator lock-freedom	410
Stopless versus Chicken versus Clover	412
Fragmented allocation	412
19.8 Issues to consider	415
Glossary	417
Bibliography	429
Index	463