スティール評価法を備えたPaiLispシステムの実現と評価

1L－8

川本　真一　伊藤　貴康
東北大学　情報科学研究科

1. はじめに

PaiLispは様々な並列構文を持つ並列Schemeである．従来のPaiLispの処理系は並列構文の評価法としてETC (Eager Task Creation)を用いていたが，ETCはプロセス過剰生成問題があって効率が悪い．この問題に対し，並列構文の効率的な評価法としてスティール評価法[1]が提案されている．マルチスレッド機構に基づくPaiLispインタプリタPaiLisp/MT[2]にスティール評価法を導入し効率の良い処理系を設計・試作したので，その実現と実験結果について報告する．

2. PaiLispとその処理系

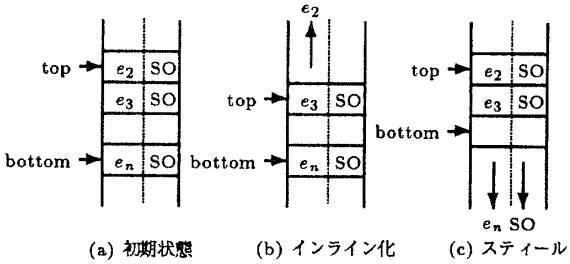
PaiLispは **pcall, pbegin, plet, par-and, pcond future** などの様々な並列構文を持つ共有メモリ型並列Schemeである．**pcall** 構文は関数適用式の並列化構文で，(**pcall** $f\ e_1 \cdots e_n$) のように用いられ，式 $e_1, \cdots, e_n$ を並列に評価し，すべての評価が終了した後，式 f を評価し，得られた関数を $e_1, \cdots, e_n$ の値に適用する．PaiLispのインタプリタであるPaiLisp/MT[2]の並列構文は主としてETCで実現され，**future** 構文はMul-Tで導入されたLTC (Lazy Task Creation)を用いて実現されている．**future** 構文とそのLTC実現は強力であるが，効率的な並列プログラムの作成が難しいという問題がある[3]．ETCは式 (**pcall** $f\ e_1 \cdots e_n$) の評価に際し，式 $e_1, \cdots, e_n$ をそれぞれ評価する n 個のプロセスを生成する．このため頻繁に並列構文が呼び出されると多くのプロセスが生成され効率が悪いという**プロセス過剰生成問題**が発生する．以下のpfibプログラムを

```
(define (pfib n)
  (if (zero? n) n
      (pcall + (pfib (- n 1)) (pfib (- n 1)))))
```

(pfib 20) として呼び出すと，ETCでは21890個ものプロセスが生成され，逐次プログラムより効率が悪い(表1参照)．この問題に対し，**pcall** などの構造化並列構文の効率的評価法としてスティール評価法 (Steal Based Evaluation, SBE)[1] が提案されている．

3. スティール評価法

スティール評価法は並列構文が出現してもプロセスを生成せず逐次的に評価を進め，他のプロセッサが空き状態になると初めてプロセスを生成するので，ETCに比べてプロセスの生成を抑制できる効率の良い方法である．スティール評価法では，各プロセッサはSST (Stealable STack)と呼ばれるスタックを持つ．SSTは2つのポインタtopとbottomで管理される．式 (**pcall** $f\ e_1 \cdots e_n$) を評価したプロセッサはホームプロセッサと呼ばれ，式 $e_1, \cdots, e_n$ を評価するプロセスを生成する代りに，式とその式を評価するために必要な環境などの情報からなる共有オブジェクト (SO)を組にして，その各組を $e_n, \cdots, e_2$ の順に自分のSSTのtopに加える(図1(a)参照)．そして，式 e_1 を評価しその評価が終了すると，SSTのtopから式を取り出し(図1(b)参照)それを評価する．ホームプロセッサはこの操作を繰返し行い，式 $e_1, \cdots, e_n$ すべての評価が終了した後，f の評価と適用を行う．この逐次評価をインライン化と呼ぶ．



(a) 初期状態　(b) インライン化　(c) スティール

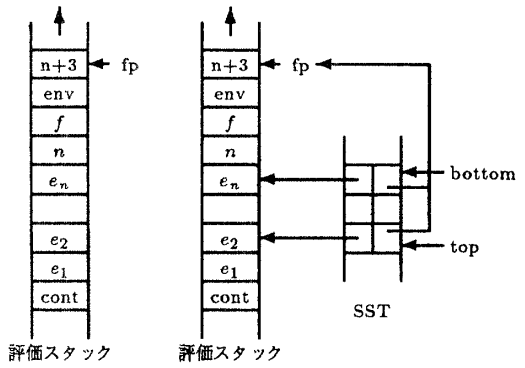
図1: ホームプロセッサのSSTとその操作

ホームプロセッサ以外のプロセッサが空き状態となれば，そのプロセッサはホームプロセッサのSSTのbottomから式と共有オブジェクトの組を取り出し(図1(c)参照)その式の評価を行う(これを**スティール**という)．ホームプロセッサのSSTのbottomから最も古い式を優先的にスティールするのは，最も処理コストの大きな式をスティールして並列評価することで，スティールの回数すなわちプロセス生成の回数を減らすためである．

4. PaiLisp/MTにおけるスティール評価法の実現

マルチスレッド機構を用いたPaiLispインタプリタPaiLisp/MT[2]に上述のスティール評価法を導入した効率の良いインタプリタの実現法について説明する．スティール評価法を効率良く実現するには，SST，共有オブジェクト，スティール操作の3つを如何に実現するかが重要となる．スティール評価法では，並列構文の実行においてスティールによるプロセス生成が抑制され，式の大部分はインライン化されて実行されるという特性を考慮し，インライン化のコストを小さくする方針の下，SSTと共有オブジェクトの実装を以下のように工夫し効率よく実現した．

Implementation and Evaluation of
Steal Based Evaluation in PaiLisp system
Shin-ichi Kawamoto Takayasu Ito
Department of Computer and Mathematical Sciences,
Graduate School of Information Sciences,
Tohoku University



(a) フレーム構造　(b) SST とその構成

図 2: PaiLisp/MT のフレーム構造と SST

関数適用式 ($f\ e_1 \cdots e_n$) の評価を例として考えると, PaiLisp/MT はまず評価スタックに図 2(a) のようなフレームを構成する. fp は現在のフレームポインタ, cont は残りの計算, env はこの式を評価する環境を示す. フレームの一番上にはこれから評価すべき引数式のフレーム上の位置 (オフセット) が格納されている. フレームを生成した後, そのフレームのオフセットが差し示す引数式を取り出し (下記リスト 2, 3 行目), その式を評価して得られた値をその引数式が格納されていた位置に書き戻し (5 行目), オフセットを更新する (6 行目) という処理を繰り返す.

```
1: while ( n-- ) {
2:    sp  = fp + current_frame(0);
3:    exp = current_frame(sp);
4:    env = current_frame(1);
5:    current_frame(sp) = eval(exp, env);
6:    current_frame(0) = sp - 1; }
```

このようにフレームには式 $e_1, \cdots, e_n$ を評価するために必要なすべての情報が含まれるので, 共有オブジェクトを生成する代りにフレームポインタを保存する. また式 $e_2, \cdots, e_n$ の代りにその式のオフセットを保存する. ホームプロセッサによる式 (**pcall** $f\ e_1 \cdots e_n$) の評価は, 式 $e_1, \cdots, e_n$ を評価スタックに push し, $e_n, \cdots, e_2$ 各式のオフセットと現在のフレームポインタの組を SST の top に追加し (図 2(b) 参照), 式 e_1 の評価を行ったあと次のインライン化の処理を行う.

```
1: while ( --n > 0 ) {
2:    mutex_lock(SST_mutex);
3:    if ( top > bottom) ) {
4:       top = top -1;
5:       mutex_unlock(SST_mutex);
6:       sp  = current_frame(0);
7:       exp = current_frame(sp);
8:       env = current_frame(1);
9:       current_frame(sp) = eval(exp, env);
10:      current_frame(0) = sp - 1 ;
11:   } else {
12:      mutex_unlock(SST_mutex);
13:      wait_children();  }}
```

この処理は関数適用式の引数式評価処理に, SST の排他アクセスのための mutex_lock の導入と (2, 5, 12 行目), SST の top と bottom を比較して引数式のスティールの有無を確認する (3 行目) という最小限の処理を追加して効率良く実現されている. 引数式がスティールされ評価すべき式が存在しない場合はスティールされた式の評価の終了を待つ (13 行目). 一方, スティールは SST の bottom からオフセットとフレームポインタをまず取り出し, それらを用いてホームプロセッサのフレームから式と環境などの情報を取り出して評価する.

5. スティール評価法の実験

DEC7000 上で PaiLisp/MT のスティール評価法の実験を行った. インライン化のオーバヘッドは並列構文 1 つにつき約 4.8[μsec] で, この値は mutex_lock を一回実行するコストとほぼ一致し, mutex_lock を用いた実装ではこれが限界である. スティールのコストは 50[μsec] と ETC のプロセス生成コスト 45[μsec] より若干大きいが, 大部分の式はインライン化されるのでその影響は小さい. 次に, **pcall** によって並列化されたプログラムをプロセッサ 6 台の下で評価した結果を表 1 に示す. カッコ内は生成されたプロセス数を示す.

表 1: スティール評価法の実験結果 [sec]

プログラム	SBE	ETC	逐次
(fib 20)	0.07(72)	0.51(21890)	0.32(0)
(tarai 9 4 0)	0.19(406)	0.96(41421)	0.88(0)
(sum v)	0.11(42)	0.53(19998)	0.51(0)
(queen 8)	0.20(78)	0.39(11016)	1.12(0)
(qsort 1)	0.23(83)	0.29(2556)	1.04(0)

fib, tarai, sum の各プログラムを ETC の下での評価すると, 大量のプロセスが生成され逐次よりも効率が悪いが, スティール評価法 (SBE) の場合はプロセス生成数が大幅に削減され, 逐次の約 1/4 の実行時間で効率良く実行されている. queen や qsort は, ETC でも逐次より速いが, スティール評価法ではさらに効率が良く, 特に queen の場合は逐次の約 1/5.6 の実行時間となっており, 非常に効率が良い.

6. おわりに

PaiLisp/MT におけるスティール評価法の実現とその実験について述べた. この方法はインタプリタに対しては十分効率的である. コンパイラの場合は新たな工夫が必要でありこれについては別の機会に報告したい.

参考文献

[1] T.Ito.: Efficient evaluation strategies for structured concurrency constructs in parallel Scheme systems, LNCS 1068, pp.22-52, Springer (1996).

[2] S.Kawamoto, T.Ito.: Multi-threaded PaiLisp with granularity adaptive parallel execution, LNCS 907, pp.94-120, Springer (1995).

[3] 川本真一, 伊藤貴康: Scheme プログラムの自動並列化とスティール評価法による実行, 情報処理学会プログラミング研究会, PRO5, (1996)