GHC の GC

LT, Haskell Day
2019.11.9

@kazu_yamamoto

自己紹介

IIJ II 勤務

ネットワークプロトコルをHaskell実装

Warp、network、TLS ライブラリの開発者

今のGC

世代別 並列 コピー GC

次のGC

新世代 「並列 コピー GC」のまま
旧世代「並行 マーク＆スイープ GC」

コピー GC

領域を二つに分割し
生きてるオブジェクトを他方へコピーする

GCが速い
領域がコンパクトになる
オブジェクトの割り当てが速い
領域の半分しか使えない

マーク＆スイープGC

生きてるオブジェクトをマークし
全領域をスキャンして
死んだオブジェクトをリストでつなぐ

GCが遅い
領域が断片化される
オブジェクトの割り当てが遅い
領域は全部使える

関数プログラミング

新しい値を作っては捨てる
コピーGCと相性がいい

並列

領域は大きさごとに
複数用意されている

簡単に並列化できる

世代別

領域を新世代と旧世代に分類する

新世代のGCを3回生き延びると
旧世代へ配置替えされる

世代別仮説

古いオブジェクトよりも
新しいオブジェクトの方が
すぐに死ぬ可能性が高い

関数プログラミングでは
仮説は正しい！

新世代のGCを頻繁にやればよい

世代別 並列 コピー GC

なにか問題でも？

新世代のGC

マイナーGCは速い
領域の大きさは固定

旧世代のGC

メジャーGCは遅い
領域は無限に大きくなる

Stop The World!

GCは一挙にやる

スループットは高い
応答性は低い

"From Haskell to Go"

スループットは遅いが
応答性が高いGCを持つ
Go に乗り換えちゃうもんね

応答性を高めるために
GC をインクリメンタルにやる

コアが複数あれば
GCとプログラムが同時に走れるので
並行GCと呼ばれる

インクリメンタル コピーGC

理論はある
実装には「リードバリア」が必要

インクリメンタル マーク＆スイープ GC

多くの実装がこれ
リードバリアは必要ない
旧世代だとコピーGCにこだわる必要なし

今のGC

世代別 並列 コピー GC

次のGC

新世代 「並列 コピー GC」のまま
旧世代「並行 マーク＆スイープ GC」