Pen 言語における並列計算

目次

世はマルチコア時代

- マルチコア CPU を使って処理を速くしたい
  - データ並列: 複数の（同種の）データを同時に処理したい <- 今日はこっち
  - タスク並列: 異なるコードを同時に実行したい

Go 言語のチャネル

- いわゆる concurrent queue
- スレッドセーフ
- 用途:HTTP サーバ、バッチ処理

例

- チャネル型: `chan int`
- エンキュー: `c <- 42`
- デキュー: `x := <- c`
- チャネルの作成: `make(chan int, 64)`
- チャネルのクローズ: `close(c)`

Go 言語のチャネルの利点・欠点

利点

- 高パフォーマンス
- スレッドのような概念と相性がいい

欠点

- 関数型言語と相性が悪い
    - ミュータブルで状態を持つ
    - 手続き的な操作

Cloe 言語とは

- https://cloe-lang.org
- インタプリタ型
- 動的型付け
- 遅延評価
- Go 言語製
- 今は、もう、メンテされていない
- **遅延リストとプリミティブの関数で並列計算**を実装
  - https://cloe-lang.org/api-reference/builtin#parallelism-and-concurrency

遅延リストの性能

- HTTP のサーバーを実装
  - HTTP リクエストをコンカレントに受け付ける
- Go, Node.js, Cloe で比較

結果

- Go >>> Node.js > Cloe
- Cloe が Node.js より 20 ~ 30%遅い
- 実際の HTTP サーバーの実装では、各リクエストの処理で時間がかかる
  - 実用に耐える？
- CPU 処理過多な用途では確実に負けそう

遅延リストによる並列計算の制限

- サンクの実装にパフォーマンスが律速される
  - 並行キューに比べてヒープのアロケーションが多い
  - Haskell でのベンチマーク
  - I/O 過多なプログラムでは問題ない?
- 全ての値が帰納的 -> 再帰ができない
  - Pen 言語特有の問題
  - 大体のコンカレンシーパターンは実装できる
    - 内部的には concurrent queue を使うので、それはそう
    - "Concurrency in Go" by Katherine Cox-Buday
  - ある種のデッドロックを防げる
  - e.g. タスクがタスクを生む場合、バックプレッシャ
- スペースリーク

Pen 言語での設計

必要条件

- ジェネリクス
- 決定性を破壊しない -> コンテキストを渡す

例

```
import Core'Parallel

foo = \(ctx ConcurrencyContext, xs [number]) [number] {
  concurrency = 200

  Parallel'Map(ctx, concurrency) ~ xs
}
```

色々なコンカレンシーパターン

```
Parallel'Map(ctx, c) # [number] -> [number]
Parallel'Race(ctx, c) # [number] -> [number]

Parallel'Split(ctx, n) # [number] -> [[number]]
Parallel'Join(ctx) # [[number]] -> [number]
```

ご意見・ご感想お待ちしております