Manticore
A heterogeneous parallel language

全体ミーティング (2010/05/26)

秋山 茂樹

Manticore

- 様々な並列処理サポートを備えた関数型言語
  - Standard ML ベース
    - 静的型付け
    - 正格評価
  - 参照型はなし
  - 並列処理サポート
    - explicit parallel threads
    - parallel arrays, tuples, bindings, case

並列処理サポート

- Concurrent ML
  - 同期メッセージパッシング
  - spawn, channel, send, recv, etc...
- parallel arrays
  - map, reduce などを並列に行う
- parallel tuples, parallel bindings, parallel case

概要

- Manticore のランタイム（主にスレッド周り）を紹介
  - Fiber
  - Virtual Processor
  - Heap
  - Infrastructure for Nested Scheduler
    - 例: Round-Robin, Gang Scheduling
  - GC や CML, parallel arrays, etc については扱わない

Fiber

- Fiber
  - 軽量スレッド
  - first-class continuation で表現

```
type fiber = unit cont
fun fiber f = let
      cont k () = ( f () ; stop () )
      in k end
```

1. f () を実行
2. スケジューラにスレッド停止を通知

- Fiber Local Storage (FLS)
  - fiber ごとに存在する読み書き可能な領域

Preemption

- Fiber のプリエンプション
  - POSIX の signal を用いて定期的に発生させる
    - シグナルハンドラでフラグを立てる
    - 安全な状態（allocation 時）になったらコンテキストスイッチ
- スケジューラについては後述

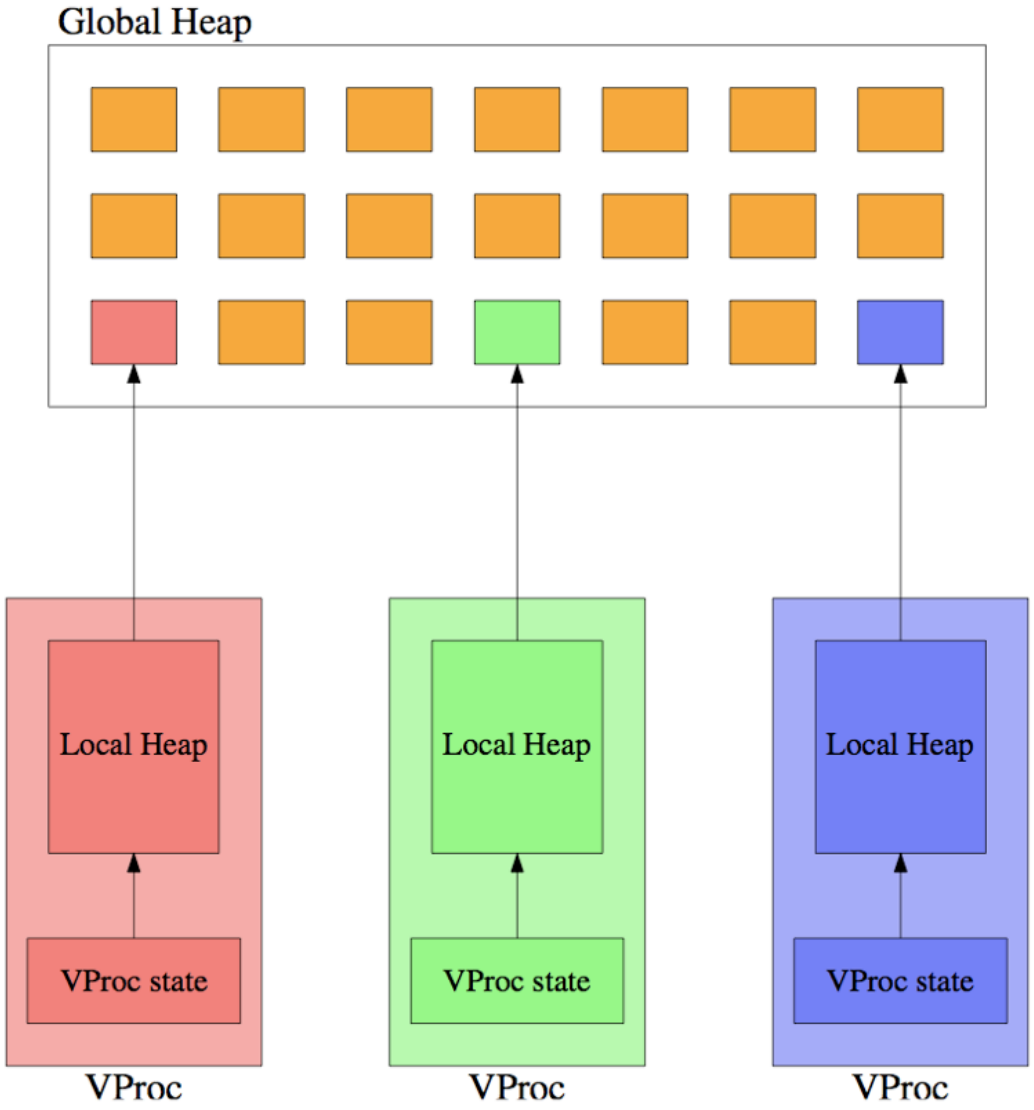
Virtual Processor (vproc)

- OS thread と対応する仮想プロセッサ
  - この上で fiber が並行に実行される

- 保持するデータ
  - ローカルヒープ
  - Ready Queue, Scheduler Stack
    - スケジューラにより操作される（後述）

Heap

- VProc ごとのヒープ
- グローバルヒープ



Infrastructure for Nested Scheduler

- スレッドのスケジューリングポリシーを定義するための枠組み
- なぜプログラマブルな形で提供するのか？
  - 並列化機能ごとにスケジューリングアルゴリズムを変えたい
  - 将来的にはユーザによる定義も可能に？

Scheduling Operations

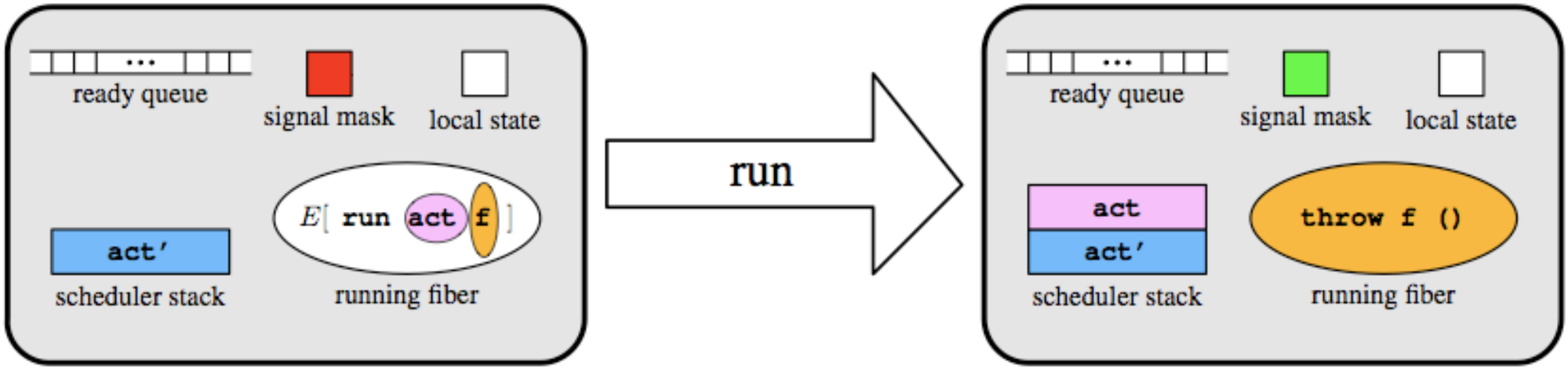
- Primitive
  - datatype signal = STOP | PREEMPT of fiber
  - type action = signal -> void
  - val run : (action * fiber) -> void
  - val forward : signal -> void
  - Ready Queue に対する操作
- Operations
  - spawn, stop, preempt, yield, migrateTo
- Utility
  - mask, unmask, atomicYield, concurrent queue

スケジューラの表現

- datatype signal = STOP | PREEMPT of fiber
  - スケジューラが受け取るシグナル
    - STOP: スレッドを終了
    - PREEMPT: スレッドをサスペンドし、コンテキストスイッチ
- type action = signal -> void
  - スケジューラの動作を表現
  - この関数を定義する

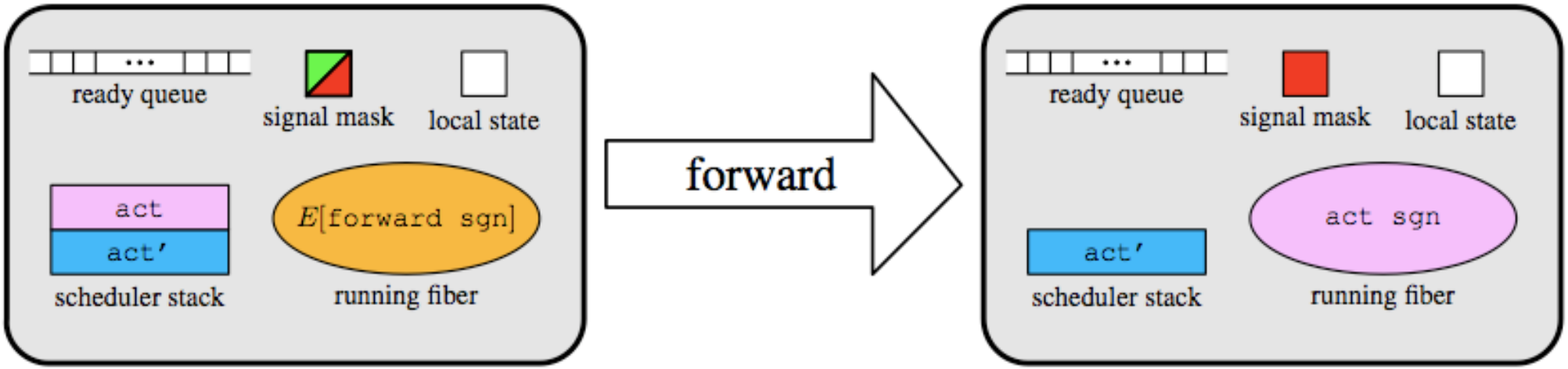
Primitive

- fiber の実行
  - run ： (action * fiber) -> void
  - 指定したアクションを scheduler stack に積み、fiber を起動（再開）



Primitive

- スケジューラの実行
  - forward ： signal -> void
  - scheduler stack からアクションを取り出して実行



Ready Queue

- enq ： fiber -> unit
  - 現在の vproc のキューに fiber を追加
- deq ： unit -> fiber
  - 現在の vproc のキューから fiber を取り出す
- enqOnVP ： (vproc * fiber) -> unit
  - 他の vproc のキューに fiber を追加

注: deqOnVP は存在しない
（コンテキストスイッチのコストを削減するため）

Spawn

- スレッドの生成

```
fun spawn f =
    enq (fiber (fn () =>
                    setFls (newFls ());
                    f () ))
```

Fiber Local Storage の初期化

1. FLS を初期化し、f () を実行する fiber を生成
2. fiber を Ready Queue に入れる

Stop, Yield

- スレッドを終了させる

```
fun stop () = forward STOP
```

スケジューラにスレッド停止を通知

- スレッドを中断させる

```
fun yield () =
    let cont k x = x in
        forward (PREEMPT k)
    end
```

現時点での continuation を取得

スケジューラにプリエンプション発生を通知

Migration

- 実行中のスレッドを他の vproc に移動

```
fun migrateTo vp =
    let
        val  fls = getFls ()
        cont k x = ( setFls fls;  x)
    in
        enqOnVP (vp, k);
        stop ()
    end
```

FLS を保存

FLS を戻して続きを実行する continuation

他の vproc に fiber を渡して停止

Utility Functions

- atomicYield ： unit -> void
  - scheduler stack からスケジューラを取り出して実行
  - スケジューラ内で別のスケジューラに処理を受け渡す場合に使う
- concurrent queue
  - non-blocking queue
  - emptyQ, addQ, remQ

例: Round-Robin

- Uniprocessor 用

val roundRobin : signal -> void

```
cont roundRobin sgn =
    case sgn of
      STOP => dispatch ()
    | PREEMPT k =>
        let val fls = getFls ()
            cont k' () = ( setFls fls ; throw k () )
        in  enq k' ; dispatch ()  end
cont dispatch () = run (roundRobin, deq ())
```

continuation を作って Ready Queue に入れる

スケジューラを Round-Robin としてスレッドをディスパッチ

Load Balancing

- vproc が idle 状態になったとき
  - 他の vproc に定期的に thief スレッドを生成
  - thief スレッドは生成先の vproc を調べ、十分な load があるなら、スレッドを元の vproc に移動させる

例: Gang Scheduling

- 関連するスレッドをなるべく同時並行的に実行されるようにスケジュールする
  - parallel arrays にて利用
- Gang Scheduling 用の future の生成

```
fun future thunk =
    let val fut = newFutureCell thunk in
        addQ (getReadyQueue (),
            fiber (fn () => futureStealAndEval fut) ) ;
        fut
    end
```

自前の Ready Queue を用意し、そこに fiber を入れる

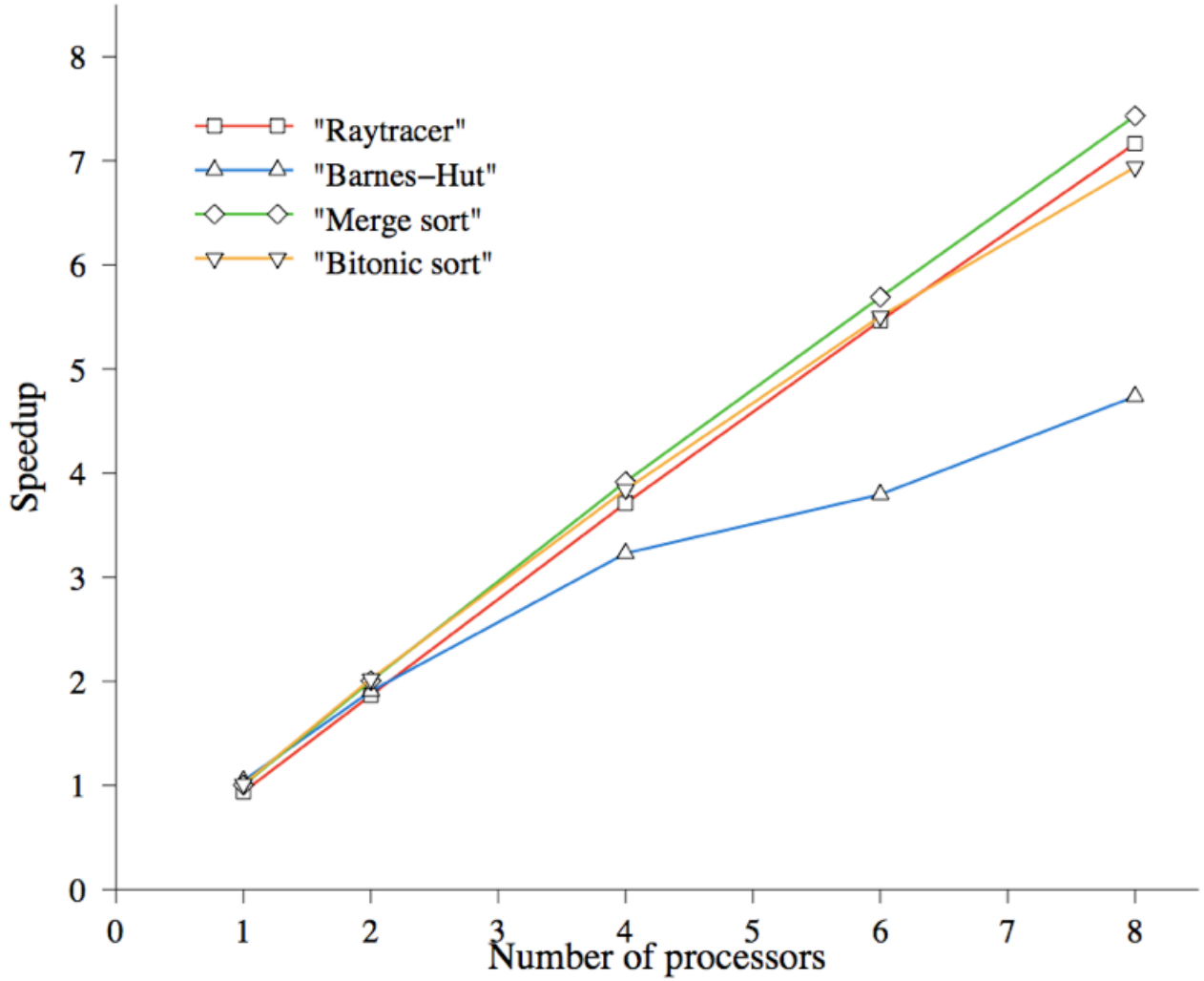
```
val gq = initGanQueue ()
val fls = getFls ()
cont gsAction sgn = let
    cont dispatch () = case remQ gq of
        NONE => ( atomicYield () ; dispatch () )
      | SOME k => ( setFls fls ; run (gsAction, k) )
    in
      case sgn of
          STOP => dispatch ()
        | PREEMPT k =>
            ( addQ (gq, k) ; atomicYield () ; dispatch () )
    end
```

キューに fiber がなければ既存のスケジューラに処理を任せる

キューに fiber があればそれを実行する(スケジューラは Gang)

プリエンプションが発生したときはキューに fiber を加える

Evaluation


"Raytracer"
"Barnes–Hut"
"Merge sort"
"Bitonic sort"
Speedup
Number of processors
0
1
2
3
4
5
6
7
8

References

- Matthew Fluet et al. "Programming in Manticore, a Heterogeneous Parallel Functional Language"
- Matthew Fluet et al. "A scheduling framework for general-purpose parallel languages" (ICFP '08)
- Matthew Fluet et al. "Manticore: A heterogeneous parallel language" (DAMP '07)