DaCapo Chopin Benchmark Descriptions and Statistics

In this document, we include the complete nominal statistics, LBO graphs, and post-GC heap size graphs for each benchmark. For the latency-sensitive benchmarks, we also include the simple and metered latency graphs for 2× and 6× heaps.

Author's address:

Table 1. The 48 nominal statistics used to characterize the DaCapo Chopin workloads. Not every statistic is available on or applicable to every workload. We use these to conduct principal components analysis of the diversity of the suite, and to inform our performance analysis of the workloads. The nominal statistics for each workload can be printed using DaCapo Chopin's '-p' command line option. The first letter of the metric name reflects its grouping: **A**llocation, **B**ytecode, **G**arbage collection, **P**erformance, and **U**(μ)-architecture.

Metric	Description
AOA	nominal average object size (bytes)
AOL	nominal 90-percentile object size (bytes)
AOM	nominal median object size (bytes)
AOS	nominal 10-percentile object size (bytes)
ARA	nominal allocation rate (bytes / μsec)
BAL	nominal aaload per usec
BAS	nominal aastore per usec
BEF	nominal execution focus / dominance of hot code
BGF	nominal getfield per usec
BPF	nominal putfield per usec
BUB	nominal thousands of unique bytecodes executed
BUF	nominal thousands of unique function calls executed
GCA	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	nominal GC count at 2X minimum heap size (G1)
GCM	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	nominal percentage of time spent in GC pauses at 2X minimum heap size (G1)
GLK	nominal percent 10th iteration memory leakage (10 iterations / 1 iterations)
GMD	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	nominal minimum heap size (MB) for default size *without* compressed pointers
GMV	nominal minimum heap size (MB) for vlarge size configuration (with compressed pointers)
GSS	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	nominal percentage slowdown due to forced c2 compilation compared to tiered baseline (compiler cost)
PCS	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivity to compiler)
PET	nominal execution time (sec)
PFS	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	nominal percentage slowdown due to using the interpreter (sensitivity to interpreter)
PKP	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	nominal percentage slowdown due to slower DRAM (memory speed sensitivity)
PPE	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	nominal iterations to warm up to within 1.5% of best
UAA	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	nominal backend bound (CPU)
UBM	nominal bad speculation: mispredicts
UBP	nominal bad speculation: pipeline restarts
UBS	nominal bad speculation
UDC	nominal data cache misses per K instructions
UDT	nominal DTLB misses per M instructions
UIP	nominal 100 x instructions per cycle (IPC)
ULL	nominal LLC misses M instructions
USB	nominal 100 x back end bound
USC	nominal SMT contention
USF	nominal 100 x front end bound

Table 2. The twelve most determinant nominal statistics as revealed by our principal components analysis, and their values for each of the DaCapo Chopin benchmarks. Each cell presents the rank of the respective benchmark with respect to that nominal statistic (black) and the concrete value reported (grey).

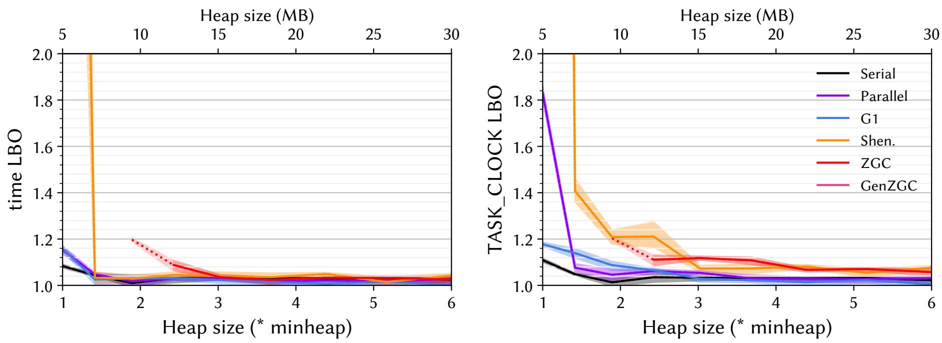
Benchmark	GCA	GCC	GCP	GMS	GSS	PCC	PET	PSD	UBM	UBP	UBS	ULL
avrora	19 (80)	18 (526)	18 (1)	18 (5)	19 (15)	21 (59)	5 (5)	2 (3)	21 (15)	22 (87)	21 (15)	13 (2606)
batik	3 (120)	20 (110)	9 (9)	11 (19)	15 (38)	5 (295)	15 (2)	13 (0)	9 (45)	4 (2135)	9 (47)	16 (1948)
biojava	10 (101)	8 (1976)	14 (2)	16 (7)	17 (29)	10 (217)	5 (5)	13 (0)	19 (28)	1 (3209)	17 (31)	18 (1586)
cassandra	4 (116)	13 (909)	18 (1)	3 (77)	16 (30)	20 (64)	3 (6)	13 (0)	14 (33)	17 (624)	14 (34)	3 (5511)
eclipse	18 (82)	11 (1026)	14 (2)	12 (13)	18 (17)	4 (359)	1 (8)	13 (0)	2 (95)	11 (978)	2 (96)	11 (3094)
fop	11 (100)	14 (820)	2 (24)	15 (9)	4 (804)	1 (1092)	19 (1)	6 (1)	3 (90)	5 (1709)	3 (91)	14 (2302)
graphchi	5 (114)	10 (1277)	14 (2)	2 (141)	7 (381)	6 (272)	9 (3)	13 (0)	22 (5)	15 (821)	22 (6)	17 (1730)
h2	12 (98)	17 (558)	11 (4)	6 (69)	14 (40)	17 (80)	15 (2)	6 (1)	16 (32)	13 (894)	15 (33)	8 (4300)
h2o	6 (112)	5 (5839)	7 (13)	8 (29)	9 (272)	11 (211)	9 (3)	3 (2)	18 (29)	9 (1153)	19 (30)	2 (8475)
jme	21 (24)	22 (31)	21 (0)	8 (29)	21 (0)	18 (71)	2 (7)	13 (0)	5 (77)	16 (701)	5 (78)	19 (1557)
jython	14 (92)	6 (3047)	10 (8)	10 (25)	3 (1421)	9 (231)	9 (3)	13 (0)	4 (83)	10 (1107)	4 (85)	20 (1394)
kafka	17 (87)	19 (247)	21 (0)	1 (157)	21 (0)	7 (266)	3 (6)	6 (1)	17 (31)	20 (421)	17 (31)	1 (8545)
luindex	15 (89)	9 (1412)	14 (2)	12 (13)	12 (64)	12 (200)	9 (3)	13 (0)	1 (109)	2 (3161)	1 (112)	21 (985)
lusearch	19 (80)	1 (17270)	3 (19)	18 (5)	10 (185)	19 (70)	9 (3)	13 (0)	14 (33)	14 (838)	15 (33)	9 (3285)
pmd	1 (136)	16 (788)	4 (18)	16 (7)	6 (508)	13 (179)	19 (1)	3 (2)	12 (37)	8 (1229)	12 (38)	6 (4435)
spring	13 (96)	7 (2462)	6 (16)	7 (43)	11 (118)	14 (137)	15 (2)	6 (1)	7 (59)	6 (1424)	7 (61)	7 (4352)
sunflow	8 (109)	3 (10937)	4 (18)	18 (5)	5 (710)	15 (90)	15 (2)	1 (13)	20 (23)	3 (2566)	20 (25)	15 (2217)
tomcat	7 (110)	4 (7676)	8 (10)	12 (13)	2 (1916)	2 (465)	7 (4)	13 (0)	10 (43)	18 (613)	10 (44)	4 (4821)
tradebeans	15 (89)	12 (965)	12 (3)	5 (73)	13 (55)	16 (83)	7 (4)	3 (2)	11 (38)	7 (1330)	11 (40)	12 (3001)
tradesoap	9 (104)	15 (818)	12 (3)	4 (75)	8 (299)	3 (461)	9 (3)	6 (1)	6 (67)	12 (977)	6 (68)	10 (3135)
xalan	2 (123)	2 (14338)	1 (77)	18 (5)	1 (7778)	22 (-1)	19 (1)	6 (1)	13 (35)	19 (586)	13 (35)	5 (4702)
zxing	22 (20)	21 (68)	18 (1)	18 (5)	20 (6)	8 (256)	19 (1)	6 (1)	8 (49)	21 (383)	8 (50)	22 (318)

1 AVRORA

This workload is based on the AVRORA simulation and analysis framework for AVR microcontrollers [1]. It is one of the most unusual workloads in DaCapo Chopin. Each simulated entity in the microcontroller is represented by a thread, so there is a high degree of fine-grained concurrency. It has the lowest allocation rate in the suite (ARA), the highest percentage of time spent in the kernel (PKP), is very insensitive to compiler selection (PCS), and is the most front end bound workload (USF). The last three of these are likely due to its very heavy use of locking primitives. It has very low back end stalls (USB), and low bad speculation (UBS). Although avrora is highly concurrent, it has very low parallel efficiency (PPE).

Table 3. Complete nominal statistics for avrora. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	1	34	18	28	58	210	nominal average object size (bytes)
AOL	1	32	19	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	0	41	20	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	4	23	12	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	7	5	7	1	4	28	nominal execution focus / dominance of hot code
BGF	5	510	9	26	507	33553	nominal getfield per usec
BPF	7	152	7	2	84	3346	nominal putfield per usec
BUB	4	33	11	8	35	177	nominal thousands of unique bytecodes executed
BUF	5	4	9	1	4	29	nominal thousands of unique function calls
GCA	2	80	19	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	2	526	18	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	2	80	18	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	2	1	18	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	0	5	22	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	1	15	17	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	2	5	18	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	0	7	22	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	2	15	19	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	3	33	14	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	1	59	21	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	1	3	21	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	8	5	5	1	3	8	nominal execution time (sec)
PFS	7	16	7	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	1	3	21	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	10	62	1	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	4	1	15	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	3	1	16	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	1	3	20	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	10	3	2	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	7	4	8	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	9	1135	4	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	0	-22	22	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	4	26	15	15	33	181	nominal backend bound (CPU)
UBM	1	15	21	5	37	109	nominal bad speculation: mispredicts
UBP	0	87	22	87	977	3209	nominal bad speculation: pipeline restarts
UBS	1	15	21	6	38	112	nominal bad speculation
UDC	6	15	9	2	13	27	nominal data cache misses per K instructions
UDT	9	641	3	13	289	1101	nominal DTLB misses per M instructions
UIP	2	106	19	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	5	2606	13	318	3001	8545	nominal LLC misses M instructions
USB	1	17	21	6	29	53	nominal 100 x back end bound
USC	2	5	19	1	53	348	nominal SMT contention
USF	10	61	1	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 1. Lower bounds on the overheads [2] for avrora for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

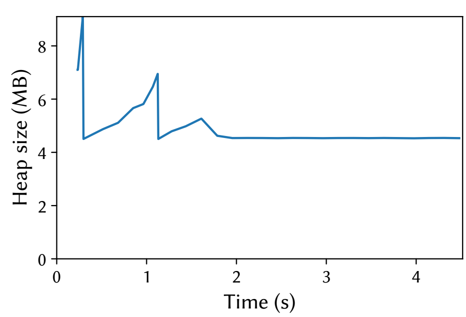


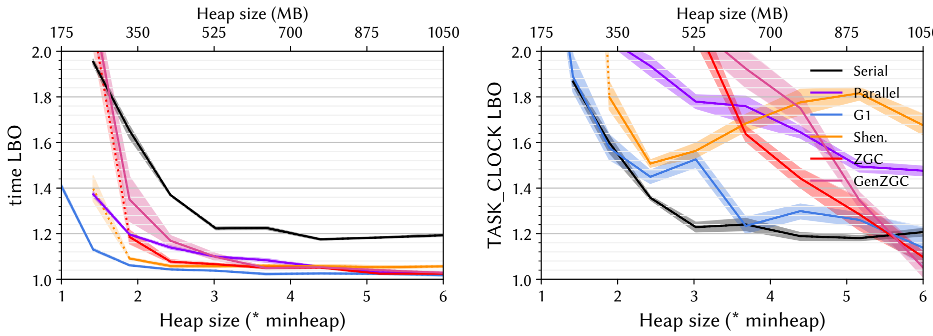
Fig. 2. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

2 BATIK

This workload uses the Batik Apache scalable vector graphics (SVG) toolkit to render a number of svg files. Batik consists of nearly 400 K lines of Java code. It has very low allocation rate (ARA) and memory turnover (GTO), and is the most sensitive workload to CPU frequency scaling (PFS). It is one of the most back end bound (USB) and one of the highest pipeline restarts (UBP), yet has one of the highest IPCs (UIP).

Table 4. Complete nominal statistics for batik. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	5	58	11	28	58	210	nominal average object size (bytes)
AOL	6	72	9	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	1	513	18	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	6	41	8	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	5	4	9	1	4	28	nominal execution focus / dominance of hot code
BGF	1	128	17	26	507	33553	nominal getfield per usec
BPF	2	28	16	2	84	3346	nominal putfield per usec
BUB	4	32	12	8	35	177	nominal thousands of unique bytecodes executed
BUF	5	4	9	1	4	29	nominal thousands of unique function calls
GCA	9	120	3	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	1	110	20	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	9	129	3	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	6	9	9	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	8	175	5	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	9	1759	2	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	5	19	11	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	9	229	3	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	4	38	15	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	0	3	20	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	8	295	5	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	3	23	16	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	4	2	15	1	3	8	nominal execution time (sec)
PFS	10	19	1	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	3	23	16	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	1	0	21	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	5	3	11	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	4	5	15	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	7	4	8	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	5	737	11	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	6	29	9	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	8	82	5	15	33	181	nominal backend bound (CPU)
UBM	6	45	9	5	37	109	nominal bad speculation: mispredicts
UBP	9	2135	4	87	977	3209	nominal bad speculation: pipeline restarts
UBS	6	47	9	6	38	112	nominal bad speculation
UDC	2	4	19	2	13	27	nominal data cache misses per K instructions
UDT	2	57	19	13	289	1101	nominal DTLB misses per M instructions
UIP	8	223	5	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	3	1948	16	318	3001	8545	nominal LLC misses M instructions
USB	10	50	2	6	29	53	nominal 100 x back end bound
USC	3	21	16	1	53	348	nominal SMT contention
USF	2	9	19	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 3. Lower bounds on the overheads [2] for batik for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

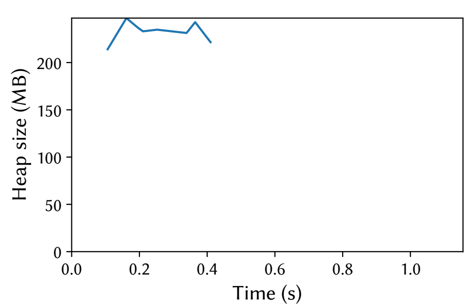


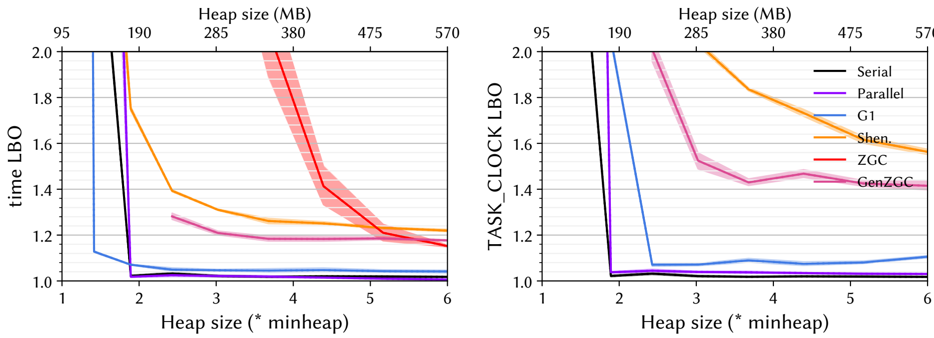
Fig. 4. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

3 BIOJAVA

(**New**) This workload uses the BioJava framework to generate ten physico-chemical properties of protein sequences of different sizes. BioJava consists of over 300 K lines of Java code. The workload has the tightest hot code focus in the suite (BEF), one of the highest IPCs (UIP), one of the lowest data cache misses (UDC), very low DTLB misses (UDT), last level cache misses (ULL), front and back end stalls (USB),USF), and SMT contention (USC), as well as the smallest average object size (AOA). Its large configuration has a 1 GB minimum heap size.

Table 5. Complete nominal statistics for biojava. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	0	28	20	28	58	210	nominal average object size (bytes)
AOL	0	24	20	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	4	2044	12	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	1	0	17	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	10	28	1	1	4	28	nominal execution focus / dominance of hot code
BGF	2	172	15	26	507	33553	nominal getfield per usec
BPF	0	2	18	2	84	3346	nominal putfield per usec
BUB	2	18	16	8	35	177	nominal thousands of unique bytecodes executed
BUF	3	2	14	1	4	29	nominal thousands of unique function calls
GCA	6	101	10	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	7	1976	8	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	5	94	12	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	4	2	14	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	6	2	10	0	0	98	nominal percent 10th iteration memory leakage
GMD	6	93	10	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	8	1029	4	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	3	7	16	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	8	183	5	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	3	29	17	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	6	99	8	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	6	217	10	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	8	103	6	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	8	5	5	1	3	8	nominal execution time (sec)
PFS	9	18	3	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	8	103	6	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	4	2	14	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	2	0	18	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	4	5	15	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	2	1	19	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	7	896	8	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	5	21	13	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	5	33	12	15	33	181	nominal backend bound (CPU)
UBM	2	28	19	5	37	109	nominal bad speculation: mispredicts
UBP	10	3209	1	87	977	3209	nominal bad speculation: pipeline restarts
UBS	3	31	17	6	38	112	nominal bad speculation
UDC	0	2	22	2	13	27	nominal data cache misses per K instructions
UDT	1	35	21	13	289	1101	nominal DTLB misses per M instructions
UIP	10	476	1	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	2	1586	18	318	3001	8545	nominal LLC misses M instructions
USB	1	18	20	6	29	53	nominal 100 x back end bound
USC	1	2	21	1	53	348	nominal SMT contention
USF	1	6	20	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 5. Lower bounds on the overheads [2] for biojava for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

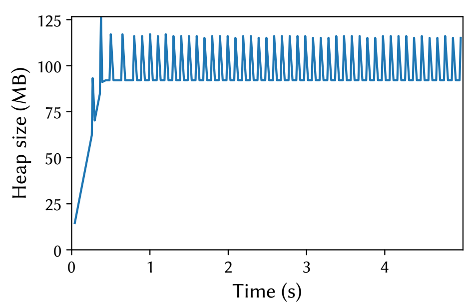


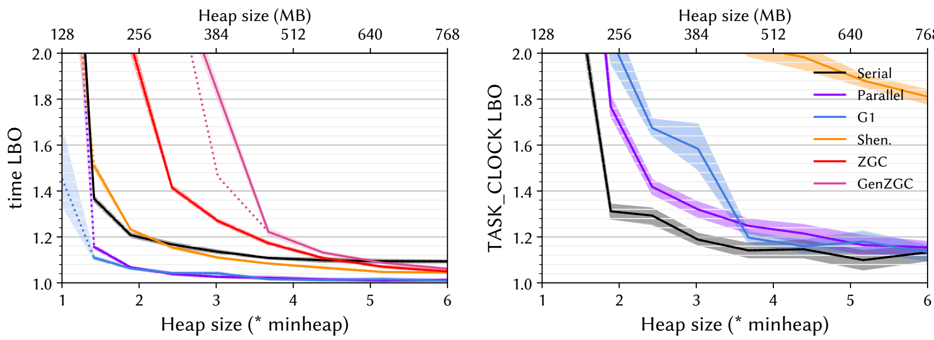
Fig. 6. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

4 CASSANDRA

(**New**) This workload executes the Yahoo! Cloud Serving Benchmark (YCSB) over the Apache Cassandra NoSQL database management system, which consists of nearly 700 K lines of Java code. It is a request-based workload, reporting request latencies. cassandra is one of the least GC-intensive workloads in the suite (GCP), but it has one of the highest DTLB miss rate (UDT) one of the highest data cache miss rates (UDC), is one of the most front end bound (USF), yielding low IPC (UIP).

Table 6. Complete nominal statistics for cassandra. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	2	38	16	28	58	210	nominal average object size (bytes)
AOL	5	56	11	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	3	908	15	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	3	11	14	0	33	2305	nominal aaload per usec
BAS	7	2	7	0	1	87	nominal aastore per usec
BEF	2	2	15	1	4	28	nominal execution focus / dominance of hot code
BGF	4	292	12	26	507	33553	nominal getfield per usec
BPF	4	55	12	2	84	3346	nominal putfield per usec
BUB	8	119	5	8	35	177	nominal thousands of unique bytecodes executed
BUF	8	19	5	1	4	29	nominal thousands of unique function calls
GCA	9	116	4	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	5	909	13	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	9	113	4	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	2	1	18	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	7	5	7	0	0	98	nominal percent 10th iteration memory leakage
GMD	6	128	9	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	5	135	10	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	9	77	3	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	6	127	9	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	3	30	16	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	4	34	13	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	1	64	20	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	4	31	15	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	9	6	3	1	3	8	nominal execution time (sec)
PFS	1	1	20	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	4	31	15	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	8	14	6	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	5	3	13	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	5	3	11	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	7	12	7	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAI	1	-4	21	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	2	23	19	15	33	181	nominal backend bound (CPU)
UBM	4	33	14	5	37	109	nominal bad speculation: mispredicts
UBP	3	624	17	87	977	3209	nominal bad speculation: pipeline restarts
UBS	4	34	14	6	38	112	nominal bad speculation
UDC	9	23	3	2	13	27	nominal data cache misses per K instructions
UDT	10	832	2	13	289	1101	nominal DTLB misses per M instructions
UIP	2	108	18	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	9	5511	3	318	3001	8545	nominal LLC misses M instructions
USB	5	30	11	6	29	53	nominal 100 x back end bound
USC	6	88	10	1	53	348	nominal SMT contention
USF	9	40	4	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 7. Lower bounds on the overheads [2] for cassandra for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

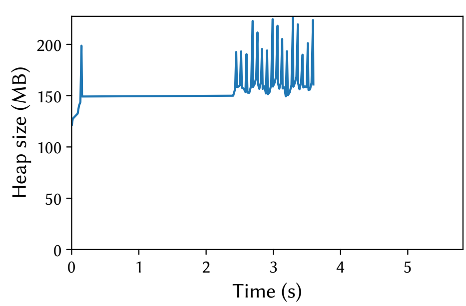
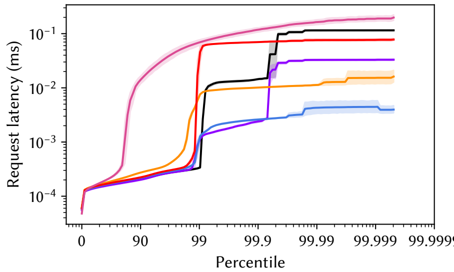
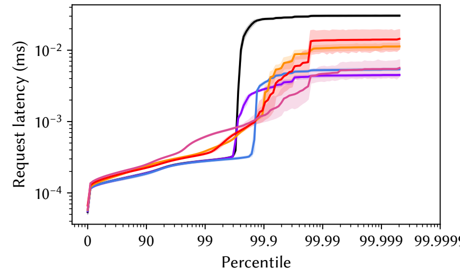


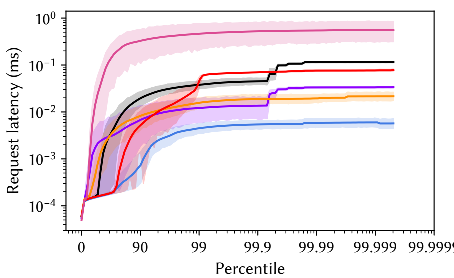
Fig. 8. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



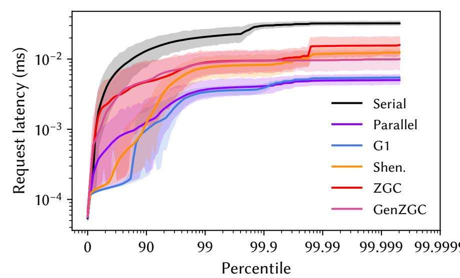
(a) Simple latency, 2.0× heap.



(b) Simple latency, 6.0× heap.



(c) Metered latency, 2.0× heap.



(d) Metered latency, 6.0× heap.

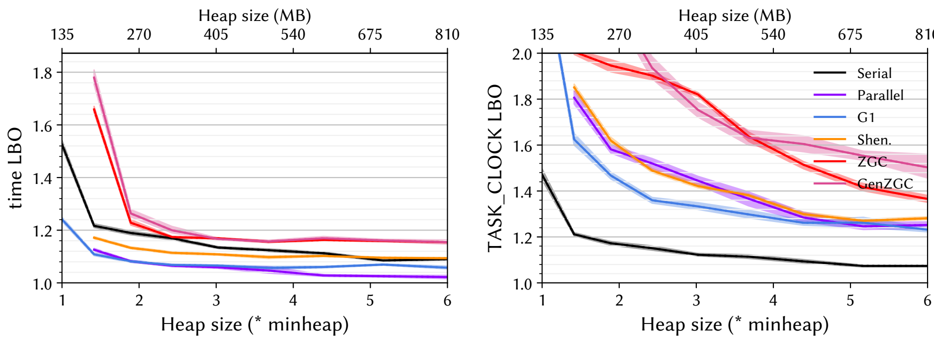
Fig. 9. Distribution of request latencies for cassandra for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

5 ECLIPSE

This workload executes the eclipse performance tests. Eclipse is a widely used IDE consisting of over 6 M lines of Java code. It is the most sensitive workload to compiler configuration (PCC), PCS) and one of the most sensitive to last level cache size (PLS) and CPU frequency scaling (PFS). It suffers high bad speculation due to mispredicts (UBS), UBM).

Table 7. Complete nominal statistics for eclipse. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	7	85	6	28	58	210	nominal average object size (bytes)
AOL	8	88	4	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	3	1045	14	41	2063	12243	nominal allocation rate (bytes / usec)
GCA	2	82	18	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	5	1026	11	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	2	74	19	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	4	2	14	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	1	11	0	0	98	nominal percent 10th iteration memory leakage
GMD	7	135	7	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	5	139	9	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	5	13	12	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	7	167	7	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	2	17	18	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	5	53	11	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	9	359	4	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	10	234	2	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	10	8	1	1	3	8	nominal execution time (sec)
PFS	9	17	4	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	10	234	2	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	7	7	8	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	9	17	3	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	6	5	10	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	5	6	13	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	3	12	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	4	664	13	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	7	30	7	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	6	39	10	15	33	181	nominal backend bound (CPU)
UBM	10	95	2	5	37	109	nominal bad speculation: mispredicts
UBP	5	978	11	87	977	3209	nominal bad speculation: pipeline restarts
UBS	10	96	2	6	38	112	nominal bad speculation
UDC	4	11	14	2	13	27	nominal data cache misses per K instructions
UDT	5	289	12	13	289	1101	nominal DTLB misses per M instructions
UIP	6	182	9	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	5	3094	11	318	3001	8545	nominal LLC misses M instructions
USB	5	29	12	6	29	53	nominal 100 x back end bound
USC	4	22	15	1	53	348	nominal SMT contention
USF	6	31	10	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 10. Lower bounds on the overheads [2] for eclipse for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

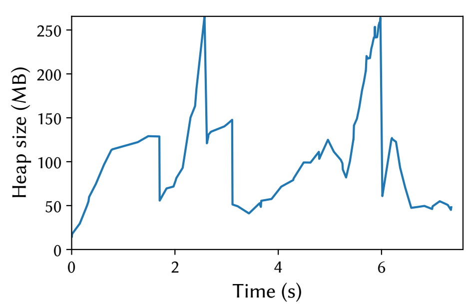


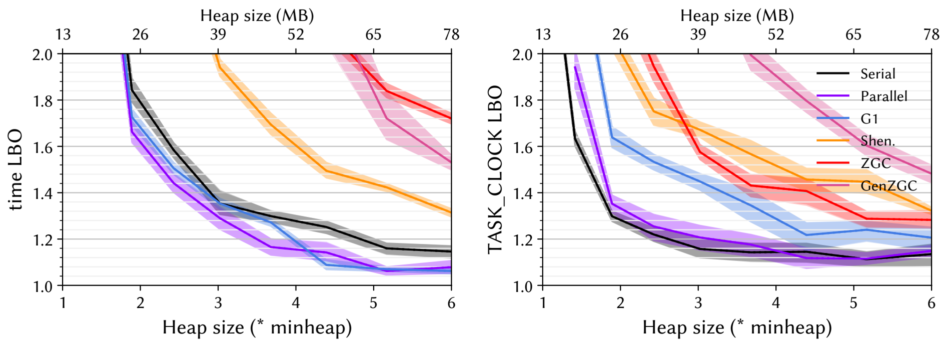
Fig. 11. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

6 FOP

This workload uses the Apache fop print formatter to render a number of XLS-FO files as pdfs. The Apache fop framework consists of over 400 K lines of Java code. fop has the largest number of unique bytecodes executed (BUB), and is the slowest benchmark to warm up (PWU). It has the highest percentage of time spent in GC pauses at a 2× heap (GCP), and is one of the most heap-size sensitive workloads (GSS). It suffers from bad speculation (UBS), UBP), UBM).

Table 8. Complete nominal statistics for fop. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	5	59	10	28	58	210	nominal average object size (bytes)
AOL	5	56	11	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	6	3300	9	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	5	33	10	0	33	2305	nominal aaload per usec
BAS	8	6	5	0	1	87	nominal aastore per usec
BEF	0	1	18	1	4	28	nominal execution focus / dominance of hot code
BGF	5	507	10	26	507	33553	nominal getfield per usec
BPF	5	92	9	2	84	3346	nominal putfield per usec
BUB	10	177	1	8	35	177	nominal thousands of unique bytecodes executed
BUF	8	26	4	1	4	29	nominal thousands of unique function calls
GCA	5	100	11	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	4	820	14	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	7	107	7	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	10	24	2	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	1	13	20	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GMS	4	9	15	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	1	17	20	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	9	804	4	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	6	77	9	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	10	1092	1	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	3	23	16	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	2	1	19	1	3	8	nominal execution time (sec)
PFS	7	14	8	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	3	23	16	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	4	2	14	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	8	13	6	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	8	7	6	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	5	9	11	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	10	9	1	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	4	663	14	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	10	38	2	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	5	35	11	15	33	181	nominal backend bound (CPU)
UBM	9	90	3	5	37	109	nominal bad speculation: mispredicts
UBP	8	1709	5	87	977	3209	nominal bad speculation: pipeline restarts
UBS	9	91	3	6	38	112	nominal bad speculation
UDC	6	14	10	2	13	27	nominal data cache misses per K instructions
UDT	4	209	14	13	289	1101	nominal DTLB misses per M instructions
UIP	6	178	10	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	4	2302	14	318	3001	8545	nominal LLC misses M instructions
USB	5	28	13	6	29	53	nominal 100 x back end bound
USC	4	26	14	1	53	348	nominal SMT contention
USF	6	31	10	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 12. Lower bounds on the overheads [2] for fop for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

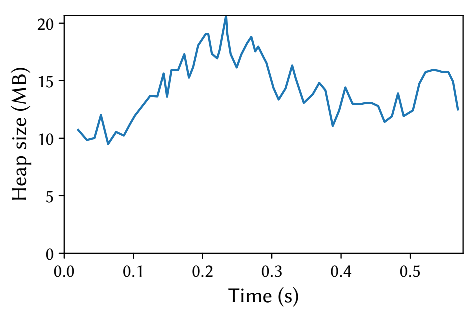


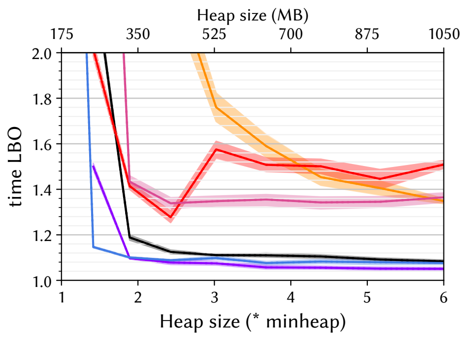
Fig. 13. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

7 GRAPHCHI

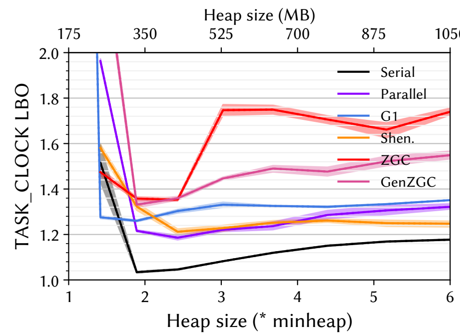
(**New**) This workload performs ALS matrix factorization using the Netflix Challenge dataset with the Java port of the GraphChi engine [3]. It has the smallest number of unique bytecodes executed (BUB) and one of the highest aaload and putfield rates (BAL), BPF). It is the most sensitive workload to compiler configuration (PCS). It has very low front end stalls (USF), bad speculation (UBM), DTLB and data cache miss rates (UDT), UDC), yielding one of the best IPCs (UIP) despite suffering SMT contention (USC) and being backend bound (USB). In its large configuration, it has a 1.1 GB minimum heap size.

Table 9. Complete nominal statistics for graphchi. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	8	109	4	28	58	210	nominal average object size (bytes)
AOL	9	160	2	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	2	16	16	16	24	24	nominal 10-percentile object size (bytes)
ARA	5	2768	10	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	9	2229	2	0	33	2305	nominal aaload per usec
BAS	5	1	9	0	1	87	nominal aastore per usec
BEF	9	12	2	1	4	28	nominal execution focus / dominance of hot code
BGF	9	9322	2	26	507	33553	nominal getfield per usec
BPF	3	44	14	2	84	3346	nominal putfield per usec
BUB	0	8	18	8	35	177	nominal thousands of unique bytecodes executed
BUF	1	1	17	1	4	29	nominal thousands of unique function calls
GCA	8	114	5	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	6	1277	10	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	8	110	6	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	4	2	14	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	8	175	5	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	9	1152	3	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	10	141	2	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	8	179	6	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	7	381	7	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	4	39	12	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	8	272	6	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	10	321	1	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	7	14	8	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	10	321	1	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	2	1	19	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	6	5	10	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	8	10	5	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	6	10	9	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	9	1194	3	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	6	28	10	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	10	181	1	15	33	181	nominal backend bound (CPU)
UBM	0	5	22	5	37	109	nominal bad speculation: mispredicts
UBP	4	821	15	87	977	3209	nominal bad speculation: pipeline restarts
UBS	0	6	22	6	38	112	nominal bad speculation
UDC	1	3	20	2	13	27	nominal data cache misses per K instructions
UDT	1	52	20	13	289	1101	nominal DTLB misses per M instructions
UIP	9	235	4	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	3	1730	17	318	3001	8545	nominal LLC misses M instructions
USB	8	38	6	6	29	53	nominal 100 x back end bound
USC	9	194	3	1	53	348	nominal SMT contention
USF	0	4	22	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 14. Lower bounds on the overheads [2] for graphchi for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

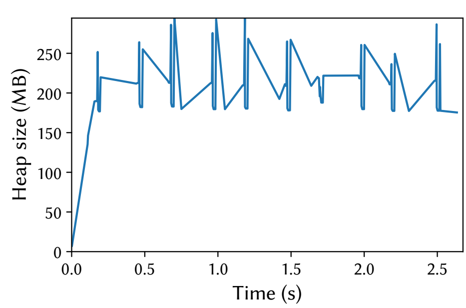


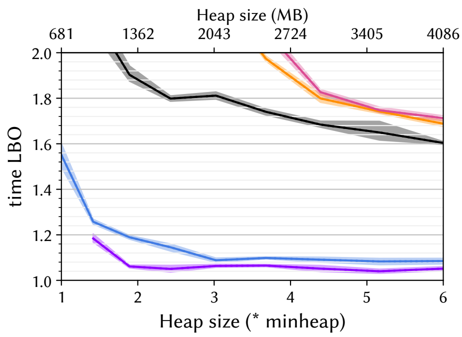
Fig. 15. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

8 H2

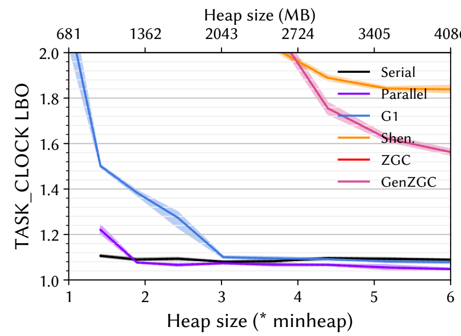
This workload is latency-sensitive. It executes a TPC-C-like transactional workload over the H2 database configured for in-memory operation. h2 has about 240 K lines of Java source code. It has the largest heap sizes for default, large, and vlarge configurations (681 MB, 10.2 GB, and 20.6 GB). It has very low memory turnover (GTO) and has the highest sensitivity to slower DRAM speeds (PMS). It has high DTLB and data cache miss rates (UDT), UDC) and high SMT contention (USC). It spends very little time in kernel mode (PKP).

Table 10. Complete nominal statistics for h2. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	3	41	14	28	58	210	nominal average object size (bytes)
AOL	5	64	10	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	9	11575	2	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	8	226	4	0	33	2305	nominal aaload per usec
BAS	9	27	3	0	1	87	nominal aastore per usec
BEF	7	7	6	1	4	28	nominal execution focus / dominance of hot code
BGF	7	3543	6	26	507	33553	nominal getfield per usec
BPF	8	582	5	2	84	3346	nominal putfield per usec
BUB	1	17	17	8	35	177	nominal thousands of unique bytecodes executed
BUF	3	2	14	1	4	29	nominal thousands of unique function calls
GCA	5	98	12	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	3	558	17	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	3	82	16	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	5	4	11	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	10	681	1	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	10	10193	1	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	8	69	6	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	10	902	1	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GMV	10	20609	1	369	1125	20609	nominal minimum heap size (MB) for vlarge size configuration (with compressed pointers)
GSS	4	40	14	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	2	31	16	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	3	80	17	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	4	54	14	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	4	2	15	1	3	8	nominal execution time (sec)
PFS	2	4	18	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	4	54	14	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	1	0	21	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	6	5	10	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	10	35	1	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	8	24	6	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	7	967	7	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	5	22	11	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	2	24	18	15	33	181	nominal backend bound (CPU)
UBM	3	32	16	5	37	109	nominal bad speculation: mispredicts
UBP	5	894	13	87	977	3209	nominal bad speculation: pipeline restarts
UBS	4	33	15	6	38	112	nominal bad speculation
UDC	7	16	7	2	13	27	nominal data cache misses per K instructions
UDT	8	498	6	13	289	1101	nominal DTLB misses per M instructions
UIP	5	138	12	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	7	4300	8	318	3001	8545	nominal LLC misses M instructions
USB	9	43	4	6	29	53	nominal 100 x back end bound
USC	8	138	5	1	53	348	nominal SMT contention
USF	3	17	17	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 16. Lower bounds on the overheads [2] for h2 for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

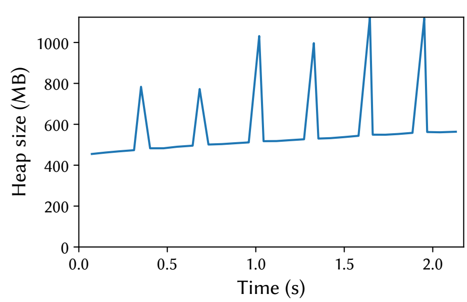
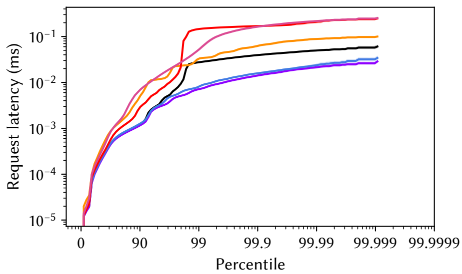
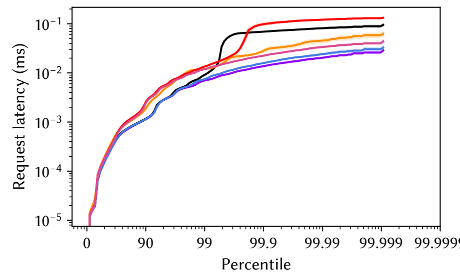


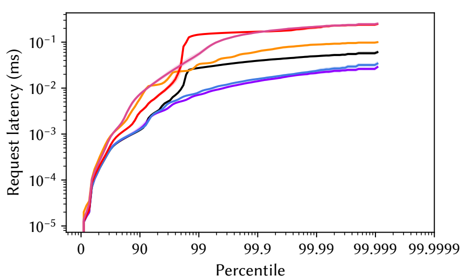
Fig. 17. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



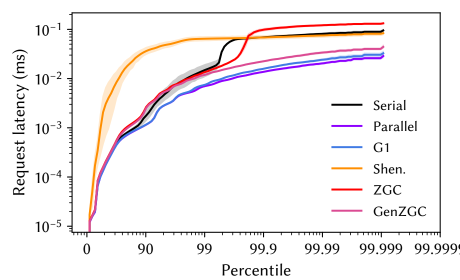
(a) Simple latency, 2.0× heap.



(b) Simple latency, 6.0× heap.



(c) Metered latency, 2.0× heap.



(d) Metered latency, 6.0× heap.

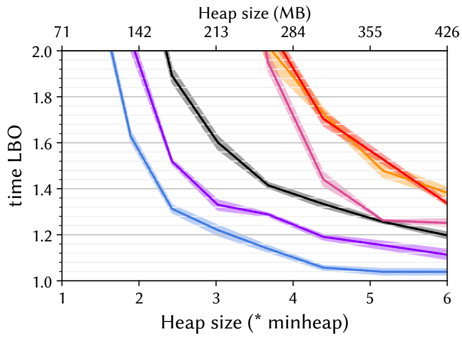
Fig. 18. Distribution of request latencies for h2 for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

9 H2O

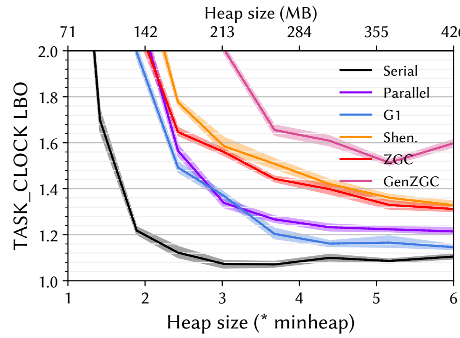
(**New**) This workloads performs machine learning using the H2O ML platform and the 201908-citibike-tripdata dataset. H2O consists of about 330 K lines of Java code. h2o is very sensitive to slower DRAM speeds (PMS) and exhibits one of the highest standard deviations among invocations (PSD). It has the one of the smallest median object sizes (AOM) but one of the largest average object sizes (AOA). It has very low IPC (UIP) and very high DTLB, last level cache and data cache miss rates (UDT),ULL), UDC) and is among the most back end bound (USB).

Table 11. Complete nominal statistics for h2o. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	9	134	2	28	58	210	nominal average object size (bytes)
AOL	9	152	3	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	2	16	16	16	24	24	nominal 10-percentile object size (bytes)
ARA	6	5143	8	41	2063	12243	nominal allocation rate (bytes / usec)
GCA	8	112	6	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	8	5839	5	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	8	112	5	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	7	13	7	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	9	16	3	0	0	98	nominal percent 10th iteration memory leakage
GMD	5	71	12	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GMS	7	29	8	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	5	73	12	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	6	272	9	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	7	172	6	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	5	211	11	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	6	69	10	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	3	8	16	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	6	69	10	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	5	5	11	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	7	7	8	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	9	17	3	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	4	5	15	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	9	2	3	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	7	4	8	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	5	701	12	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	5	21	13	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	9	112	3	15	33	181	nominal backend bound (CPU)
UBM	2	29	18	5	37	109	nominal bad speculation: mispredicts
UBP	6	1153	9	87	977	3209	nominal bad speculation: pipeline restarts
UBS	2	30	19	6	38	112	nominal bad speculation
UDC	10	24	2	2	13	27	nominal data cache misses per K instructions
UDT	8	618	5	13	289	1101	nominal DTLB misses per M instructions
UIP	0	92	22	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	10	8475	2	318	3001	8545	nominal LLC misses M instructions
USB	10	53	1	6	29	53	nominal 100 x back end bound
USC	7	105	7	1	53	348	nominal SMT contention
USF	3	18	16	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 19. Lower bounds on the overheads [2] for h2o for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

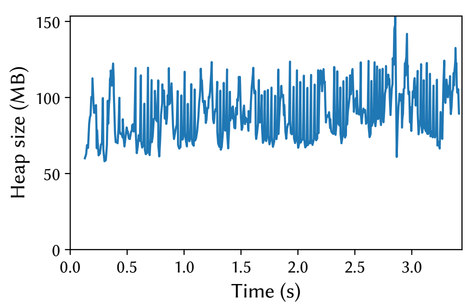


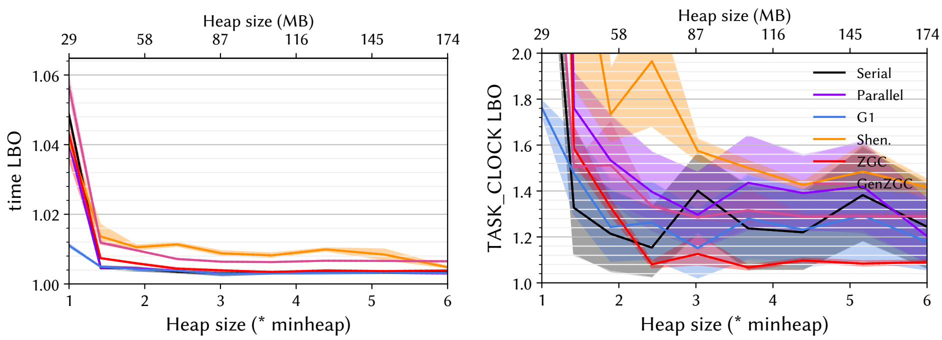
Fig. 20. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

10 JME

(**New**) This workload is latency-sensitive, using jMonkey Engine, a 3-D game development suite, to render a series of video frames. jme has about 200 K lines of Java source code. It is one of the least GC-intensive workloads (GCA), GCC), GSS), GTO). It is insensitive to frequency scaling (PFS) and warms up quickly (PWU). These factors are consistent with jme making extensive use of the GPU. It has the lowest SMT contention (USC) and is one of the most backend bound due to the CPU (UBC).

Table 12. Complete nominal statistics for jme. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	4	42	13	28	58	210	nominal average object size (bytes)
AOL	5	56	11	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	1	54	19	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	1	0	17	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	5	4	9	1	4	28	nominal execution focus / dominance of hot code
BGF	0	26	18	26	507	33553	nominal getfield per usec
BPF	1	10	17	2	84	3346	nominal putfield per usec
BUB	5	35	10	8	35	177	nominal thousands of unique bytecodes executed
BUF	5	4	9	1	4	29	nominal thousands of unique function calls
GCA	1	24	21	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	0	31	22	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	1	24	21	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	1	0	21	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	4	29	14	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	2	29	15	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	7	29	8	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	3	29	17	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	1	0	21	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	1	12	18	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	2	71	18	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	0	2	22	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	10	7	2	1	3	8	nominal execution time (sec)
PFS	1	0	21	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	0	2	22	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	7	7	8	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	2	0	18	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	1	3	20	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	2	1	19	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	0	19	21	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	2	1	19	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	9	83	4	15	33	181	nominal backend bound (CPU)
UBM	8	77	5	5	37	109	nominal bad speculation: mispredicts
UBP	3	701	16	87	977	3209	nominal bad speculation: pipeline restarts
UBS	8	78	5	6	38	112	nominal bad speculation
UDC	5	12	13	2	13	27	nominal data cache misses per K instructions
UDT	4	200	15	13	289	1101	nominal DTLB misses per M instructions
UIP	7	202	7	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	2	1557	19	318	3001	8545	nominal LLC misses M instructions
USB	2	25	18	6	29	53	nominal 100 x back end bound
USC	0	1	22	1	53	348	nominal SMT contention
USF	7	32	8	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 21. Lower bounds on the overheads [2] for jme for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

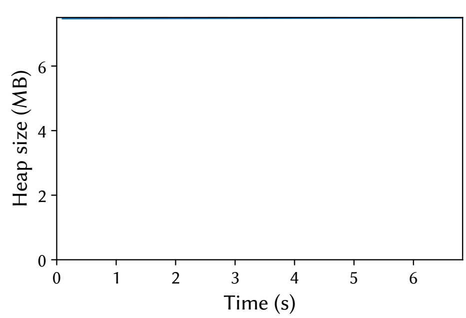
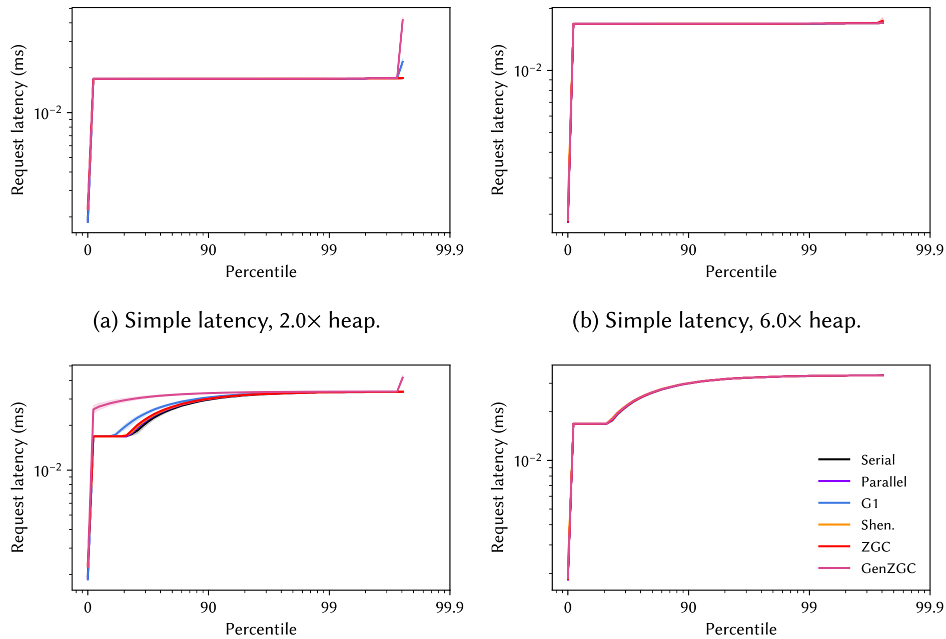


Fig. 22. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

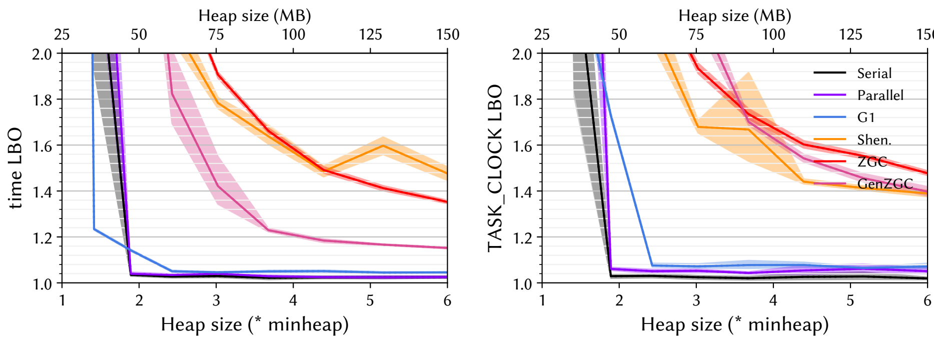
Fig. 23. Distribution of request latencies for jme for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

11 JYTHON

The jython workload executes a standard Python performance test on top of Jython, a Java implementation of the Python programming language. Jython has about 310 K lines of Java code. It has the most unique function calls executed (BUF) and a large number of unique bytecodes executed (BUB). Consistent with this, it has the longest warmup (PWU) and is sensitive to compiler configuration (PCS). It has very high IPC (UIP) and high bad speculation due to mispredicts (UBS), UBM).

Table 13. Complete nominal statistics for jython. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	2	37	17	28	58	210	nominal average object size (bytes)
AOL	3	48	15	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	2	16	16	16	24	24	nominal 10-percentile object size (bytes)
ARA	4	1476	13	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	5	39	9	0	33	2305	nominal aaload per usec
BAS	8	13	4	0	1	87	nominal aastore per usec
BEF	8	8	5	1	4	28	nominal execution focus / dominance of hot code
BGF	3	259	14	26	507	33553	nominal getfield per usec
BPF	5	84	10	2	84	3346	nominal putfield per usec
BUB	8	149	4	8	35	177	nominal thousands of unique bytecodes executed
BUF	10	29	1	1	4	29	nominal thousands of unique function calls
GCA	4	92	14	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	8	3047	6	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	5	89	13	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	6	8	10	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	8	12	5	0	0	98	nominal percent 10th iteration memory leakage
GMD	3	25	17	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	2	25	16	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	6	25	10	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	4	31	14	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	9	1421	3	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	7	138	7	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	6	231	9	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	9	190	3	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	9	17	4	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	9	190	3	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	2	1	19	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	2	0	18	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	5	6	13	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	10	9	1	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	8	995	6	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	5	21	13	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	4	26	15	15	33	181	nominal backend bound (CPU)
UBM	9	83	4	5	37	109	nominal bad speculation: mispredicts
UBP	6	1107	10	87	977	3209	nominal bad speculation: pipeline restarts
UBS	9	85	4	6	38	112	nominal bad speculation
UDC	4	8	15	2	13	27	nominal data cache misses per K instructions
UDT	3	101	17	13	289	1101	nominal DTLB misses per M instructions
UIP	10	276	2	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	1	1394	20	318	3001	8545	nominal LLC misses M instructions
USB	3	27	16	6	29	53	nominal 100 x back end bound
USC	2	11	18	1	53	348	nominal SMT contention
USF	4	20	14	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 24. Lower bounds on the overheads [2] for jython for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

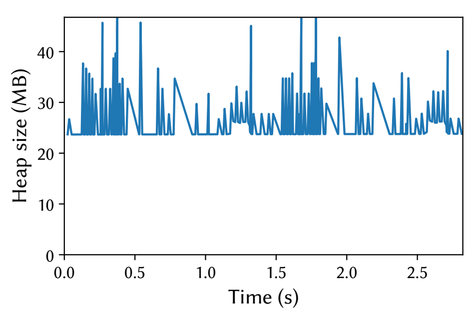


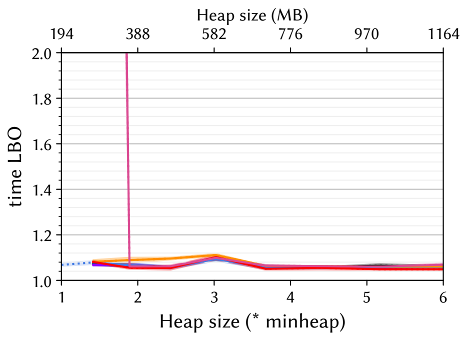
Fig. 25. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

12 KAFKA

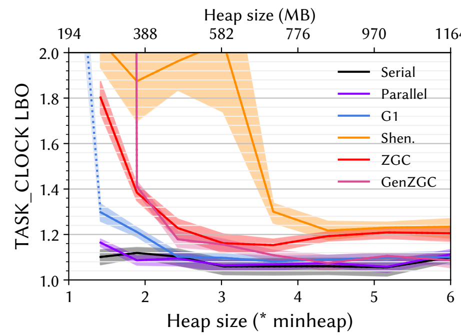
(**New**) This is a latency-sensitive workload that issues requests to the Apache Kafka framework for high-throughput publish-subscribe messaging. Kafka has about 840 K lines of Java and Scala source code. kafka has low garbage collection sensitivity (GSS), GCP). It is kernel-intensive (PKP) and insensitive to CPU frequency scaling (PFS) and memory speed (PMS). It has a very high data cache and last level cache miss rates (UDC), ULL) and is one of the most front end bound workloads (USF).

Table 14. Complete nominal statistics for kafka. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	4	55	12	28	58	210	nominal average object size (bytes)
AOL	5	56	11	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	2	16	16	16	24	24	nominal 10-percentile object size (bytes)
ARA	2	801	17	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	2	1	16	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	2	2	15	1	4	28	nominal execution focus / dominance of hot code
BGF	2	162	16	26	507	33553	nominal getfield per usec
BPF	3	49	13	2	84	3346	nominal putfield per usec
BUB	9	161	3	8	35	177	nominal thousands of unique bytecodes executed
BUF	9	28	2	1	4	29	nominal thousands of unique function calls
GCA	3	87	17	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	2	247	19	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	4	87	14	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	1	0	21	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	9	191	3	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	7	305	6	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	10	157	1	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	9	203	4	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	1	0	21	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	2	19	17	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	7	266	7	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	2	16	18	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	9	6	3	1	3	8	nominal execution time (sec)
PFS	1	0	21	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	2	16	18	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	10	31	2	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	2	0	18	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	1	3	20	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	7	5	7	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	1	474	19	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	3	14	17	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	3	25	17	15	33	181	nominal backend bound (CPU)
UBM	3	31	17	5	37	109	nominal bad speculation: mispredicts
UBP	1	421	20	87	977	3209	nominal bad speculation: pipeline restarts
UBS	3	31	17	6	38	112	nominal bad speculation
UDC	10	27	1	2	13	27	nominal data cache misses per K instructions
UDT	6	419	9	13	289	1101	nominal DTLB misses per M instructions
UIP	4	117	15	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	10	8545	1	318	3001	8545	nominal LLC misses M instructions
USB	3	26	17	6	29	53	nominal 100 x back end bound
USC	3	15	17	1	53	348	nominal SMT contention
USF	10	48	2	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 26. Lower bounds on the overheads [2] for kafka for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

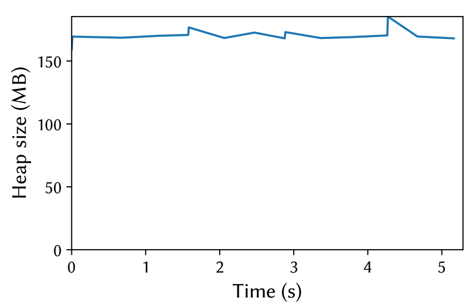
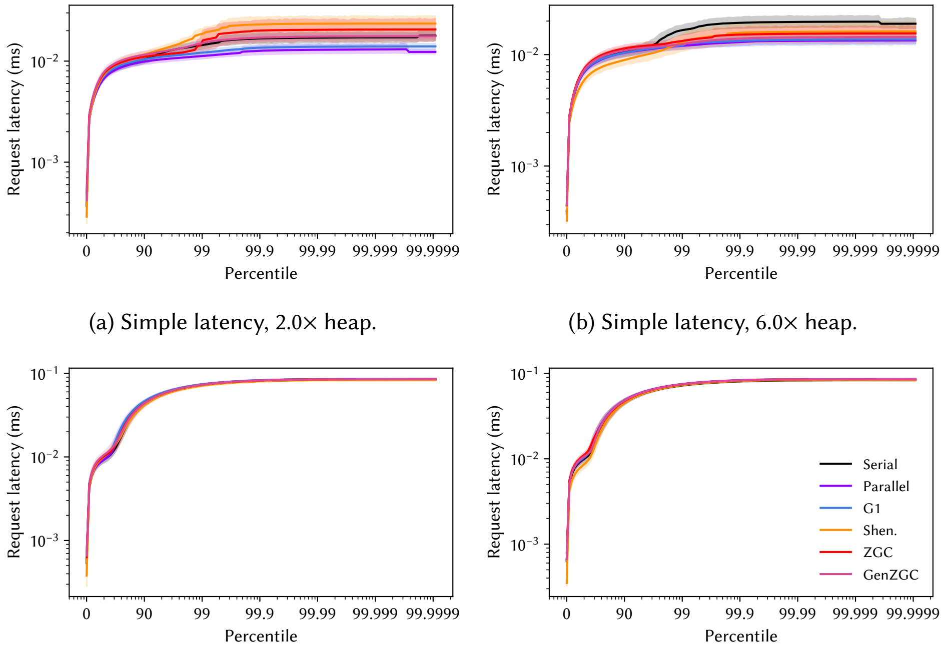


Fig. 27. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

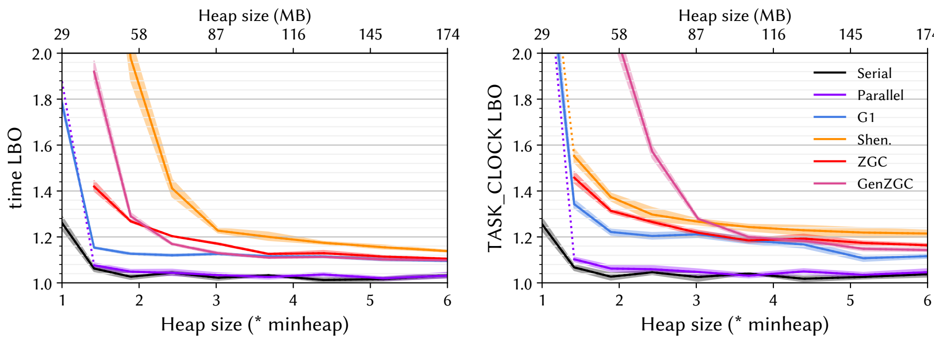
Fig. 28. Distribution of request latencies for kafka for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

13 LUINDEX

This workload constructs a search index from a document corpus using the Apache Lucene search engine. Lucene has about 830 K lines of Java source code. luindex has the largest objects in the suite (AOA), AOL), AOM), AOS). It is the most sensitive to CPU frequency scaling (PFS) and last level cache size (PLS). It has one of the highest IPCs (UIP) but suffers one of the worst bad speculation rates (UBS), UBM), UBP), but has low cache miss rates (UDC), UDT), ULL).

Table 15. Complete nominal statistics for luindex. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	10	210	1	28	58	210	nominal average object size (bytes)
AOL	8	88	4	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	2	835	16	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	5	33	10	0	33	2305	nominal aaload per usec
BAS	5	1	9	0	1	87	nominal aastore per usec
BEF	3	3	13	1	4	28	nominal execution focus / dominance of hot code
BGF	6	1176	8	26	507	33553	nominal getfield per usec
BPF	7	305	6	2	84	3346	nominal putfield per usec
BUB	5	54	9	8	35	177	nominal thousands of unique bytecodes executed
BUF	6	5	8	1	4	29	nominal thousands of unique function calls
GCA	4	89	15	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	6	1412	9	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	5	96	11	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	4	2	14	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	4	29	14	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	3	37	13	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	5	13	12	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	4	31	14	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	5	64	12	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	5	75	10	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	5	200	12	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	5	60	13	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	10	19	1	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	5	60	13	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	4	2	14	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	10	36	1	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	3	1	16	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	4	5	15	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	2	1	19	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	2	553	17	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	5	22	11	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	7	44	8	15	33	181	nominal backend bound (CPU)
UBM	10	109	1	5	37	109	nominal bad speculation: mispredicts
UBP	10	3161	2	87	977	3209	nominal bad speculation: pipeline restarts
UBS	10	112	1	6	38	112	nominal bad speculation
UDC	2	6	18	2	13	27	nominal data cache misses per K instructions
UDT	2	85	18	13	289	1101	nominal DTLB misses per M instructions
UIP	9	263	3	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	1	985	21	318	3001	8545	nominal LLC misses M instructions
USB	7	36	7	6	29	53	nominal 100 x back end bound
USC	1	4	20	1	53	348	nominal SMT contention
USF	2	12	18	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 29. Lower bounds on the overheads [2] for luindex for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

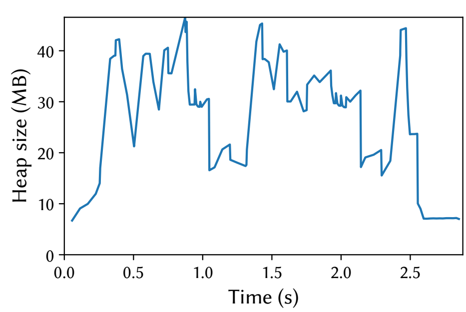


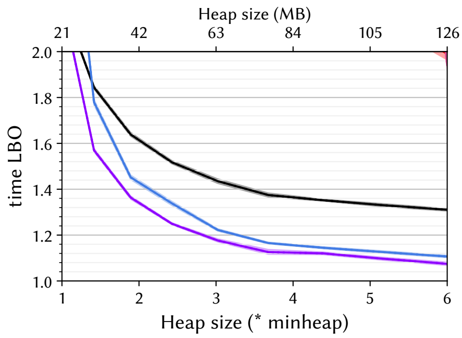
Fig. 30. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

14 LUSEARCH

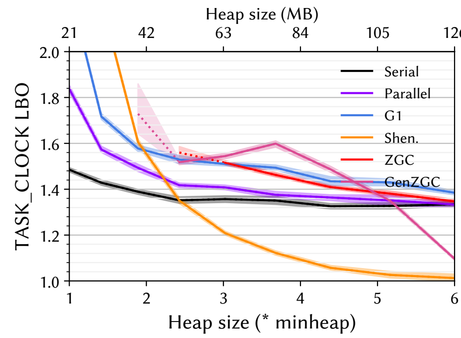
This is a latency-sensitive workload that issues search requests to the Apache Lucene search engine. lusearch has the highest memory turn over (GTO) one of the highest allocation rates (ARA), one of the most generational ((GCM), one of the highest aastore and putfield rates (BAS), BPF) and one of the tightest execution focuses (BEF). It uses a very small heap (GCA), GMD).

Table 16. Complete nominal statistics for lusearch. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	7	75	7	28	58	210	nominal average object size (bytes)
AOL	8	88	4	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	8	10371	4	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	7	110	6	0	33	2305	nominal aaload per usec
BAS	9	55	2	0	1	87	nominal aastore per usec
BEF	7	5	7	1	4	28	nominal execution focus / dominance of hot code
BGF	8	5378	5	26	507	33553	nominal getfield per usec
BPF	9	1689	2	2	84	3346	nominal putfield per usec
BUB	3	26	13	8	35	177	nominal thousands of unique bytecodes executed
BUF	3	3	13	1	4	29	nominal thousands of unique function calls
GCA	2	80	19	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	10	17270	1	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	1	66	20	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	9	19	3	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	2	21	18	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	4	89	11	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	2	5	18	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	2	21	19	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	6	185	10	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	10	1103	1	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	2	70	19	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	7	100	7	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	5	12	11	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	7	100	7	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	8	16	5	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	7	12	7	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	8	7	6	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	8	31	5	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	2	551	18	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	10	41	1	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	8	74	6	15	33	181	nominal backend bound (CPU)
UBM	4	33	14	5	37	109	nominal bad speculation: mispredicts
UBP	4	838	14	87	977	3209	nominal bad speculation: pipeline restarts
UBS	4	33	15	6	38	112	nominal bad speculation
UDC	5	13	11	2	13	27	nominal data cache misses per K instructions
UDT	6	391	10	13	289	1101	nominal DTLB misses per M instructions
UIP	4	118	14	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	6	3285	9	318	3001	8545	nominal LLC misses M instructions
USB	7	36	7	6	29	53	nominal 100 x back end bound
USC	8	136	6	1	53	348	nominal SMT contention
USF	5	27	12	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 31. Lower bounds on the overheads [2] for lusearch for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

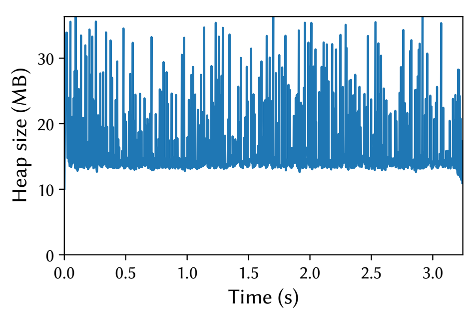
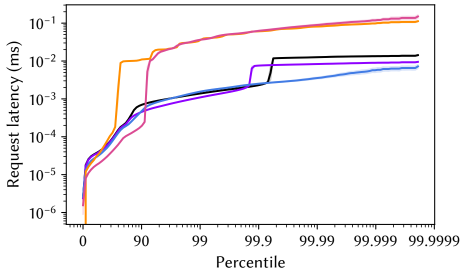
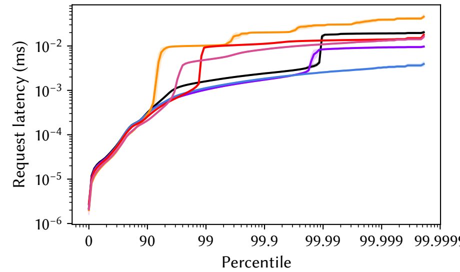


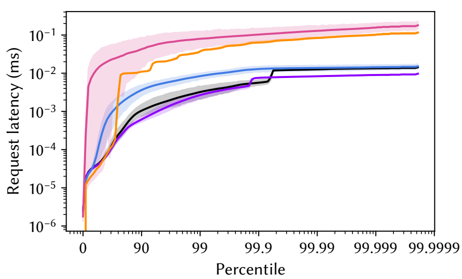
Fig. 32. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



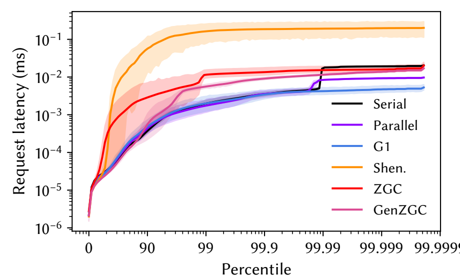
(a) Simple latency, 2.0× heap.



(b) Simple latency, 6.0× heap.



(c) Metered latency, 2.0× heap.



(d) Metered latency, 6.0× heap.

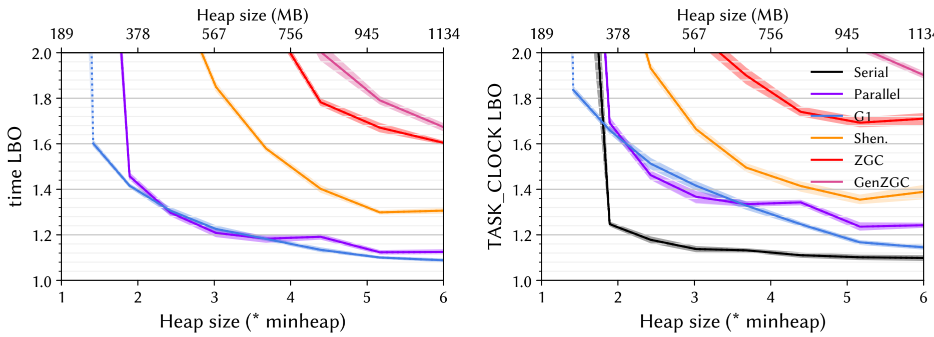
Fig. 33. Distribution of request latencies for lusearch for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

15 PMD

This workload uses the PMD static code analyzer to check a corpus of source code. PMD has about 120 K lines of Java code. pmd is the most last level cache size-sensitive workload (PLS) and is sensitive to memory speed (PMS). It is one of the least generational workloads (GCM), and is one of the slowest to warm up (PWU). It is among the most back end bound (USB), with high SMT contention (USC) and high last level cache miss rate (ULL).

Table 17. Complete nominal statistics for pmd. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	1	32	19	28	58	210	nominal average object size (bytes)
AOL	3	48	15	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	2	16	16	16	24	24	nominal 10-percentile object size (bytes)
ARA	8	6819	5	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	7	83	7	0	33	2305	nominal aaload per usec
BAS	5	1	9	0	1	87	nominal aastore per usec
BEF	5	4	9	1	4	28	nominal execution focus / dominance of hot code
BGF	7	1743	7	26	507	33553	nominal getfield per usec
BPF	8	592	4	2	84	3346	nominal putfield per usec
BUB	7	95	7	8	35	177	nominal thousands of unique bytecodes executed
BUF	7	15	7	1	4	29	nominal thousands of unique function calls
GCA	10	136	1	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	3	788	16	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	10	150	1	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	9	18	4	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	6	4	9	0	0	98	nominal percent 10th iteration memory leakage
GMD	9	189	4	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GMS	3	7	16	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	10	272	2	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	8	508	6	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	3	32	15	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	5	179	13	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	6	75	9	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	2	1	19	1	3	8	nominal execution time (sec)
PFS	5	12	11	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	6	75	9	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	4	2	14	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	10	25	2	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	9	16	4	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	7	11	8	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	9	2	3	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	9	7	4	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	8	1014	5	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	9	31	4	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	7	45	7	15	33	181	nominal backend bound (CPU)
UBM	5	37	12	5	37	109	nominal bad speculation: mispredicts
UBP	7	1229	8	87	977	3209	nominal bad speculation: pipeline restarts
UBS	5	38	12	6	38	112	nominal bad speculation
UDC	7	16	7	2	13	27	nominal data cache misses per K instructions
UDT	5	268	13	13	289	1101	nominal DTLB misses per M instructions
UIP	3	110	17	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	8	4435	6	318	3001	8545	nominal LLC misses M instructions
USB	8	41	5	6	29	53	nominal 100 x back end bound
USC	9	155	4	1	53	348	nominal SMT contention
USF	5	21	13	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 34. Lower bounds on the overheads [2] for pmd for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

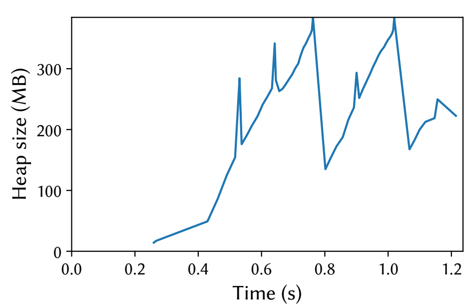


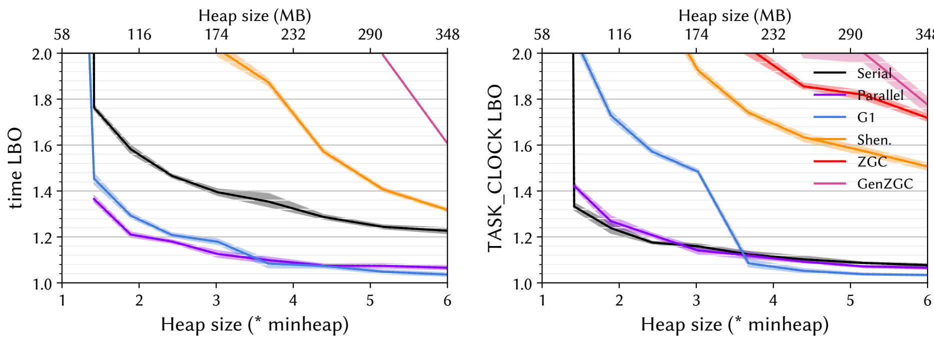
Fig. 35. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

16 SPRING

(**New**) This is a latency-sensitive workload that runs the petclinic workload over the Spring Boot microservices web framework. DaCapo Chopin replaces petclinc's synthetic load generator with a deterministic request workload. Spring Boot has about 580 K lines of Java source code. spring is the workload most sensitive to memory speed (PMS). It has one of the highest number of unique bytecodess executed (BUB) and unique function calls (BUF) and is sensitive to choice of compiler (PCS).

Table 18. Complete nominal statistics for spring. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	6	74	9	28	58	210	nominal average object size (bytes)
AOL	10	216	1	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	10	12243	1	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	3	12	13	0	33	2305	nominal aaload per usec
BAS	7	2	7	0	1	87	nominal aastore per usec
BEF	2	2	15	1	4	28	nominal execution focus / dominance of hot code
BGF	4	424	11	26	507	33553	nominal getfield per usec
BPF	6	101	8	2	84	3346	nominal putfield per usec
BUB	9	175	2	8	35	177	nominal thousands of unique bytecodes executed
BUF	9	27	3	1	4	29	nominal thousands of unique function calls
GCA	5	96	13	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	7	2462	7	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	3	81	17	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	8	16	6	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	7	5	7	0	0	98	nominal percent 10th iteration memory leakage
GMD	5	58	13	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	4	72	12	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	7	43	7	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	5	67	13	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	5	118	11	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	8	312	4	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	4	137	14	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	8	119	5	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	4	2	15	1	3	8	nominal execution time (sec)
PFS	4	9	15	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	8	119	5	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	7	10	7	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	8	16	5	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	10	19	2	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	9	32	3	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	7	4	8	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	3	588	16	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	9	31	4	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	4	29	14	15	33	181	nominal backend bound (CPU)
UBM	7	59	7	5	37	109	nominal bad speculation: mispredicts
UBP	8	1424	6	87	977	3209	nominal bad speculation: pipeline restarts
UBS	7	61	7	6	38	112	nominal bad speculation
UDC	5	13	11	2	13	27	nominal data cache misses per K instructions
UDT	7	463	8	13	289	1101	nominal DTLB misses per M instructions
UIP	5	119	13	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	7	4352	7	318	3001	8545	nominal LLC misses M instructions
USB	6	31	10	6	29	53	nominal 100 x back end bound
USC	7	101	8	1	53	348	nominal SMT contention
USF	7	32	8	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 36. Lower bounds on the overheads [2] for spring for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

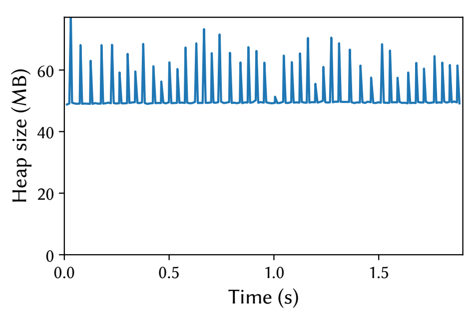
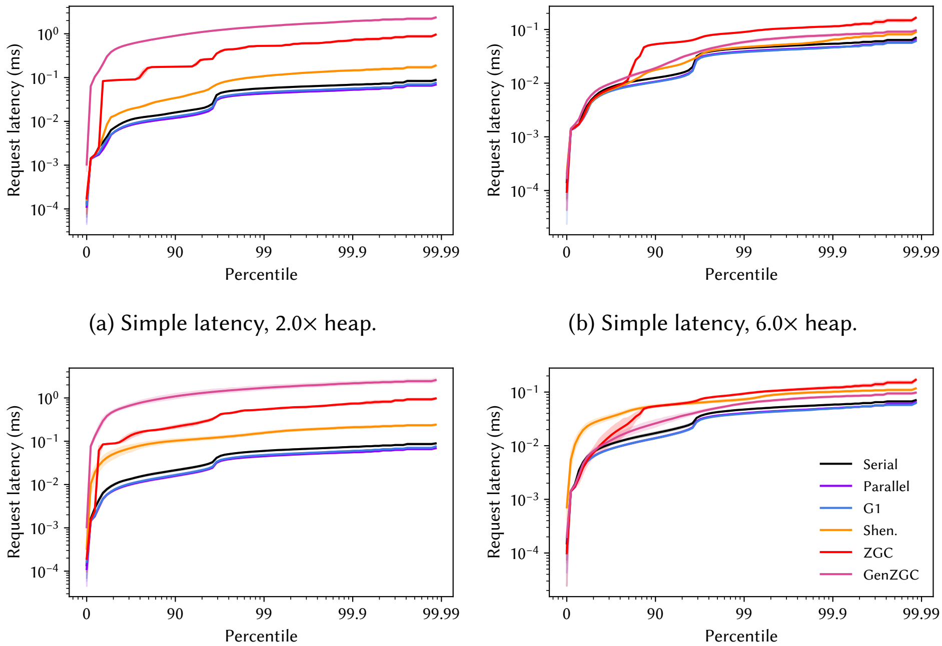


Fig. 37. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

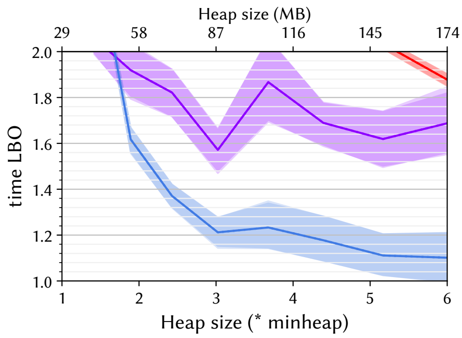
Fig. 38. Distribution of request latencies for spring for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

17 SUNFLOW

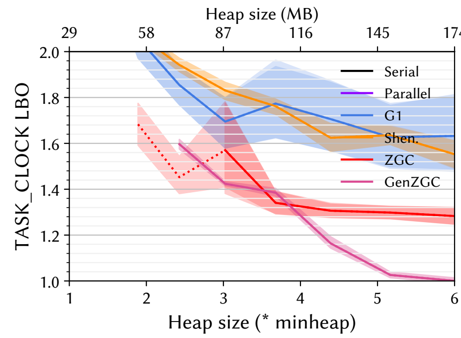
This workload uses the Sunflow photorealistic renderer to render a series of images. Sunflow consists of about 25 K lines of Java code. sunflow has a high allocation rate (ARA), and the highest aaload, getfield, and putfield rates (BAL), BGF), BPF). It is the slowest to warm up (PWU) and has the highest execution variance (PSD). It is the least sensitive to last level cache size (PLS). It is one of the least front end bound (USF) and one of the most back end bound (USB) and suffers high SMT contention (USC).

Table 19. Complete nominal statistics for sunflow. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	3	40	15	28	58	210	nominal average object size (bytes)
AOL	3	48	15	24	56	216	nominal 90-percentile object size (bytes)
AOM	10	48	1	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	9	10999	3	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	10	2305	1	0	33	2305	nominal aaload per usec
BAS	7	3	6	0	1	87	nominal aastore per usec
BEF	3	3	13	1	4	28	nominal execution focus / dominance of hot code
BGF	10	33553	1	26	507	33553	nominal getfield per usec
BPF	10	3346	1	2	84	3346	nominal putfield per usec
BUB	2	20	15	8	35	177	nominal thousands of unique bytecodes executed
BUF	1	1	17	1	4	29	nominal thousands of unique function calls
GCA	7	109	8	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	9	10937	3	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	6	100	9	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	9	18	4	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	9	13	4	0	0	98	nominal percent 10th iteration memory leakage
GMD	4	29	14	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	6	149	8	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	2	5	18	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	4	31	14	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	8	710	5	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	9	711	3	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	4	90	15	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	2	13	19	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	4	2	15	1	3	8	nominal execution time (sec)
PFS	5	10	13	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	2	13	19	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	4	2	14	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	0	-1	22	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	5	2	13	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	9	32	3	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	10	13	1	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	10	9	1	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	6	808	9	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	3	15	16	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	10	121	2	15	33	181	nominal backend bound (CPU)
UBM	1	23	20	5	37	109	nominal bad speculation: mispredicts
UBP	9	2566	3	87	977	3209	nominal bad speculation: pipeline restarts
UBS	1	25	20	6	38	112	nominal bad speculation
UDC	4	8	15	2	13	27	nominal data cache misses per K instructions
UDT	3	103	16	13	289	1101	nominal DTLB misses per M instructions
UIP	3	116	16	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	4	2217	15	318	3001	8545	nominal LLC misses M instructions
USB	9	48	3	6	29	53	nominal 100 x back end bound
USC	10	245	2	1	53	348	nominal SMT contention
USF	1	5	21	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 39. Lower bounds on the overheads [2] for sunflow for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

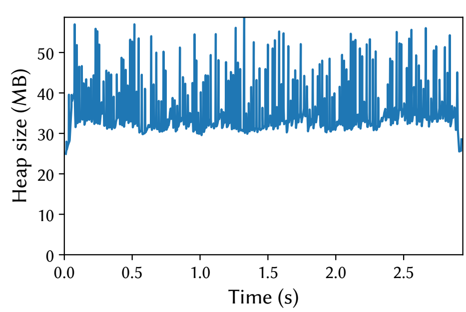


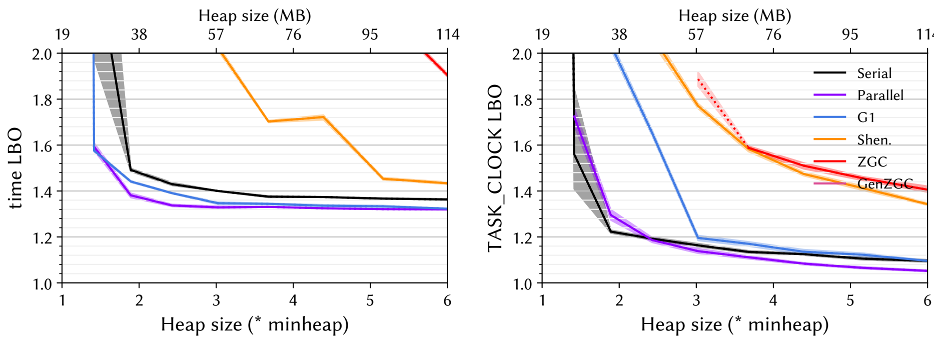
Fig. 40. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

18 TOMCAT

This is a latency-sensitive workload that issues requests to the Apache Tomcat web server. Tomcat consists of about 380 K lines of Java code. Tomcat has the highest parallel efficiency (PPE). It is the most sensitive to heap size (GSS) and has a high GC turnover (GTO) and GC count (GCC). It spends a relatively large amount of time in the kernel (PKP), which is unsurprising for a web server. It has one of the highest data cache, last level cache, and DTLB miss rates (UDC), ULL), UDT) and one of the lowest IPCs (UIP).

Table 20. Complete nominal statistics for tomcat. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	7	75	7	28	58	210	nominal average object size (bytes)
AOL	7	80	7	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	7	5227	7	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	2	9	15	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	8	9	4	1	4	28	nominal execution focus / dominance of hot code
BGF	3	268	13	26	507	33553	nominal getfield per usec
BPF	4	74	11	2	84	3346	nominal putfield per usec
BUB	7	118	6	8	35	177	nominal thousands of unique bytecodes executed
BUF	7	17	6	1	4	29	nominal thousands of unique function calls
GCA	7	110	7	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	9	7676	4	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	7	105	8	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	7	10	8	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	2	19	19	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	3	31	14	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	5	13	12	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	2	23	18	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	10	1916	2	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	9	896	2	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	10	465	2	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	5	64	11	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	7	4	7	1	3	8	nominal execution time (sec)
PFS	2	2	19	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	5	64	11	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	9	21	4	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	5	3	13	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	5	2	13	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	10	91	1	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	5	0	13	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	1	71	20	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	2	6	18	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	5	30	13	15	33	181	nominal backend bound (CPU)
UBM	6	43	10	5	37	109	nominal bad speculation: mispredicts
UBP	2	613	18	87	977	3209	nominal bad speculation: pipeline restarts
UBS	6	44	10	6	38	112	nominal bad speculation
UDC	8	17	6	2	13	27	nominal data cache misses per K instructions
UDT	9	631	4	13	289	1101	nominal DTLB misses per M instructions
UIP	1	103	20	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	9	4821	4	318	3001	8545	nominal LLC misses M instructions
USB	2	24	19	6	29	53	nominal 100 x back end bound
USC	6	88	10	1	53	348	nominal SMT contention
USF	9	45	3	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 41. Lower bounds on the overheads [2] for tomcat for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

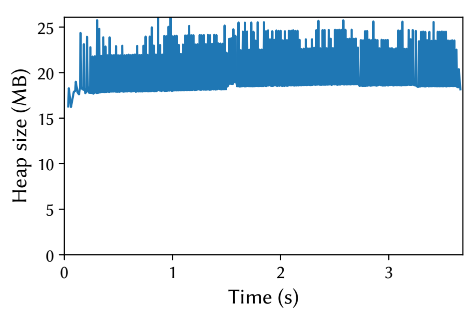
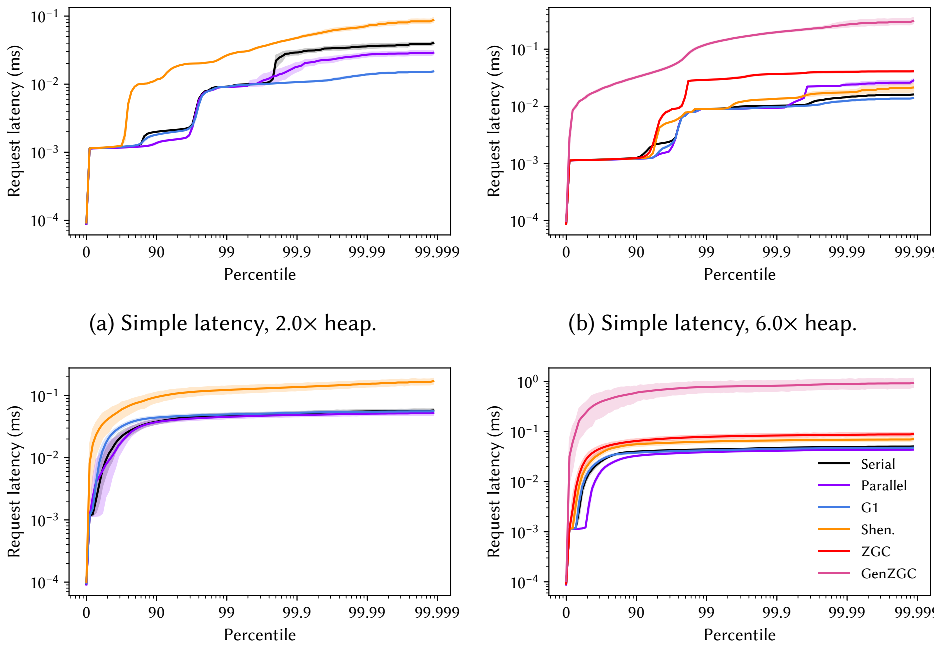


Fig. 42. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

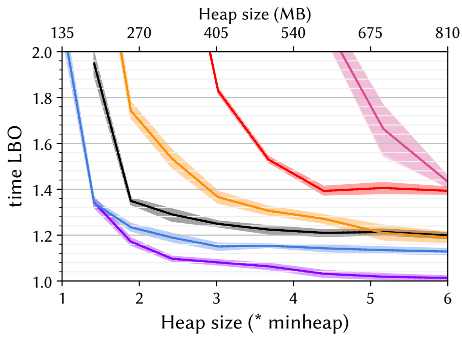
Fig. 43. Distribution of request latencies for tomcat for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

19 TRADEBEANS

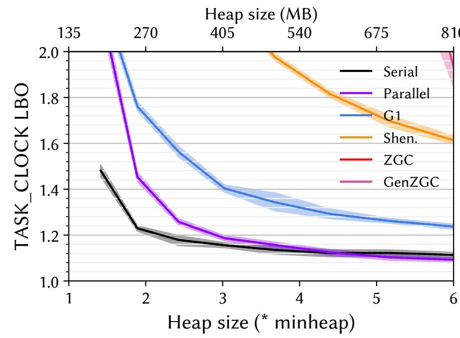
This is a latency-sensitive workload that executes the DayTrader workload over the Wildfly application server. Wildfly has about 4.2 M lines of Java source code. The DayTrader workload was originally developed by IBM Research to model customer applications on their production application server [4, 5]. DaCapo Chopin replaces the DayTrader synthetic load generator with a deterministic load. tradebeans is sensitive to compiler configuration (PCC) and memory speed (PMS). It is slow to warm up (PWU) and has high variance (PSD). It has a minimum heap size of 1.1 GB in its vlarge configuration. It is one of the least back end bound workloads (UBC).

Table 21. Complete nominal statistics for tradebeans. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
GCA	4	89	15	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	5	965	12	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	4	87	14	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	5	3	12	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	10	26	2	0	0	98	nominal percent 10th iteration memory leakage
GMD	7	135	7	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	8	603	5	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	8	73	5	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	7	141	8	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GMV	4	1125	2	369	1125	20609	nominal minimum heap size (MB) for vlarge size configuration (with compressed pointers)
GSS	5	55	13	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
PCC	3	83	16	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	9	161	4	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	7	4	7	1	3	8	nominal execution time (sec)
PFS	6	13	10	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	9	161	4	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	5	3	13	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	5	4	12	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	8	7	6	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	4	5	15	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	9	2	3	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	9	7	4	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	10	1452	1	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	9	31	4	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	0	15	22	15	33	181	nominal backend bound (CPU)
UBM	5	38	11	5	37	109	nominal bad speculation: mispredicts
UBP	7	1330	7	87	977	3209	nominal bad speculation: pipeline restarts
UBS	5	40	11	6	38	112	nominal bad speculation
UDC	4	8	15	2	13	27	nominal data cache misses per K instructions
UDT	5	383	11	13	289	1101	nominal DTLB misses per M instructions
UIP	7	187	8	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	5	3001	12	318	3001	8545	nominal LLC misses M instructions
USB	5	28	13	6	29	53	nominal 100 x back end bound
USC	5	49	13	1	53	348	nominal SMT contention
USF	7	34	7	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 44. Lower bounds on the overheads [2] for tradebeans for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

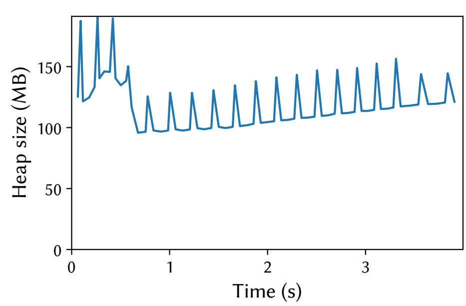
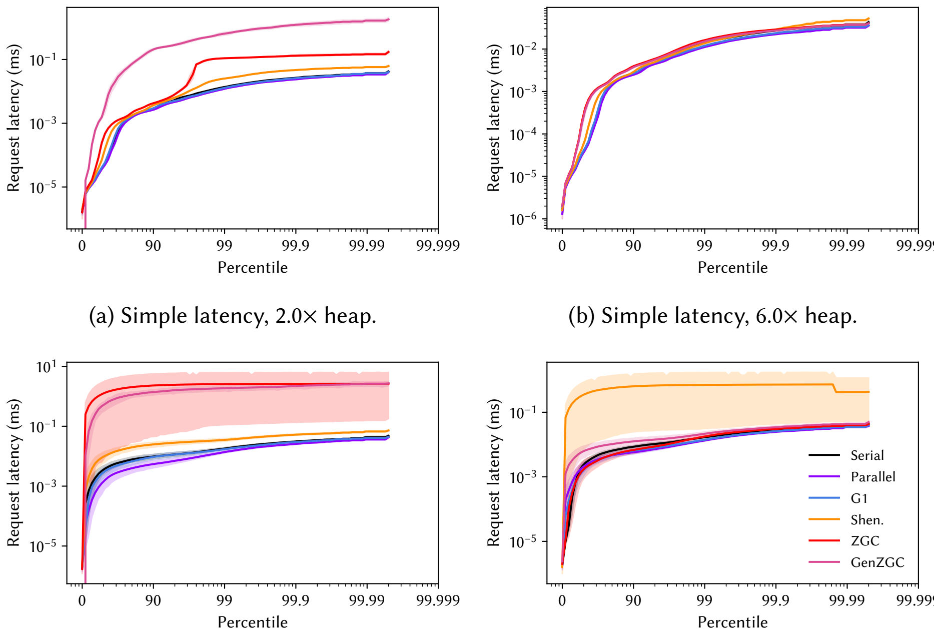


Fig. 45. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

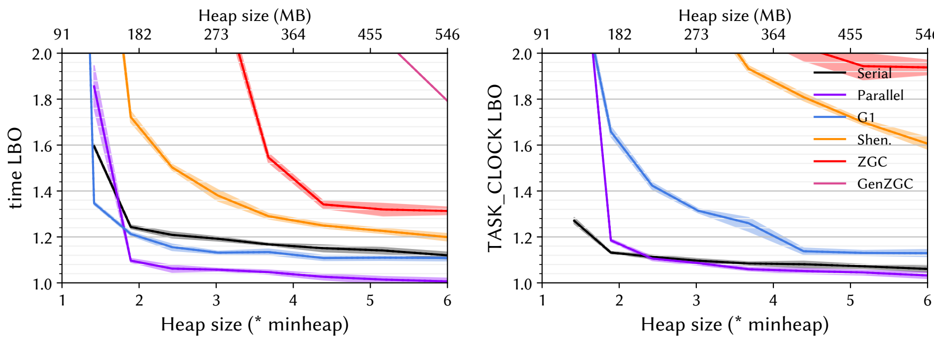
Fig. 46. Distribution of request latencies for tradebeans for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

20 TRADESOAP

This is a latency-sensitive workload that executes the DayTrader workload over the Wildfly application server. It differs from tradebeans in that it uses the full SOAP protocol to communicate with the server. DaCapo includes the two variants of the DayTrader workload on the recommendation of the authors of the original work that pointed to the inefficiencys of such web frameworks [6]. It is sensitive to CPU frequency scaling (PFS) and last level cache size (PLS). It has one of the highest DTLB miss rates (UDT), but is one of the least back end bound (UBC).

Table 22. Complete nominal statistics for tradesoap. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
GCA	6	104	9	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	4	818	15	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	6	100	9	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	5	3	12	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	8	6	6	0	0	98	nominal percent 10th iteration memory leakage
GMD	5	91	11	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	7	229	7	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	9	75	4	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	6	115	10	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GMV	0	369	3	369	1125	20609	nominal minimum heap size (MB) for vlarge size configuration (with compressed pointers)
GSS	7	299	8	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
PCC	9	461	3	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	7	90	8	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	6	3	9	1	3	8	nominal execution time (sec)
PFS	9	17	4	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	7	90	8	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	5	5	11	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	9	17	3	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	6	6	9	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	5	7	12	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	5	2	13	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	10	1239	2	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	7	30	7	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	1	19	21	15	33	181	nominal backend bound (CPU)
UBM	8	67	6	5	37	109	nominal bad speculation: mispredicts
UBP	5	977	12	87	977	3209	nominal bad speculation: pipeline restarts
UBS	8	68	6	6	38	112	nominal bad speculation
UDC	8	18	5	2	13	27	nominal data cache misses per K instructions
UDT	10	1101	1	13	289	1101	nominal DTLB misses per M instructions
UIP	5	140	11	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	6	3135	10	318	3001	8545	nominal LLC misses M instructions
USB	5	28	13	6	29	53	nominal 100 x back end bound
USC	5	53	12	1	53	348	nominal SMT contention
USF	8	37	6	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 47. Lower bounds on the overheads [2] for tradesoap for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

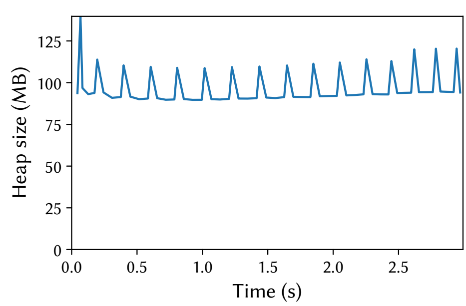
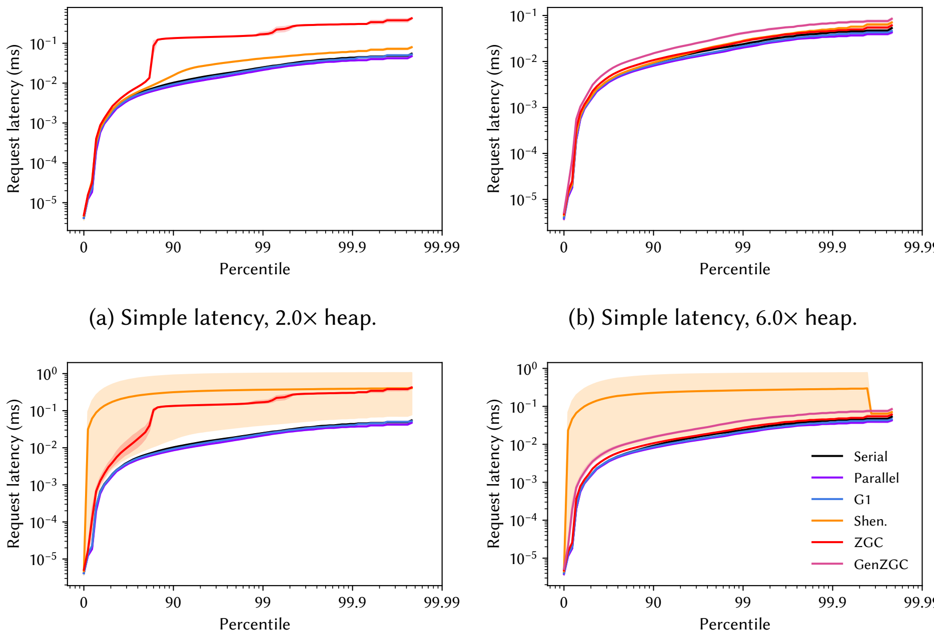


Fig. 48. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.



(a) Simple latency, 2.0× heap.

(b) Simple latency, 6.0× heap.

(c) Metered latency, 2.0× heap.

(d) Metered latency, 6.0× heap.

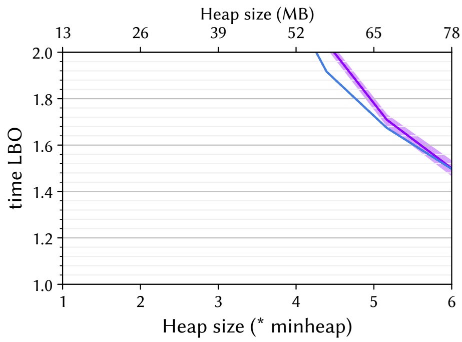
Fig. 49. Distribution of request latencies for tradesoap for each of OpenJDK 21's six production collectors. The figures in the left top row simply plot the request latencies, while the figures in the bottom row use DaCapo's metered latency, which models a request queue and the cascading effect of delays.

21 XALAN

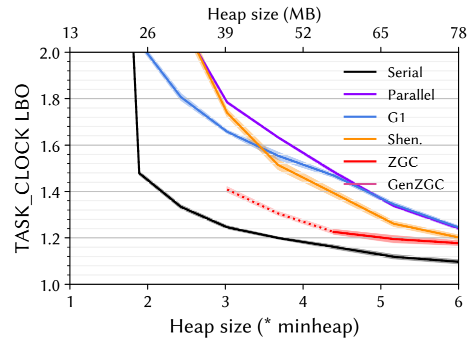
This workload uses the Apache Xalan XSLT processor to transform a set of documents. xalan is the workload most sensitive to heap size (GSS), GCA), GCC), GCM), GCP), GTO). It has the highest allocation rate (ARA), the highest aastore rate (BAS) and very high aaload, putfield, and getfield rates (BAL), BPF), BGF). It is very insensitive to compiler configuration (PCC), PCS). It has one of the worst data cache miss rates (UDC), and one of the lowest IPCs (UIP).

Table 23. Complete nominal statistics for xalan. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	8	107	5	28	58	210	nominal average object size (bytes)
AOL	3	48	15	24	56	216	nominal 90-percentile object size (bytes)
AOM	4	24	13	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	7	6493	6	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	9	285	3	0	33	2305	nominal aaload per usec
BAS	10	87	1	0	1	87	nominal aastore per usec
BEF	5	4	9	1	4	28	nominal execution focus / dominance of hot code
BGF	9	6198	3	26	507	33553	nominal getfield per usec
BPF	9	783	3	2	84	3346	nominal putfield per usec
BUB	3	21	14	8	35	177	nominal thousands of unique bytecodes executed
BUF	3	2	14	1	4	29	nominal thousands of unique function calls
GCA	10	123	2	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	10	14338	2	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	10	130	2	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	10	77	1	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	5	0	12	0	0	98	nominal percent 10th iteration memory leakage
GMD	1	13	20	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GML	0	13	18	13	135	10193	nominal minimum heap size (MB) for large size configuration (with compressed pointers)
GMS	2	5	18	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	1	17	20	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	10	7778	1	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	8	286	5	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	0	-1	22	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	1	11	20	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	2	1	19	1	3	8	nominal execution time (sec)
PFS	3	8	16	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	1	11	20	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	9	26	3	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	6	6	9	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	2	0	18	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	6	10	9	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	2	1	19	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	6	763	10	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	1	-3	20	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	6	40	9	15	33	181	nominal backend bound (CPU)
UBM	5	35	13	5	37	109	nominal bad speculation: mispredicts
UBP	2	586	19	87	977	3209	nominal bad speculation: pipeline restarts
UBS	5	35	13	6	38	112	nominal bad speculation
UDC	9	21	4	2	13	27	nominal data cache misses per K instructions
UDT	7	485	7	13	289	1101	nominal DTLB misses per M instructions
UIP	1	94	21	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	8	4702	5	318	3001	8545	nominal LLC misses M instructions
USB	6	32	9	6	29	53	nominal 100 x back end bound
USC	6	99	9	1	53	348	nominal SMT contention
USF	8	39	5	4	27	61	nominal 100 x front end bound



(a) Wall clock.



(b) Total CPU overheads (TASK_CLOCK).

Fig. 50. Lower bounds on the overheads [2] for xalan for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

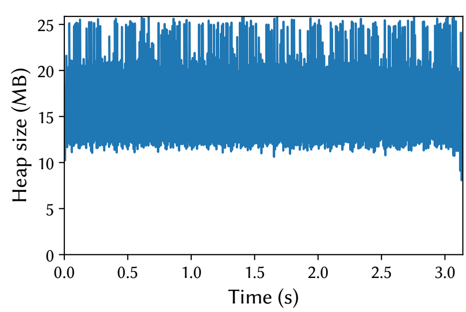


Fig. 51. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

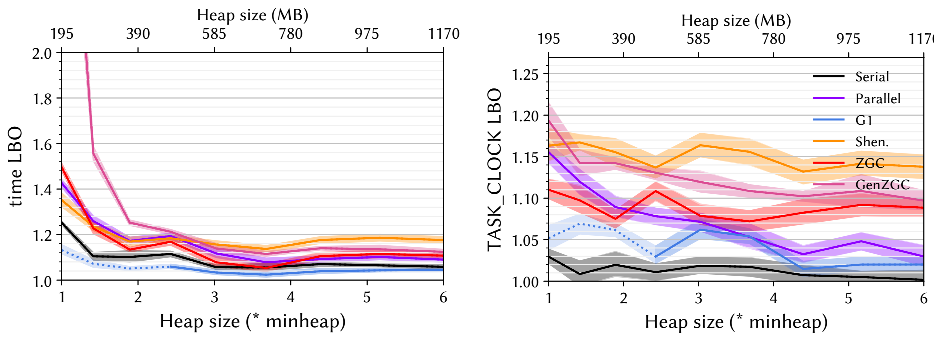
22 ZXING

(**New**) This workload uses the ZXing[1] barcode reader to read a series of 1D and 2D barcodes. ZXing has about 48 K lines of Java source code. zxing has the highest parallel efficiency among the workloads (PPE). It is one of the slowest to warm up (PWU), has one of the largest average and median object sizes (AOA), AOM) and is one of the least sensitive to garbage collection (GCA), GCC), GCM), GCP). It is among the least back end bound (USB), and has among the lowest data cache, last level cache, and DTLB miss rates (UDC), ULL), UDT).

[1] Pronounced *zebra crossing*.

Table 24. Complete nominal statistics for zxing. Value represents the concrete value for that metric with respect to Description. Min, Median, and Max are the summary statistics for that metric across all benchmarks. For each metric, the benchmark obtains a Score between 0 and 10 (10 being the largest concrete value for that metric). Similarly, the benchmark obtains a Rank between 1 and the number of benchmarks having that metric (1 being the largest).

Metric	Score	Value	Rank	Min	Median	Max	Description
AOA	9	115	3	28	58	210	nominal average object size (bytes)
AOL	7	80	7	24	56	216	nominal 90-percentile object size (bytes)
AOM	9	32	2	24	32	48	nominal median object size (bytes)
AOS	10	24	1	16	24	24	nominal 10-percentile object size (bytes)
ARA	5	2063	11	41	2063	12243	nominal allocation rate (bytes / usec)
BAL	8	127	5	0	33	2305	nominal aaload per usec
BAS	4	0	12	0	1	87	nominal aastore per usec
BEF	9	11	3	1	4	28	nominal execution focus / dominance of hot code
BGF	8	6123	4	26	507	33553	nominal getfield per usec
BPF	2	31	15	2	84	3346	nominal putfield per usec
BUB	6	55	8	8	35	177	nominal thousands of unique bytecodes executed
BUF	5	4	9	1	4	29	nominal thousands of unique function calls
GCA	0	20	22	20	98	136	nominal average post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCC	1	68	21	31	965	17270	nominal GC count at 2X heap size (G1)
GCM	0	22	22	22	94	150	nominal median post-GC heap size as percent of min heap, when run at 2X min heap with G1
GCP	2	1	18	0	3	77	nominal percentage of time spent in GC pauses at 2X heap size (G1)
GLK	10	98	1	0	0	98	nominal percent 10th iteration memory leakage
GMD	10	195	2	5	71	681	nominal minimum heap size (MB) for default size configuration (with compressed pointers)
GMS	2	5	18	5	13	157	nominal minimum heap size (MB) for small size configuration (with compressed pointers)
GMU	5	97	11	7	73	902	nominal minimum heap size (MB) for default size without compressed pointers
GSS	1	6	20	0	64	7778	nominal heap size sensitivity (slowdown with tight heap, as a percentage)
GTO	1	8	19	3	53	1103	nominal memory turnover (total alloc bytes / min heap bytes)
PCC	7	256	8	-1	200	1092	nominal percentage slowdown due to aggressive c2 compilation compared to baseline (compiler cost)
PCS	5	63	12	2	63	321	nominal percentage slowdown due to worst compiler configuration compared to best (sensitivty to compiler)
PET	2	1	19	1	3	8	nominal execution time (sec)
PFS	5	10	13	0	12	19	nominal percentage speedup due to enabling frequency scaling (CPU frequency sensitivity)
PIN	5	63	12	2	63	321	nominal percentage slowdown due to using the interpreter (sensitivty to interpreter)
PKP	6	6	10	0	5	62	nominal percentage of time spent in kernel mode (as percentage of user plus kernel time)
PLS	3	0	16	-1	4	36	nominal percentage slowdown due to 1/16 reduction of LLC capacity (LLC sensitivity)
PMS	5	2	13	0	3	35	nominal percentage slowdown due to slower memory (memory speed sensitivity)
PPE	10	66	2	3	7	91	nominal parallel efficiency (speedup as percentage of ideal speedup for 32 threads)
PSD	8	1	6	0	1	13	nominal standard deviation among invocations at peak performance (as percentage of performance)
PWU	9	7	4	1	3	9	nominal iterations to warm up to within 1.5% of best
UAA	3	613	15	19	737	1452	nominal percentage change (slowdown) when running on ARM Calvium ThunderX v AMD Zen4
UAI	9	35	3	-22	22	41	nominal percentage change (slowdown) when running on Intel Alderlake v AMD Zen4
UBC	1	20	20	15	33	181	nominal backend bound (CPU)
UBM	7	49	8	5	37	109	nominal bad speculation: mispredicts
UBP	1	383	21	87	977	3209	nominal bad speculation: pipeline restarts
UBS	7	50	8	6	38	112	nominal bad speculation
UDC	1	3	20	2	13	27	nominal data cache misses per K instructions
UDT	0	13	22	13	289	1101	nominal DTLB misses per M instructions
UIP	8	209	6	92	138	476	nominal 100 x instructions per cycle (IPC)
ULL	0	318	22	318	3001	8545	nominal LLC misses M instructions
USB	0	6	22	6	29	53	nominal 100 x back end bound
USC	10	348	1	1	53	348	nominal SMT contention
USF	4	19	15	4	27	61	nominal 100 x front end bound



(a) Wall clock.

(b) Total CPU overheads (TASK_CLOCK).

Fig. 52. Lower bounds on the overheads [2] for zxing for each of OpenJDK 21's six production garbage collectors as a function of heap size. The figure on the left shows the overhead in terms of wall clock time while the figure on the right shows the overhead using the Linux perf TASK_CLOCK, which sums the running time of all threads in the process, giving the total computation overhead.

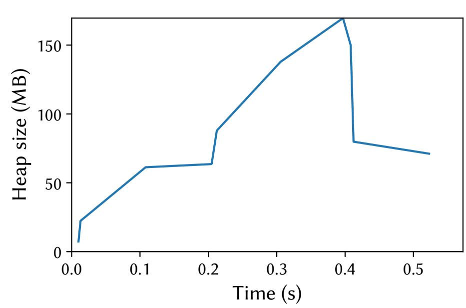


Fig. 53. Heap size post each garbage collection, with the time relative to the start of the last benchmark iteration. The benchmark is running with OpenJDK 21's G1 collector at 2.0× heap.

REFERENCES

[1] 2016. The AVR Simulation and Analysis Framework. https://github.com/avrora-framework

[2] Zixian Cai, Stephen M. Blackburn, Michael D. Bond, and Martin Maas. 2022. Distilling the Real Cost of Production Garbage Collectors. In *International IEEE Symposium on Performance Analysis of Systems and Software, ISPASS 2022, Singapore, May 22-24, 2022*. IEEE, 46–57. https://doi.org/10.1109/ISPASS55109.2022.00005

[3] Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, Chandu Thekkath and Amin Vahdat (Eds.). USENIX Association, 31–46. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola

[4] Nick Mitchell, Edith Schonberg, and Gary Sevitsky. 2009. Making Sense of Large Heaps. In *ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5653)*, Sophia Drossopoulou (Ed.). Springer, 77–97. https://doi.org/10.1007/978-3-642-03013-0_5

[5] Nick Mitchell, Edith Schonberg, and Gary Sevitsky. 2010. Four Trends Leading to Java Runtime Bloat. *IEEE Softw.* 27, 1 (2010), 56–63. https://doi.org/10.1109/MS.2010.7

[6] Nick Mitchell, Gary Sevitsky, and Harini Srinivasan. 2006. Modeling Runtime Behavior in Framework-Based Applications. In *ECOOP 2006 - Object-Oriented Programming, 20th European Conference, Nantes, France, July 3-7, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4067)*, Dave Thomas (Ed.). Springer, 429–451. https://doi.org/10.1007/11785477_25