Improvements to GHC’s parallel garbage collector

Douglas Wilson

2021-08-22 — Copyright © 2021 Well-Typed LLP
Introduction

- Independent work conducted in summer of 2020/2021.
- Will be released in GHC 9.2
- Linux, amd64
- Improved cross-thread synchronisation in the garbage collector drastically reduces cpu time
  - Measurement of improvement
  - Description of changes
  - Some experiments
Measurement of improvement

cd libraries/Cabal && ghc --make -j$i Setup.hs

Environment: AMD zen3, 8 physical cores, 16GB, Laptop, hyper-threading on, 12 core cpuset
sched_yield in 60 seconds

implemented in kernel/sched/fair.c

GHC issue 9221

From sched_yield (2) man page:

```
sched_yield() causes the calling thread to relinquish the CPU. The thread is moved to the
end of the queue for its static priority and a new thread gets to run.
```

```
... sched_yield() is intended for use with real-time scheduling policies (i.e., SCHED_FIFO
or SCHED_RR). Use of sched_yield() with nondeterministic scheduling policies such as
SCHED_OTHER is unspecified and very likely means your application design is broken.
```

sched_yield is either:

- A busy spin;
- Effectively lowering our priority by forfeiting our timeslice.

Because ghc launches assemblers and linkers, it more often experiences a lowering of
priority.
Don’t use `sched_yiel`d for busy-waits, use mutexes and condition variables. Take care to identify and maintain invariants, and to minimize `pthread_cond_broadcast`. Three places:

- GC entry
- GC exit
- work stealing
GC entry and exit

1. All worker threads change from INACTIVE to STANDING_BY and block;
2. The gc leader waits in waitForGcThreads for all worker threads to reach STANDING_BY;
3. The gc leader does some initialisation;
4. The gc leader calls wakeup_gc_threads, which sets the worker threads to RUNNING and wakes them up;
5. All worker threads run out of work and change from RUNNING to WAITING_TO_CONTINUE and block;
6. The gc leader waits in shutdown_gc_threads for all worker threads to reach WAITING_TO_CONTINUE;
7. The gc leader does some cleanup;
8. The gc leader calls releaseGcThreads, which sets all worker threads to INACTIVE and wakes them up.
work stealing

1. Track the number of threads working with global variable gc_running_threads;
2. Threads that aren’t working are waiting for work to appear in other threads’ queues;
3. gc_running_threads reaching zero is the stop condition.

How to wake up threads when blocks are ready?

Signal (not broadcast) when a block is ready and gc_running_threads < n.
Measuring sched_yield

- stat = mut_spin_yield
- stat = gc_spin_yield
- stat = waitforGCThreads_yield
- stat = any_work
- stat = wholelockClosure_yield
- stat = gc_alloc_block_sync_yield
- stat = gen_0_sync_yield
- stat = gen_1_sync_yield

Graphs show the value of sched_yield for different statistics and core counts, comparing `pargc-baseline` and `pargc-nospingc`.
Introducing background noise

Run the same benchmarks, while running ‘stress-ng’ simultaneously

stress-ng --cpu 0 --cpu-method ackerman --cpu-load 40
Future work

- Investigate removing remaining sched_yield calls;
- Benchmarking on Windows, Darwin, ARM, etc.
- Investigate stack and cabal-install exploit ghc -j
code: https://github.com/duog/ghcbench

slides: https://github.com/duog/ghcbench/tree/master/hiw/hiw.pdf

ghc commits:

- baseline
  - link: https://gitlab.haskell.org/duog/ghc/-/tree/pargc-baseline
  - commit: 3a536b890f88c16166f4d68ecf1ed8f49dd6f661

- noanywork:
  - link: https://gitlab.haskell.org/duog/ghc/-/tree/pargc-noanywork
  - commit: ce0f280908c29c1608c4f62002d325e6b931d98d

- nospingc:
  - link: https://gitlab.haskell.org/duog/ghc/-/tree/pargc-nospingc
  - commit: 0dc550105acedb714f0902cfb3dfd8f03fa08272