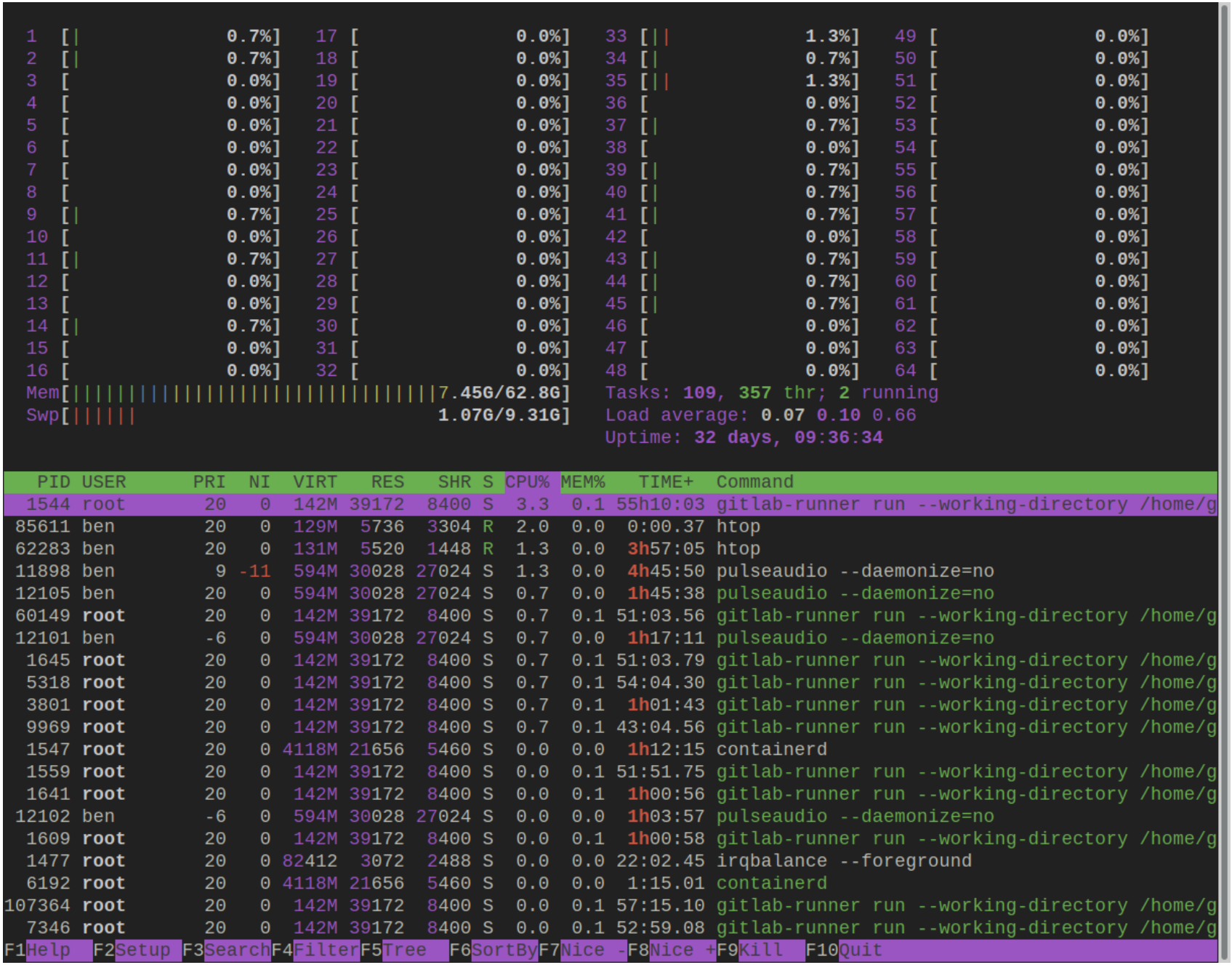
Runtime partitions for scaling GHC programs

Ben Gamari

28 Aug 2020

The Problem


1 [| 0.7%] 17 [0.0%] 33 [|| 1.3%] 49 [0.0%]
2 [| 0.7%] 18 [0.0%] 34 [| 0.7%] 50 [0.0%]
3 [0.0%] 19 [0.0%] 35 [|| 1.3%] 51 [0.0%]
4 [0.0%] 20 [0.0%] 36 [0.0%] 52 [0.0%]
5 [0.0%] 21 [0.0%] 37 [| 0.7%] 53 [0.0%]
6 [0.0%] 22 [0.0%] 38 [0.0%] 54 [0.0%]
7 [0.0%] 23 [0.0%] 39 [| 0.7%] 55 [0.0%]
8 [0.0%] 24 [0.0%] 40 [| 0.7%] 56 [0.0%]
9 [| 0.7%] 25 [0.0%] 41 [| 0.7%] 57 [0.0%]
10 [0.0%] 26 [0.0%] 42 [0.0%] 58 [0.0%]
11 [| 0.7%] 27 [0.0%] 43 [| 0.7%] 59 [0.0%]
12 [0.0%] 28 [0.0%] 44 [| 0.7%] 60 [0.0%]
13 [0.0%] 29 [0.0%] 45 [| 0.7%] 61 [0.0%]
14 [| 0.7%] 30 [0.0%] 46 [0.0%] 62 [0.0%]
15 [0.0%] 31 [0.0%] 47 [0.0%] 63 [0.0%]
16 [0.0%] 32 [0.0%] 48 [0.0%] 64 [0.0%]
Mem[|||||||||||||||||||||||||||7.45G/62.8G]
Swp[|||||| 1.07G/9.31G]
Tasks: 109, 357 thr; 2 running
Load average: 0.07 0.10 0.66
Uptime: 32 days, 09:36:34
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1544 root 20 0 142M 39172 8400 S 3.3 0.1 55h10:03 gitlab-runner run --working-directory /home/g
85611 ben 20 0 129M 5736 3304 R 2.0 0.0 0:00.37 htop
62283 ben 20 0 131M 5520 1448 R 1.3 0.0 3h57:05 htop
11898 ben 9 -11 594M 30028 27024 S 1.3 0.0 4h45:50 pulseaudio --daemonize=no
12105 ben 20 0 594M 30028 27024 S 0.7 0.0 1h45:38 pulseaudio --daemonize=no
60149 root 20 0 142M 39172 8400 S 0.7 0.1 51:03.56 gitlab-runner run --working-directory /home/g
12101 ben -6 0 594M 30028 27024 S 0.7 0.0 1h17:11 pulseaudio --daemonize=no
1645 root 20 0 142M 39172 8400 S 0.7 0.1 51:03.79 gitlab-runner run --working-directory /home/g
5318 root 20 0 142M 39172 8400 S 0.7 0.1 54:04.30 gitlab-runner run --working-directory /home/g
3801 root 20 0 142M 39172 8400 S 0.7 0.1 1h01:43 gitlab-runner run --working-directory /home/g
9969 root 20 0 142M 39172 8400 S 0.7 0.1 43:04.56 gitlab-runner run --working-directory /home/g
1547 root 20 0 4118M 21656 5460 S 0.0 0.0 1h12:15 containerd
1559 root 20 0 142M 39172 8400 S 0.0 0.1 51:51.75 gitlab-runner run --working-directory /home/g
1641 root 20 0 142M 39172 8400 S 0.0 0.1 1h00:56 gitlab-runner run --working-directory /home/g
12102 ben -6 0 594M 30028 27024 S 0.0 0.0 1h03:57 pulseaudio --daemonize=no
1609 root 20 0 142M 39172 8400 S 0.0 0.1 1h00:58 gitlab-runner run --working-directory /home/g
1477 root 20 0 82412 3072 2488 S 0.0 0.0 22:02.45 irqbalance --foreground
6192 root 20 0 4118M 21656 5460 S 0.0 0.0 1:15.01 containerd
107364 root 20 0 142M 39172 8400 S 0.0 0.1 57:15.10 gitlab-runner run --working-directory /home/g
7346 root 20 0 142M 39172 8400 S 0.0 0.1 52:59.08 gitlab-runner run --working-directory /home/g
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit

The Problem

GC synchronization leads to poorly scaling across many cores.

Server applications have competing pressures:

- Low-latency wants small nurseries $\longrightarrow$ lots of sychronization
- Multi-core scaling demands large nurseries (minimize synchronization)

A Solution: Local heaps

Marlow 2011 [1]:

- Give each core its own independent nursery
- Prohibit references between local nurseries, allowing nursery GCs to happen without synchronizations
- "Globalise" data which is needed by other cores
- Implicit globalisation: non-obvious heuristics, hard to reason about performance

[1] S. Marlow & S. Peyton-Jones. "Multicore garbage collection with local heaps." *Proceedings of the 10th International Symposium on Memory Management* (2011)

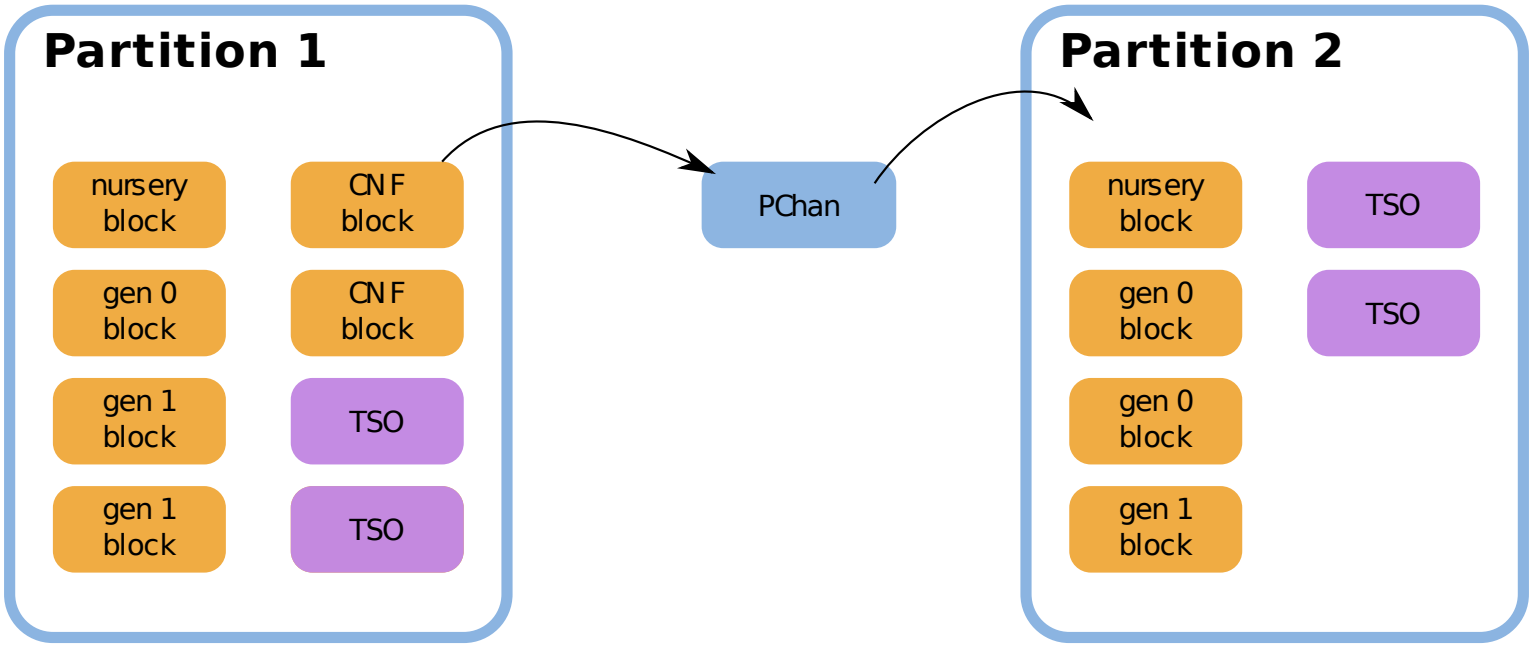
A Solution: Multiple processes

```
forkMap
  :: (NFData a, NFData b)   -- ^ needed to build compact region
  => Int                    -- ^ number of workers to spawn
  -> StaticPtr (a -> b)     -- ^ StaticPtr since we must send function
                            -- across the wire.
  -> Producer a IO r
  -> Producer b IO r
```

- Communicate via pipes
- `StaticPtr`s allow sharing of code references between processes
- Compact normal forms can be used for efficient(?) serialisation
- Feature request: CNFs could be `mmap`'d between processes

A Solution: Partitioning

Explicit heap partitions within a process.



A Solution: Partitioning (message-passing)

```haskell
-- | Similar to a TChan, but allowing sharing of values ac
data PChan a

writeTChan :: PChan a -> a -> IO ()
readTChan  :: PChan a -> IO a
```

A Solution: Partitioning

```haskell
-- | A reference-counted "globalized" value
data PRef a

newPRef :: a -> IO (PRef a)

-- | Use a value held in a 'PRef'.
withPRef :: (a #-> b) -> PRef a -> b
  -- Sadly, this is broken due to laziness.
```

“Arenas” via partitioning

ephemeral partitions:

- Run a thread in a partition
- After finished, tear down the world

Avoids GC entirely for sufficiently short-lived partitions.

Implementation

- Block descriptor: identify owner partition
- Capabilities can run threads from any partition
- `mut_list` must be flushed to global (non-capability-local) remembered set when a capability switches partitions

The challenge of CAFs

CAFs are a shared resource.

A few options:

- Don't update CAFs (yuck!)
- Introduce a new "per-partition" indirection type
- Globalize all CAF evaluations?

Volunteers?

Who wants to implement this?