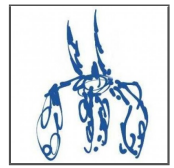
並列並行言語 Haskell



@syocy

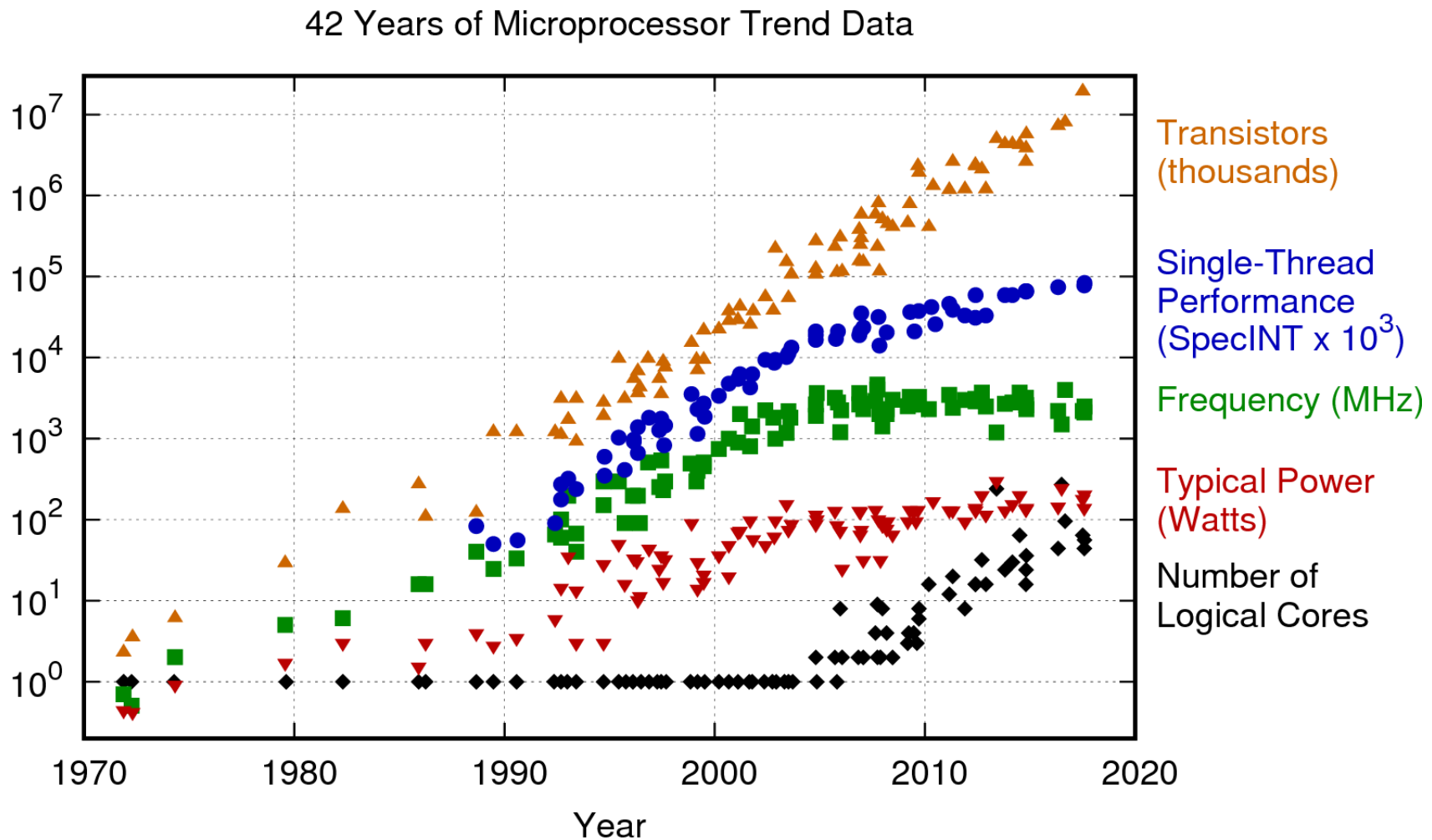
2018-11-10

参考書

Haskell による並列・並行プログラミング

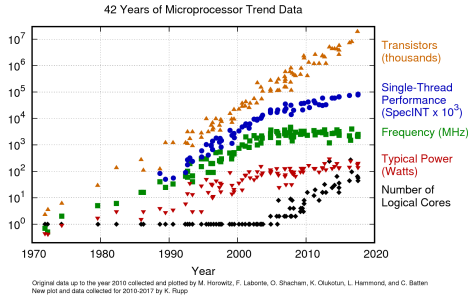
- GHC の主要開発者 Simon Marlow 自身による並列・並行 Haskell の解説書
- 並列・並行の様々なアイデアが紹介されており Haskell ユーザ以外にもおすすめ

プロセッサ性能トレンド



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

プロセッサ性能トレンド



- シングルスレッド性能は伸び悩んできている
- 一方で論理コア数は順調に増えてきている
- 現代のプロセッサの能力を引き出すにはコア数を活かすプログラミングが必要

お求めやすくなったメニーコア CPU

- **10 以上[†]の物理コアを持つ CPU がご家庭でも手に入れられるお値段になった**

CPU	Core	Freq	Price
Ryzen TR 2990WX	32-core 64-thread	3.0GHz	$1799
Ryzen TR 2950X	16-core 32-thread	3.5GHz	$899
Ryzen TR 1920X	12-core 24-thread	3.5GHz	$399

[†]メニーコアの定義は曖昧。ここでは物理 10 コア以上をメニーコアと呼ぶことにする。

並列並行言語の台頭

最近話題の言語はコアな部分に並列並行機能を備えていることが多い
→ 並列並行の重要性が意識され始めた

- Go: goroutine(コルーチン) を持つ
- Erlang, Elixir: VM が軽量プロセスを持つ
- Rust: メモリー安全性が並行性にも及ぶことを強調している

Haskell はどうか？

並列・並行と Haskell

Haskell (GHC) は古くから並列・並行を考えて設計されてきた

- 1997 年にライブラリではなく実行時システムとして並行性をサポートすることが決定
- 2004 年に実行時システムを共有メモリのマルチプロセッサ上で並列に動作させることが決定
- 2009 年の論文 “Runtime support for multicore Haskell”

並列・並行のためのよい性質

Haskell は並列・並行のためのよい特徴を持つ

- Haskell は純粋なコードと副作用 (IO など) を含むコードを分離できる
  - 並列性は**決定的**: 並列度や実行環境が変わろうと結果は変わらない
- 実行時システムが**軽量スレッド**をサポートする
  - よく並行性の実現にはスレッドが用いられるが、
  - あたかも普通の (OS) スレッドのように使えるものが、普通のスレッドよりはるかに軽く動作する

Haskell 使うっきゃない！

並列・並行（・分散）って？

これまで断りなく使ってきた言葉

- **並列** (parallel)
- **並行** (concurrent)
- (**分散** (distributed))

に違いはあるの？

並列と並行

- 並列 (parallel): 同時に走らせることで処理を高速化したい
  - 用例: n 並列、並列ダウンロード、GPU 並列計算
- 並行 (concurrent): そもそも同時にしたい（同時であるかのように見せたい）処理がある
  - 同時さは擬似的でもよい
  - 並行の表現にはよくスレッドの概念が用いられる‡

‡スレッドが本当に同時に動く実装の場合、並列のためにスレッドを使うこともある

分散

分散は並列・並行とは異なる特徴を持つ

- 分散 (distributed): 複数のマシンを使う処理のこと
  - 通信時間がかかるため、基本共有メモリを持たない
  - 一部のマシンがダウンするかもしれない
  - マシンごとに性質が異なることがありうる

このスライドでは分散にはあまり触れない

並列・並行のコード

- **並列 Haskell は Haskell 特有の性質を理解していないと把握しづらい**
- **そのためまずは並行 Haskell (軽量スレッド) から見ていく**

軽量スレッドを作成する

```haskell
-- >>> helloworld
-- Hello, World!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
helloworld :: IO ()
helloworld = do
  ts <- replicateM 100 $ async $ do -- スレッド100個作る
          threadDelay 100           ---- 100μsスリープ
          putStr "!"                ---- "!" を出力
  putStr "Hello, World"             -- "Hello, World"
  forM_ ts wait                     -- スレッド終了待ち
  putStrLn ""
```

軽量スレッドを作成する

- `async` 関数で軽量スレッドを作る
  - async パッケージに入っている
  - 実体は標準関数 `forkIO` の薄いラッパー。`async` の方がより安全で便利なので利用推奨。
- `wait` 関数で軽量スレッドの終了を待つこともできる
- `cancel` 関数で軽量スレッドを外から止めることもできる
  - 軽量スレッドに非同期例外が飛ぶ形になる

スレッド間通信

スレッド間通信には大まかに２つの方法がある

- MVar
  - シンプルな同期変数
  - アクセスの公平性が保証される
  - MVar によるチャネルとセマフォの実装がある
- STM(Software Transactional Memory)
  - 共有状態の読み書きにトランザクションの概念を導入
    - 割り込まれない一連の読み書きブロック
    - 途中で失敗したらなかったことにしてリトライできる
  - 複雑な共有状態の処理をミスなく記述しやすい
  - STM によるチャネル、キュー、制限付きキュー、セマフォ等の実装がある

スレッド間通信

STM で共有カウンターを操作する例

```
1  -- >>> atomicCounter
2  -- 1000
3  atomicCounter :: IO ()
4  atomicCounter = do
5    counter <- newTVarIO (0 :: Int)
6    ts <- replicateM 1000 $ async $ do
7            atomically $ do
8              modifyTVar' counter $ (+1)
9    forM_ ts wait
10   print =<< atomically (readTVar counter)
```

宣伝: A Tour of Go in Haskell§

A Tour of Go in Haskell

GITHUB LANGUAGE

A Tour of Go in Haskell へようこそ

A Tour of Go in Haskell へようこそ。 このサイトは、Go の有名なチュートリアル A Tour of Go の 並行性 の章を Haskell でやってみるというものです。 Haskell は Go と同じく *軽量スレッド* や *チャネル* といった並行処理の機能を持っています。 そのため Haskell で Go と同じ処理を記述して2つを見比べてみるのは興味深いことでしょう。

ここから始めましょう。

Haskellに詳しい方へ: このサイトでは Go との対比を分かりやすくするため、もっぱら async, stm などのパッケージを使って IO の中で明示的に並行性を扱います。 Evalモナド、Parモナドなどは扱いません。 また、Haskell ユーザでない人の分かりやすさのために、意図的にユーティリティ関数の使用を避けたりポイントフリーではない記述をしている部分があります。

目次

1. Goroutines
2. Channels

- Go **のチュートリアル** “A Tour of Go” **の並行性の章を** Haskell **で書いた**
- **並行構文の軽さは** Go **と** Haskell **で同じくらい**
- STM **があるぶん** Haskell **の方がうまく書ける例も**

§https://a-tour-of-go-in-haskell.syocy.net/ja_JP/index.html

評価順序を改変する: 並列化前のコード

- Haskell は必要になった式を「順番に」評価していく
- デフォルトで GC 以外が自動的に並列化されることはない (はず)
- しかし、並列に評価してほしい箇所を指示できる

```
1  -- >>> mutualPow 5 2
2  -- (25,32)
3  mutualPow :: Int -> Int -> (Int, Int)
4  mutualPow x y = let z1 = x ^ y in
5                  let z2 = y ^ x in
6                  (z1, z2) -- z1とz2を並列に計算したい
```

評価順序を改変する: 評価の並列化

- par 関数は第一引数の評価を並列化する
- pseq 関数は直列化を指示する

```
1  -- >>> mutualPowPar 5 2
2  -- (25,32)
3  mutualPowPar :: Int -> Int -> (Int, Int)
4  mutualPowPar x y = let z1 = x ^ y in
5                     let z2 = y ^ x in
6                     z1 `par` z2 `pseq` (z1, z2)
```

評価順序を改変する: 評価戦略の分離

- 典型的なデータ構造にいちいち並列化を指示するのは面倒
- **評価戦略**という形で並列化指示を分離できる
- 自作のデータ構造に評価戦略を作ることもできる

```
1  -- >>> mutualPowSt 5 2
2  -- (25,32)
3  mutualPowSt :: Int -> Int -> (Int, Int)
4  mutualPowSt x y = (mutualPow x y)
5                    `using` (parTuple2 rseq rseq)
```

より高レイヤーのツール

- ここまで Haskell のプリミティブな並列・並行機能（軽量スレッド、評価戦略）を見てきた
- ここからはより高レイヤーの並列・並行ツールを簡単に紹介していく

自動的な並列化

- Par モナド
  - データフロー並列: データフローグラフの並列にできるところを並列化
  - パイプライン並列: データ処理パイプラインの各段を並列化
- Haxl
  - データソースへのクエリを自動的に並列化する
  - Facebook のスパムフィルタで使われている

行列計算の並列化

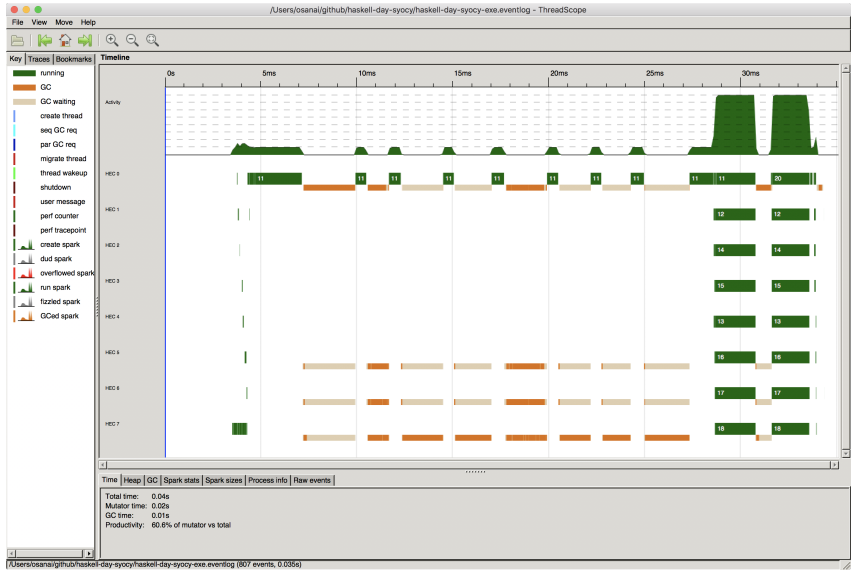
- repa
  - **行列計算について自動並列の一種であるデータ並列を導入する**
- accelerate
  - **行列計算の並列化をサポートする**
  - repa **と違い、**Haskell **以外のコードを生成する**; GPU, LLVM IR,..

分散プログラミング

- distributed-process a.k.a. Cloud Haskell
  - Haskell に分散プログラミングを導入するフレームワーク
  - 実行モデルは Erlang, Elixir の軽量プロセスに近い
  - とある暗号通貨の実装に使われているらしい

マルチスレッドプロファイリング

- ThreadScope
  - Haskell の実行時システムのログを可視化してくれるツール
  - 各 OS 向けにバイナリ配布されているので導入しやすい



まとめ

- CPU の性能を引き出すには並列・並行が必要になってくる時代
- Haskell は現在並列・並行が得意とされる言語と同等以上に並列・並行の道具が揃っている

補遺: Haskell の並列・並行関連のニュース

1. ApplicativeDo (GHC 8.0)
2. Facebook での成果: -qn オプション (GHC 8.2)
3. GHC の NUMA サポート (GHC 8.2)
4. 暗号通貨 Cardano は Cloud Haskell を使っている？

補遺: 軽量スレッドの消費メモリ (引用)

TSO data structure

[includes/rts/storage/TSO.h] (ghc 8.6)

```
typedef struct StgTSO_ {
   StgHeader           header;
   struct StgTSO_*       _link;
   struct StgTSO_*       global_link;
   struct StgStack_     *stackobj;
   StgWord16           what_next;
   StgWord16           why_blocked;
   StgWord32           flags;
   StgTSOBlockInfo       block_info;
   StgThreadID          id;
   StgWord32           saved_errno;
   StgWord32           dirty;
   struct InCall_*       bound;
   struct Capability_*    cap;
   struct StgTRecHeader_ * trec;
   struct MessageThrowTo_ * blocked_exceptions;
   struct StgBlockingQueue_ *bq;
   StgInt64  alloc_limit;
   StgWord32  tot_stack_size;
} *StgTSOPtr;
```

link to stack object

A TSO object is only ~18words + stack. Lightweight!

References : [S5]

¶

¶Reference: takenobu-hs, "haskell-ghc-illustrated" - https://takenobu-hs.github.io/downloads/haskell_ghc_illustrated.pdf