未来の並列プログラミング言語としてのHaskell

shelarcy

マルチコア化・メニーコア化

- これからも進む
  - ムーアの法則と付き合う
    - *”The free lunch is over.”*
- 進まない（どこかで止まる）
  - マルチスレッドは難しい
  - 特定の役割を持ったプロセッサが乗る

並列プログラミング

- 並行処理（Concurrency）
  - e.g. スレッド
  - 非決定的（Non-Deterministic）で難しい
- 並列処理（Parallelism）
  - 決定的（Deterministic）

Haskell の並列化機能

Low

- OS のネイティブ・スレッド
- Concurrent Haskell (Haskell', GHC)
- Parallel Haskell (GHC)
- Data Parallel Haskell (GHC)

High

GHC の実装

- ✓ SMP 対応

  NUMA 対応、SIMD 命令対応（Future？）

- ✓ Parallel GC (GHC 6.10.1)
- ▲ Nested Data Parallelism
  (Data Parallel Haskell)　(GHC 6.10.1~)

  並列プロファイラ (GHC 6.12？)

  Concurrency as a Library (Future？)

Data Parallel Haskell (DPH)

- 並列配列（[::]、PArray？）
  - 自動並列化
  - 正格評価
  - ベクトル化（フラット化）
    - 入れ子にできる (e.g. [: [: Double :]:])
    - ユーザー定義のデータ型を使用可能

DPH プログラム例

- libraries/dph/examples/barnesHut/
  - QuickSort
  - Quick-Hull
  - Barnes-Hut
  - etc ….
- Data Parallel Physics Engine
  (Google Summer of Code (GSoC) 2008)

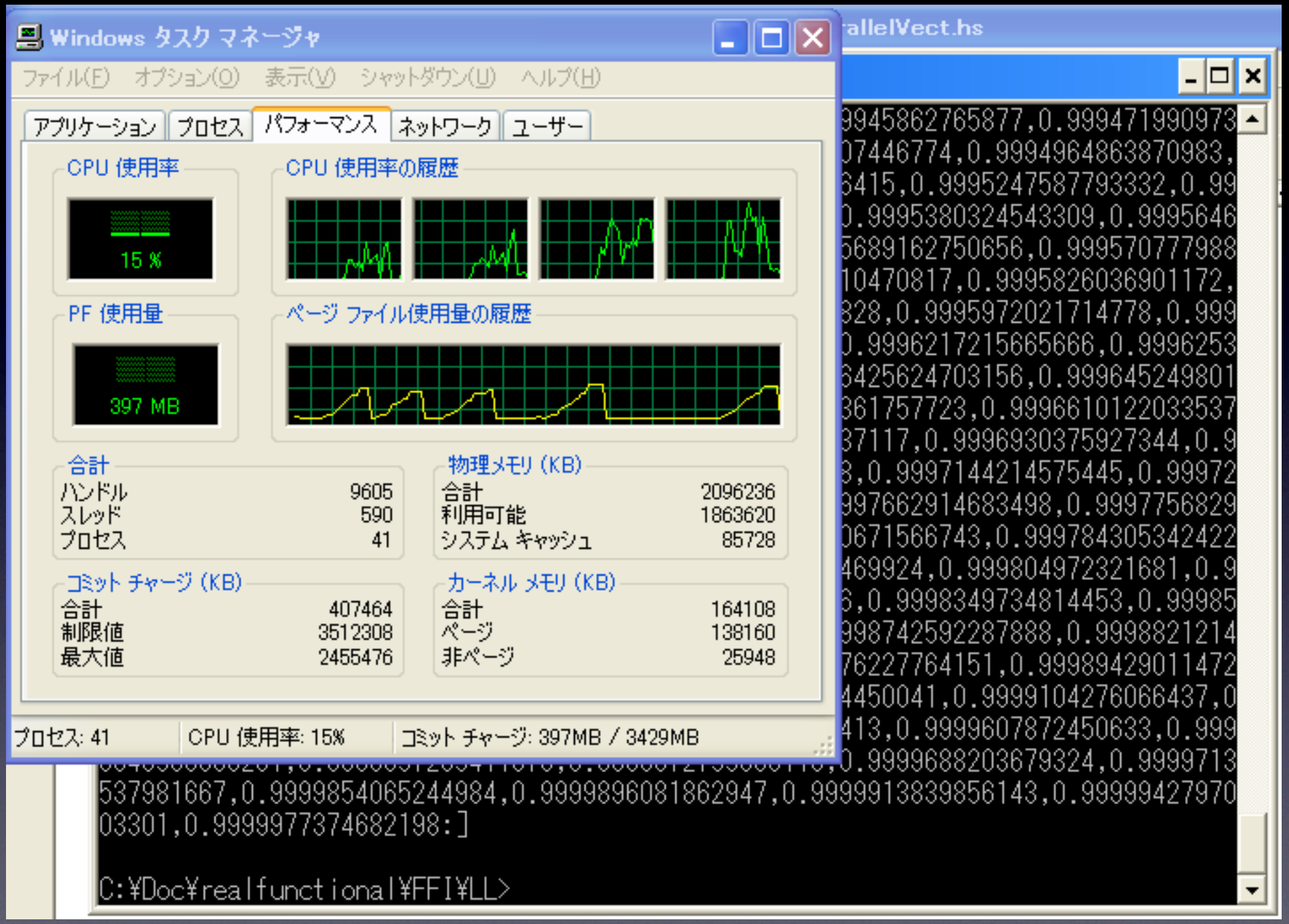
QuickSort の実行例

```
import System.Random
import QSortVect

import GHC.PArr (toP)
import GHC.Conc (numCapabilities)
import Data.Array.Parallel.Unlifted.Distributed (setGang)

main = do
    setGang numCapabilities
    g <- getStdGen
    print $ qsortVect' $ toP $ take 170000 $ randoms g
```

```
{-# LANGUAGE PArr #-}
{-# OPTIONS_GHC -fvectorise #-}
{-# OPTIONS_GHC -fno-spec-constr-count #-}
module QSortVect (qsortVect, qsortVect') where

import Data.Array.Parallel.Prelude
import Data.Array.Parallel.Prelude.Double
import qualified Data.Array.Parallel.Prelude.Int as I

import qualified Prelude

qsortVect:: PArray Double -> PArray Double
qsortVect xs = toPArrayP  (qsortVect' (fromPArrayP xs))

qsortVect':: [: Double :] -> [: Double :]
qsortVect' xs | lengthP xs I.<=  1 = xs
              | otherwise      = qsortVect' [:x | x <- xs, x < p:] +:+
                                  [:x | x <- xs, x == p:] +:+
                                 qsortVect' [:x | x <- xs, x > p:]
        where p =  (xs !: (lengthP xs `I.div` 2))
```

QuickSort の実行例

```
$ ghc -Odph -fdph-par ParallelVect.hs
  --make (または GSortVect.hs -package dph-par)

  -threaded

$ ./ParallelVect +RTS -N4
```


Windows タスク マネージャ
ファイル(F) オプション(O) 表示(V) シャットダウン(U) ヘルプ(H)
アプリケーション プロセス パフォーマンス ネットワーク ユーザー
CPU 使用率
15 %
CPU 使用率の履歴
PF 使用量
397 MB
ページ ファイル使用量の履歴
合計
ハンドル 9605
スレッド 590
プロセス 41
物理メモリ (KB)
合計 2096236
利用可能 1863620
システム キャッシュ 85728
コミット チャージ (KB)
合計 407464
制限値 3512308
最大値 2455476
カーネル メモリ (KB)
合計 164108
ページ 138160
非ページ 25948
プロセス: 41
CPU 使用率: 15%
コミット チャージ: 397MB / 3429MB
C:¥Doc¥realfunctional¥FFI¥LL>

型の使用例

```
cross  = [: distance p line | p <- points :]
packed = [: p | (p,c) <- zipP points cross
              , c > 0.0 :]
distance :: Point -> Line -> Double

data Point = Point Double Double
data Line  = Line  Point Point
```

(QH.hs, Types.hs)

```
data BHTree = BHT Double Double Double
                -- root mass point
                  [:BHTree:]
```

(BarnersHutVect.hs)

DPH の今後の課題

- 機能の拡充
  - Prelude + Data.List に比べて機能が不足
- 最適化
- 様々なアーキテクチャへの対応
  - NUMA？ SIMD？ CUDA？ 分散処理？
  - GHC 側での対応？

GHC のカスタマイズ

- 自分のバージョンを作る
  - 分散バージョン管理を活用 (darcs, git)
  - GHC API (GHC as a Library)
- プラグインで振る舞いを変える
  - GHC Plugins

GHC Plugins

- プラグイン + コードへの注釈（Annotation）
  - GHC の生成するコードを動的に変更
  - cf. Anglo Haskell 2008
  - GPU のコード生成 +
    GPU を呼ぶコードの生成
- Compiler plugins for GHC
  (GSoC 2008)

GHC Plugins の利点

- 最適化戦略の完全制御が可能
- ユーザーの手で機能を追加できる
  - GHC を再ビルドしなくてよい
  - GHC の対応を待つ必要がない

プラグインの作り方

```
-- このモジュール名は階層化されるかも
import Plugins

import GHC.Prim({-# PHASE … #-})
import GHC.Phases({-# PHASE … #-})

plugin :: Plugin
plugin = defaultPlugin {
    ……
  }
```

GHC Plugins の例

- http://code.haskell.org/cse-ghc-plugin/
  - 少し古い？
- :: (BLOGGABLE A) => A -> IO ()
  - 開発者の blog

GHC Plugins の現状

- GSoC の目的はあくまでデザイン
  - マージもブランチもなし
    - 古いのであれば CVS に (pluggable-branch)
  - 開発者の手元にはコードはある

GHC Plugins
今後の開発

- 細々と続ける？
- Ph.D. student のプロジェクト？
- Microsoft Research のインターン？
- （もしあれば）次の年の GSoC？

まとめ

- DPH
  - 高レベルの並列化
- GHC Plugins
  - 柔軟なコード生成
- DPH + GHC Plugins
  - 理想の並列化環境かもしれない

その他のプロジェクト

- Eden
- Mobile Haskell
- Distributed Haskell
- Grid Parallel Haskell
- etc .....

ご清聴ありがとうございました