Investigating the scalability of Go's garbage collector in multicore environments

Nihar Sheth

High level programming languages

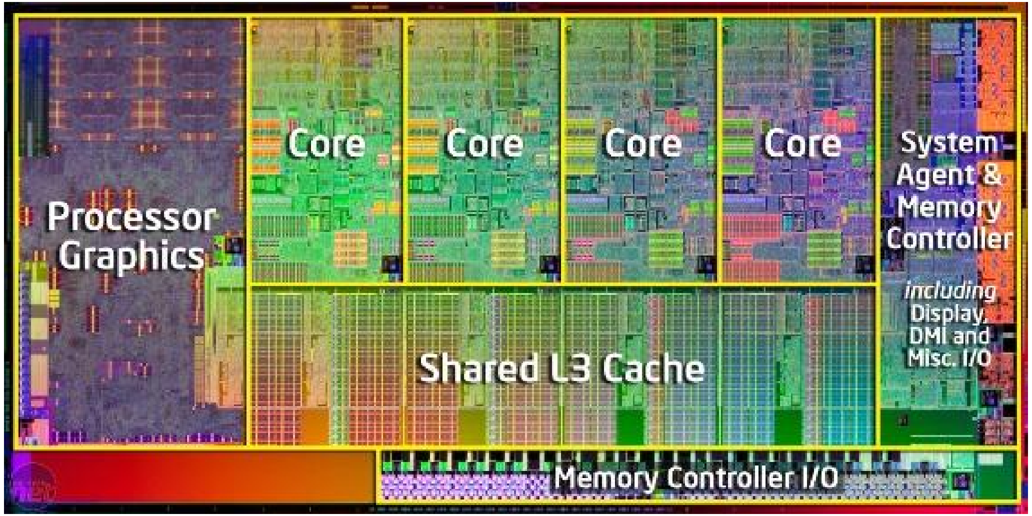
- Examples include Go, Java, Python
- Greater abstraction from hardware
- Eliminate certain classes of bugs
- Code is:
  - Easier to write
  - Easier to maintain
  - Easier to debug
- Desirable from developer perspective, less so from performance perspective

Garbage collection

- Automatic reclamation of unused memory by runtime (rather than programmer)
- Programmer does not have to worry about freeing memory and tracking when it is used/unused
- Reduces risk of hard-to-debug issues such as memory leaks

Parallelization

- Modern machines have a lot of CPUs
- If we want efficient programs, they need to be parallel



https://cnet3.cbsistatic.com/hub/i/2011/09/13/97506276-fdb9-11e2-8c7c-d4ae52e62bcc/21196cc0e9bf31954c21004a3c1ee115/inside_intel_sandy_bridge_quad_core_processor.jpg

What if the garbage collector doesn’t scale?

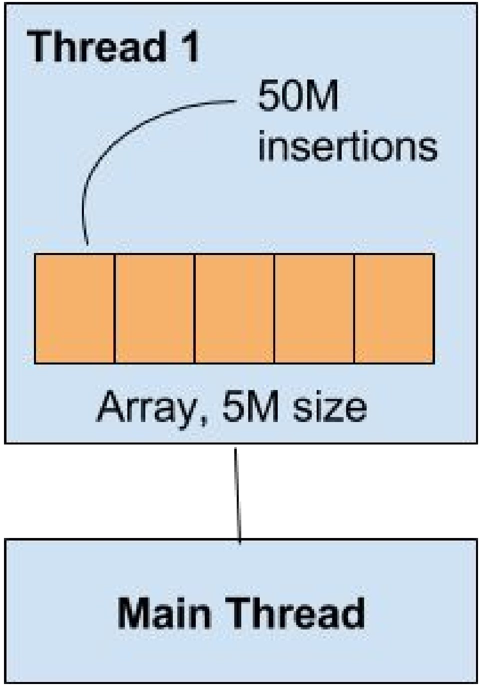
- Can’t add CPUs to speed up application, even for perfectly scalable workloads!
  - Waste of extra CPUs
- Unable to take advantage of parallelism

Goal: find potential scaling problems in the Go garbage collector

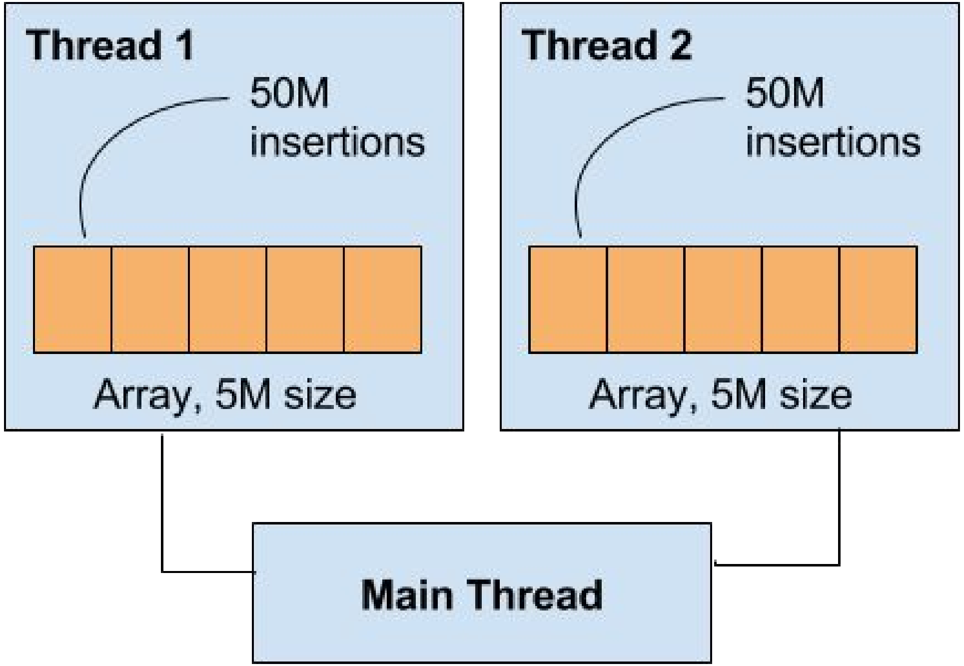
Approach

- Wrote microbenchmark to allocate at rates similar to what we would expect in a real-world setting
- Parallel allocation with varying numbers of threads
- Amount of garbage generated scaled proportionally with number of CPUs, so with true scalability, no increase in clock time would be expected

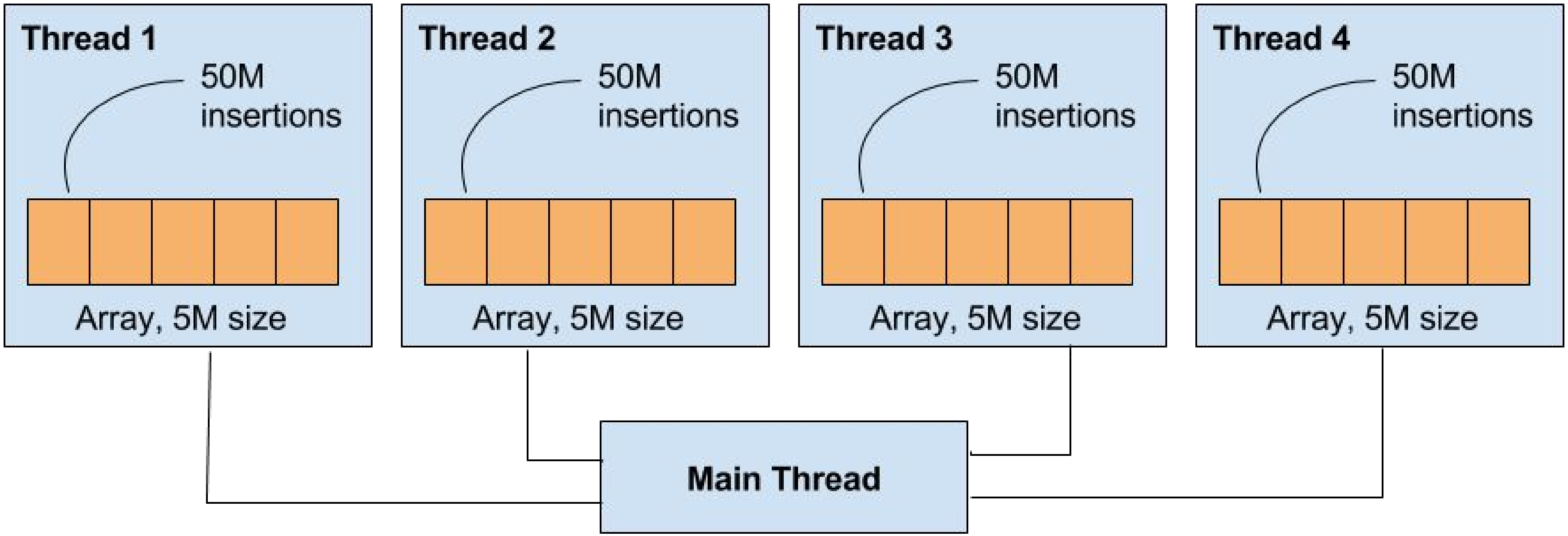
Benchmark Setup (1 CPU)



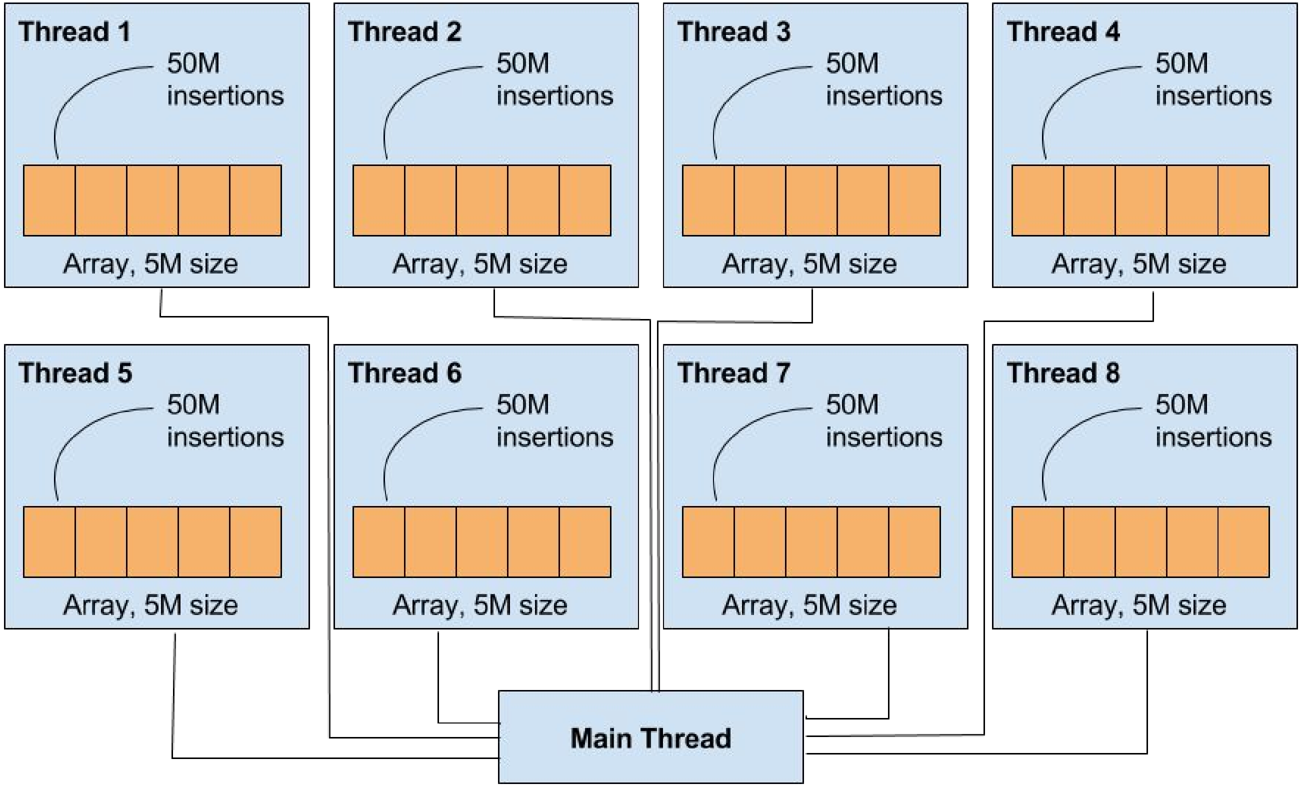
Benchmark Setup (2 CPUs)



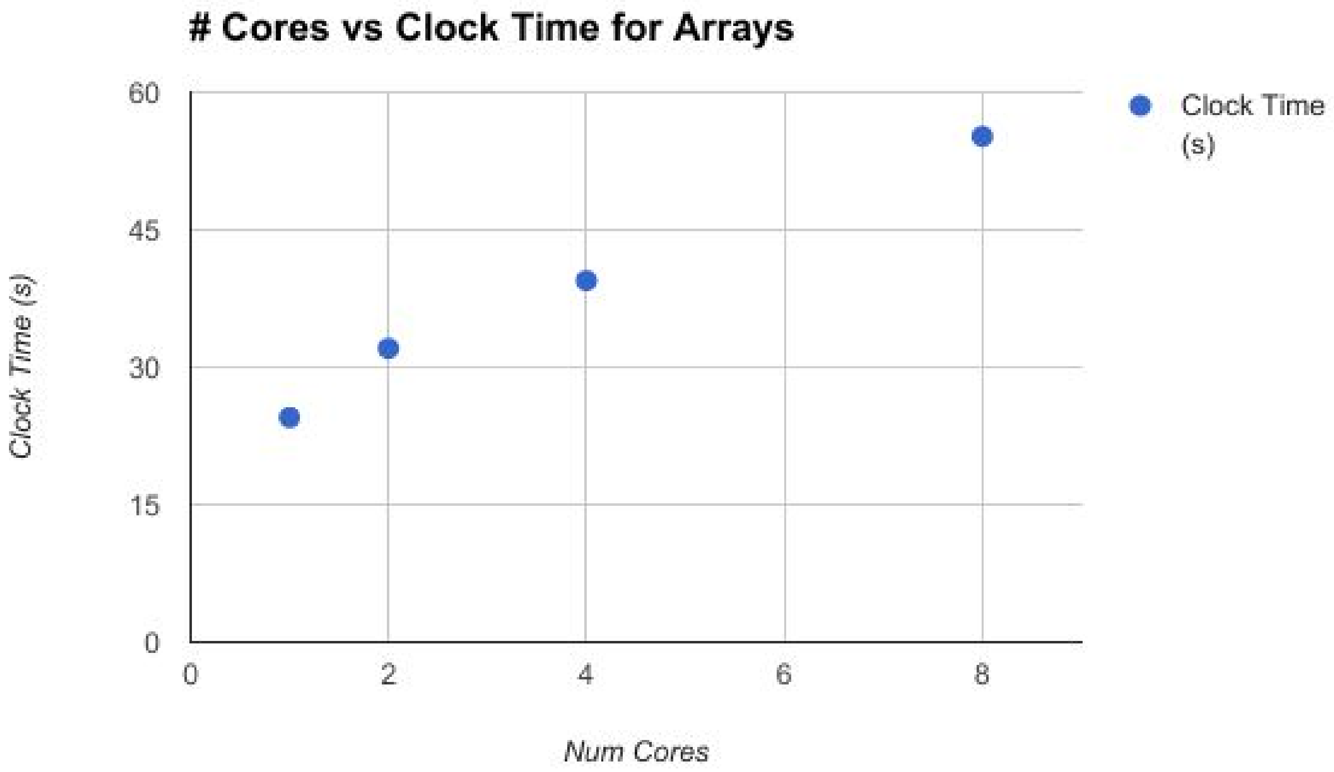
Benchmark Setup (4 CPUs)



Benchmark setup (8 CPUs)

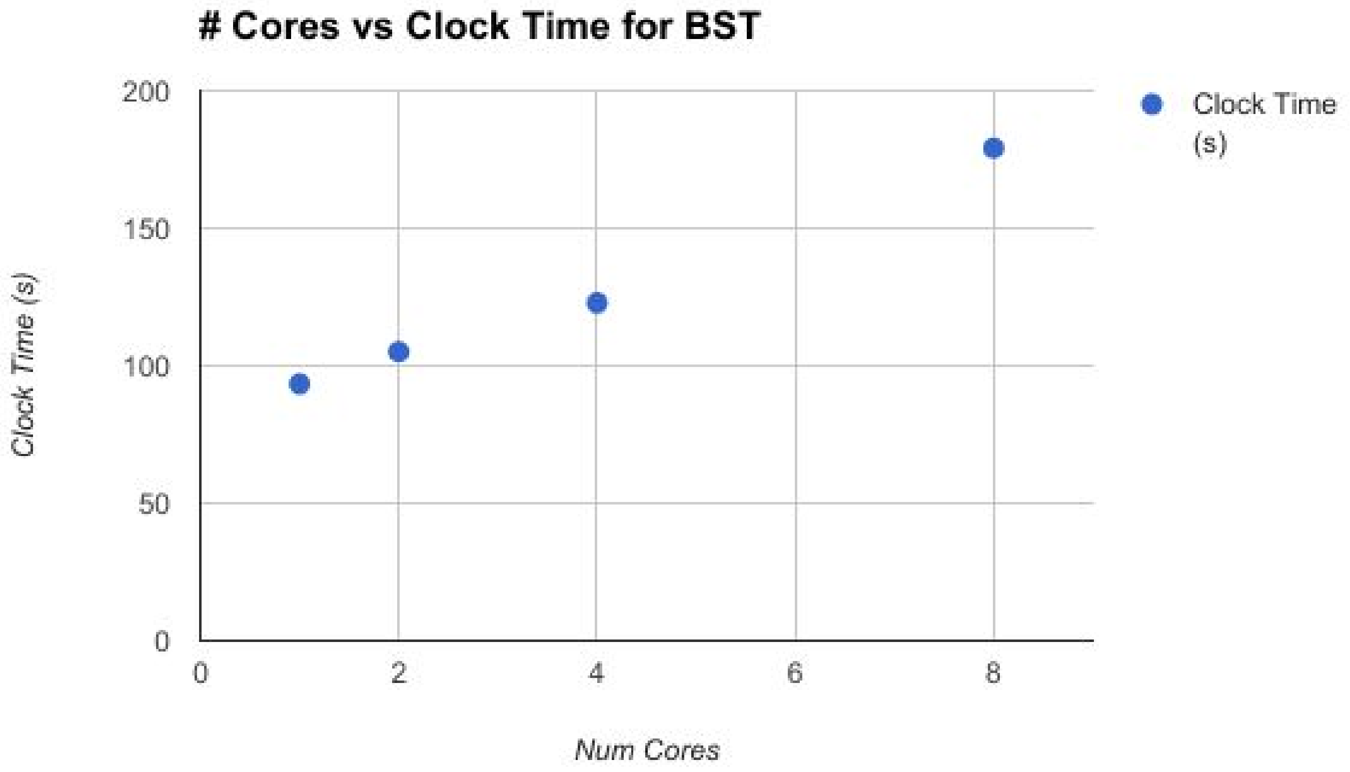


Results



Cores vs Clock Time for Arrays
60
45
30
15
0
Clock Time (s)
0
2
4
6
8
Num Cores
Clock Time (s)

Results



Cores vs Clock Time for BST
Clock Time (s)
Num Cores
Clock Time
(s)
200
150
100
50
0
0
2
4
6
8

Conclusions

- Scalability of Go’s garbage collector leaves room for improvement
  - Scalability issues, even for reasonable rates of allocation
- Why does this happen?
  - Contention on central pool of free memory

Future Work

- Test more data structures -- not just arrays and binary search trees
- Test impact of different allocation patterns